Elektronik

Sonderheft III

Preis DM 18.–

Mikroprozessoren

Produkte - Anwendungen - Tendenzen

SIEMENS

Systematik und Kontinuität für die achtziger Jahre

SMP 80 ist ein langfristig angelegtes, zukunftsicheres Mikrocomputer-Baugruppensystem von professioneller Qualität im Karten-Format 100 mm x 160 mm. Der große Vorteil für den Geräte-Entwickler liegt in der Vielfalt der Funktionseinheiten des Systems:

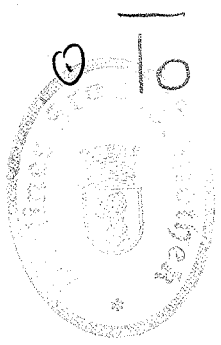
- Zentraleinheiten mit den Prozessoren 8080 und 8085 A, DMA-fähig, und mit zusätzlichem Arithmetik-Prozessor
- analoge und digitale Ein- und Ausgabe-Baugruppen unterschiedlichster Konzeptionen
- ein breites Spektrum von preisgünstigen RAM- und ROM/EPROM-Speicherbaugruppen
- dazugehörige Aufbau- und Testhilfen sowie Netzteile.

Als Systemsoftware stehen Monitorprogramm und BASIC-Interpreter zur Verfügung.
An der Weiterentwicklung von SMP 80-Hard- und Software wird ständig gearbeitet.

Die Entwicklung der Anwendersoftware unterstützt Siemens mit hochkomfortablen Programmierplätzen, mit einer Programmbibliothek, Programmierkursen und Fachberatung.
Ausführliche Informationen schicken wir Ihnen gern. Schreiben Sie an die Siemens AG, Bereich Bauelemente/ZVW 104, Postfach 103, D-8000 München, Stichwort »SMP 80«.

SMP 80 – das programmierfertige Mikrocomputer-Baugruppensystem von Siemens

neu '79



Vorwort

Die schwierige Zeit des Einstiegs in die Mikroprozessortechnik liegt hinter uns. Die Grundlagen dieses Gebietes gehören mittlerweile zum Allgemeinwissen des Ingenieurs. Doch auch der µP-Alltag hat seine Tücken. Immer noch ist das Programmieren viel zu umständlich, ist Software noch keine Ware wie jede andere.

Es gibt keinen vorgezeichneten Idealweg, den die Mikroprozessortechnik einschlagen wird. Aber es gibt Vorschläge, Alternativen und Anwendungen, die gangbare Wege aufzeigen. Einige davon finden Sie in diesem Heft – sie stammen von kompetenten Autoren und wurden zum Großteil in der Fachzeitschrift ELEKTRONIK veröffentlicht.

Da sich nicht nur Techniker im µP-Zeitalter einer veränderten Situation gegenüber sehen, wurden auch kaufmännische Gesichtspunkte berücksichtigt. Nicht zuletzt sind neue Produkte ein Thema, mit dem sich Techniker und Kaufmann auseinandersetzen müssen. Die wichtigsten finden Sie ebenfalls in diesem Heft.

An den Anfang haben wir einen Beitrag gestellt, der schon im ersten Mikroprozessor-Sonderheft erschienen ist. Er wurde auf den neuesten Stand gebracht und soll in erster Linie zur Begriffserklärung dienen. Diese Einführung bietet aber auch all jenen die Gelegenheit, sich einen ersten Überblick zu verschaffen, die keine Vorkenntnisse auf dem Mikroprozessorgebiet mitbringen.

Die Redaktion

Die Zeit ist reif für 16 bit!

Obwohl es 16-bit-Mikroprozessoren schon seit einigen Jahren gibt, treten sie erst jetzt in breiter Front ihren Siegeszug an. Auf manchen Gebieten werden sie die 8-bit-Typen sicherlich ablösen, obwohl deren Zeit keineswegs ganz abgelaufen ist.

Beiträge über die jüngsten µP-Modelle 8086, 68000, Z8000 und 9440, die auch technologisch die absolute Spitze darstellen, finden Sie in diesem Heft.

Inserentenverzeichnis Mikroprozessor III

1979

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

Kurze Produktbesprechungen

INHALT

Die Entscheidung

für das 16-bit Mikrocomputer-system MCS-86

Es gibt, je nach Ihrem ganz individuellen Anwendungsfall, eine Reihe guter Gründe, sich für das 16-bit-Mikrocomputersystem MCS-86 zu entscheiden. Die folgenden sieben Argumente sind jedoch so entscheidend, daß sie im Grunde immer zutreffen.

Lieferbarkeit

Alle Komponenten sowie der komplette SDK-86 Prototyp-Bausatz mit allen Peripherieschaltkreisen und ausführlicher Dokumentation sind lieferbar.

Dokumentation

Den Einstieg in die 16-bit-Technik des MCS-86-Systems haben wir Ihnen durch eine ausführliche, logisch aufgebaute Dokumentation erleichtert. Alle INTEL-Mikrocomputersysteme sind nicht zuletzt wegen ihres excellenten Begleitmaterials weltweit zum Industrie-Standard geworden.

Hardware

Das Konzept des MCS-86-Mikrocomputersystems baut auf den Systemen 8080A und 8085 auf.

Die 8086 hat alle Eigenschaften der 8085-CPU und zusätzlich folgende Merkmale:

- 16-bit-CPU
- 5- bzw. 8-MHz-Clock
- direkte Adressierung von 1 Megabyte RAM und von 65 Ein/Ausgabe-Kanälen
- Assemblersprache kompatibel zu 8080/8085
- 24 Operanden Adress-Modus
- Bit-, Byte-, Wort- und Block-Operationen
- 8- und 16-bit-Arithmetik, binär oder dezimal, einschließlich Multiplikation und Division.
- MULTIBUS™ -kompatible System-Schnittstelle
- vier 16-bit-Datenregister, auch 8-bit adressierbar
- zwei 16-bit-Basisregister, zwei 16-bit-Index-Register
- zweite Registerbank bestehend aus vier Segment Registern für die Erweiterung des Adressbereichs auf 1 Megabyte.
- Bus-Interface-Einheit mit 6 byte FIFO-Register „fetched-ahead-instruction-queue".

Besonders hervorzuheben sind Eigenschaften, die bisher nur in der Großcomputer-Technik zu finden waren:

- „re-entrant"-Programme
- dynamisch verschiebbare – positionsunabhängige – Programme.
- komplexe String-Operationen.

Software

Die Software zum MCS-86 ist prinzipiell identisch und kompatibel mit der Software für die INTEL-Systeme MCS-80 und MCS-85 und damit allen 8080A/8085-Anwendern vertraut.

- PL/M-86, INTELs höhere Programmiersprache
- ASM-86, Macro-Assembler, Linker, Locator
- RMX-86, Mikrocomputer-Betriebssystem
- ISIS-II, Disketten-Operating-System
- FORTRAN-ANS
- BASIC-80, höhere Programmiersprache

Kompatibilität

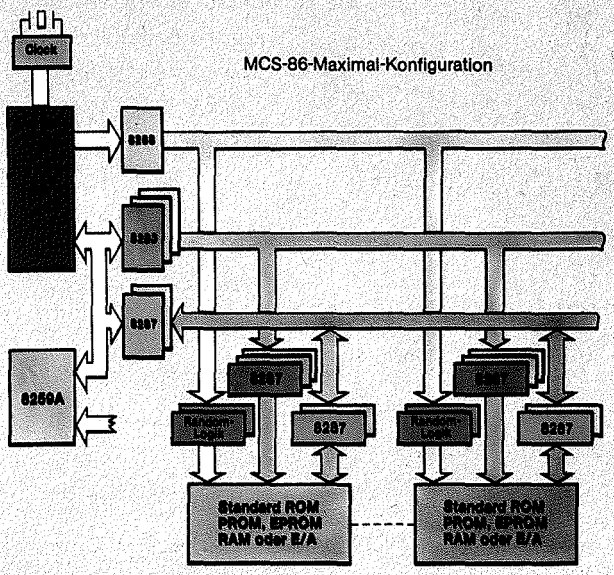
Das MCS-86-System ist Teil der INTEL-Mikrocomputer-Familien und software-kompatibel zum MCS-80/85-System. Mit dem ASM-86-Assembler kann neben der 8086-Programmerstellung auch der 8080/85-Source-Code in den Source-Code des 8086 übertragen werden. Sie können also bereits bestehende Programme in das 16-bit-System übernehmen.

Erweiterungsmöglichkeiten

INTEL liefert zum MCS-86-System eine ganze Palette von Erweiterungsbausteinen wie z. B. RAMs, ROMs, EPROMs und programmierbare Peripherie-Bausteine. Das Diagramm auf dieser Seite zeigt die MCS-86-Maximal-Konfiguration.

Entwicklungs-Hilfsmittel

Erfolgreiche Anwender in aller Welt haben das von INTEL konzipierte Mikrocomputer-Entwicklungssystem INTELLEC® zum internationalen Standard erhoben. Mit INTELLEC-II® steht Ihnen das zur Zeit komfortabelste und zeitsparendste System in drei Ausbaustufen zur Verfügung. Eine erhebliche Vereinfachung bei der Programmkontrolle und Fehlerbeseitigung stellt z. B. der „In-Circuit-Emulator" ICE-86 dar.

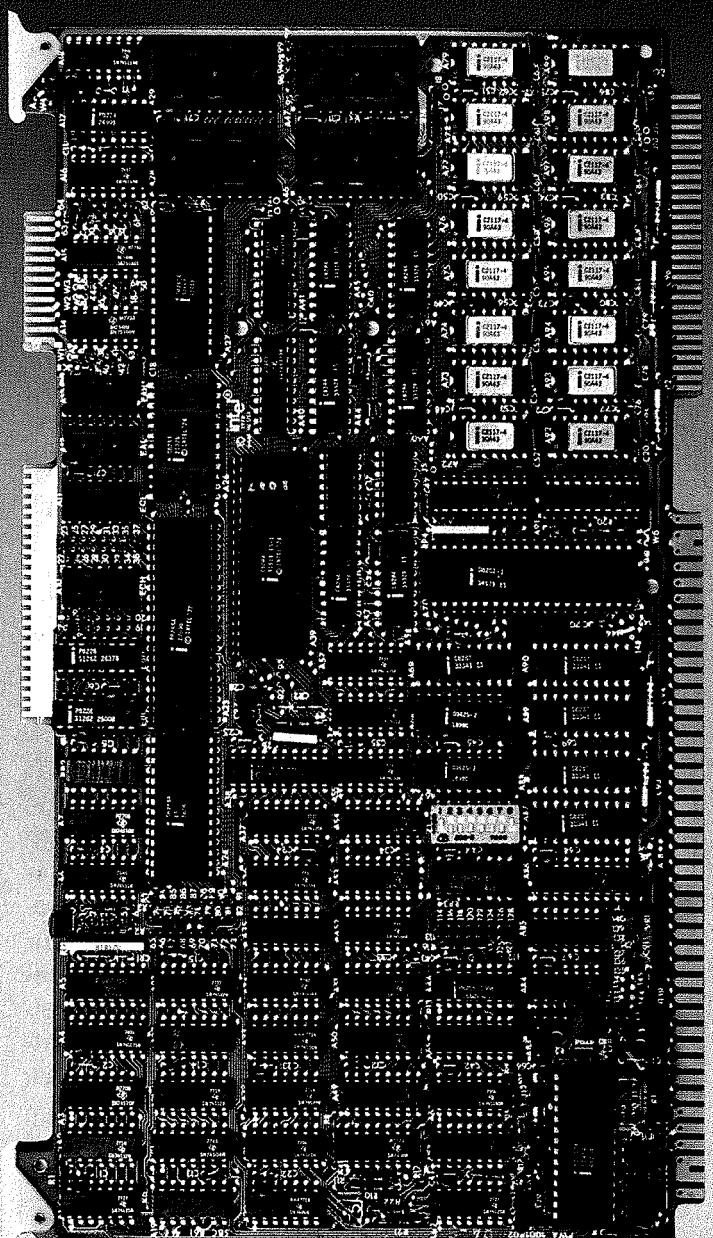


MCS-86-Maximal-Konfiguration

Die Alternative

zum 16-bit Mikrocomputer-system MCS-86

Mit iSBC-86/12 INTELs 16-bit-Einplatinen-Mikrocomputer. Der iSBC-86/12 ist für die Anwender eine Alternative zum MCS-86-System, die aus Zeitgründen oder wegen zu kleiner Produktionsstückzahlen keine eigene Computer-System-Entwicklung beginnen wollen. Der iSBC-86/12 schließt die Lücke zwischen zeitraubender Eigenentwicklung und kostspieligen Minicomputer-Lösungen.



Jeder Anwender hat mit Sicherheit eigene Gründe für den Einsatz des iSBC-86/12-Computers, – einen Vorteil genießen jedoch alle: Der 16-bit-Computer iSBC-86 ist über den MULTIBUS™ mit allen Computer-Platinen des iSBC-Systems von INTEL kompatibel, einschließlich aller Erweiterungsplatinen und Zubehörteile wie, Stromversorgungen, Chassis und Peripherie-Bauelementen.

Den Vorsprung den Sie durch den Wegfall der Hardware-Entwicklung mit Ihrem Produkt am Markt gewinnen, können Sie jedoch noch vergrößern – durch den Einsatz des Mikrocomputer-Entwicklungssystems INTELLEC-II® und der Entwicklungs-Software von INTEL. Weitere Details dazu finden Sie auf der linken Seite dieser Anzeige.

Die Kurzdaten des iSBC-86/12-Computers:

- 16 bit-CPU 8086 in HMOS-Technologie
- 32 KB Dual-Port RAM
- bis 16 KB ROM/EPROM möglich
- maximale Datenspeicherkapazität von 1 Mega-Byte
- 24 programmierbare E/A-Leitungen mit Sockeln für Interface-Bausteine.
- programmierbare, synchrone/asynchrone RS 234C-Serien-Schnittstelle mit softwaregesteuerter Baud-Rate
- Zwei programmierbare 16 bit-BCD/Binär-Zähler.
- 9 bis 65 Vektor-Interrupt-Ebenen
- zusätzlicher Power-Bus und Netzausfall-Interrupt-Logik zur RAM-Datensicherung.
- MULTIBUS™-kompatibel mit allen iSBC-Platinen
- Multimaster-Betrieb
- Software- und Hardware-Entwicklung mit INTELLEC-II®, PL/M-86, BASIC-80, ASM-86 Makroassembler, RMX-86 und ISIS-II.
- ICE-86 „In-Circuit-Emulator" (in Vorbereitung)

Die iSBC-86/12-Computer Platine ist ebenfalls kurzfristig lieferbar.

intel®

INTEL SEMICONDUCTOR GMBH · SEIDLSTRASSE 27 · 8 MÜNCHEN 2 · TEL: 089/55 81 41*

Dipl.-Ing. Reinhard Gößler

Auf dem Weg zur Mikroprozessor-Praxis

Ein zusammenfassender Überblick

Mikroprozessoren und die mit ihnen aufgebauten Mikrocomputer sind die kleinste bisher realisierte Form des Digitalrechners und insbesondere des Prozeßrechners. Sie sind somit prädestiniert, die unzähligen Aufgaben zu übernehmen, die zwar eine programmierbare Rechen- oder Steuerfunktion erfordern, für die aber selbst die frühere Kleinstform des Rechners, der Minicomputer, hinsichtlich Aufwand, Platzbedarf und Preis viel zu hoch gegriffen wäre. Im unteren Grenzfall ist der Mikroprozessor eine schon im Halbleiterwerk nach Kundenwünschen fest programmierte Baugruppe (sie wird in wenigen Jahren zum Bauelement werden), die irgendwo eingebaut wird (OEM). Im oberen Grenzfall steht der Mikroprozessor im Mittelpunkt eines vom Anwender per Programmsprache frei programmierbaren Systems, das den klassischen Minicomputer heute schon teilweise ersetzt. Weil diese Spanne so groß ist und so viele potentielle Anwender erfaßt, legten wir bei diesem Aufsatz das Schwergewicht auf eine systematische Einführung und Erläuterung der wichtigsten Begriffe, um einerseits einen möglichst großen Leserkreis anzusprechen und andererseits eine gemeinsame Verständigungsbasis zu schaffen. Nur so ist sowohl dem interessierten Techniker gedient, der ganz allgemein wissen will, wovon eigentlich die Rede ist, als auch dem Produktverantwortlichen, der sich einen allgemeinen Überblick verschaffen will. Zahlreiche Literaturhinweise, zum großen Teil aus der ELEKTRONIK selbst, ergänzen die Arbeit.

1 Begriffsbestimmungen

Die Erläuterung der zahlreichen anglo-amerikanischen Fachausdrücke erfolgt im Sinnzusammenhang des Textes. Zum großen Teil sind diese Begriffe bereits fest im technischen Sprachgebrauch verankert, und es wurde in diesen Fällen auf eine deutsche Übersetzung verzichtet. In *Tabelle 1* sind häufig auftretende Abkürzungen mit ihrem vollen Wortlaut aufgeführt.

1.1 Mikroprozessor

Unter einem Mikroprozessor (abgekürzt MIP oder auch µP) ist in der Regel eine einzelne integrierte Schaltung im Dual-in-Line-Gehäuse zu verstehen. Sie beinhaltet die Zentraleinheit für ein Prozessor-System und besteht im wesentlichen aus der Arithmetik-Einheit, verschiedenen Arbeitsregistern und der Ablaufsteuerung *(Bild 1)*. In dieser Konfiguration ist der Mikroprozessor aber noch nicht arbeitsfähig. Spezielle Typen (*slices*) lassen sich kaskadieren [39], so daß ein Prozessor beliebiger Wortlänge entsteht.

1.2 Mikrocomputer

Ergänzt man einen Mikroprozessor mit den Elementen Programm-, Arbeitsspeicher und Ein-/Ausgabeeinheit, so entsteht ein Mikrocomputer (µC). Er ist die Minimalkonfiguration eines arbeitsfähigen Systems und kann mit weiteren Bausteinen zu einer leistungsfähigen Anlage erweitert werden. Der Mikrocomputer ist kein Rechner im eigentlichen Sinn und nicht mit einem solchen zu verwechseln! Er ist zur Verarbeitung von Daten ausgelegt, die in einer festen Wortlänge vorliegen. Ein Wort kann entweder als Befehl, Adresse oder Teil von Daten aufgefaßt werden. Die Abgrenzung zum Minicomputer (Kompaktrechner) ist unscharf, weil es bereits Mikrocomputer mit der Leistungsfähigkeit eines Kompaktrechners gibt. Generell kann man sagen, daß ein Minicomputer eine Wortlänge von mindestens 16 bit besitzt und auch umfangreiche arithmetische Operationen in kurzer Zeit ausführen kann [29].

1.3 Mikrocontroller

Einen vereinfacht aufgebauten Mikrocomputer bezeichnet man auch als Mikrocontroller, wenn er für Steuerungsaufgaben konzipiert ist, bei denen z. B. nur einzelne Abläufe aufgerufen werden. Dieses System kommt ohne Arbeitsspeicher aus, da es keine Daten verarbeitet und der vorprogrammierte Befehlsablauf in einem eigenen, vom Anwender programmierten Speicher enthalten ist.

1.4 Einchipper

Durch den ständig wachsenden Integrationsgrad der Bausteine ist es heute wirtschaftlich möglich, die Struktur eines kompletten Mikrocomputers auf einem einzigen Chip unterzubringen. Dieser enthält dann außer der Zentraleinheit den Speicher (RAM und ROM) sowie die Ein/Ausgabe-Verbindungen. Dem Vorteil dieser hohen Komplexität steht als wesentlicher Nachteil entgegen, daß die Architektur keine Erweiterungsmöglichkeiten des Speichers oder der Ports zuläßt. Bei einigen Typen ist in der Entwicklungs- und Prototypenphase die Programmentwicklung schwierig, weil der interne Programmspeicher in der Regel ein maskenprogrammiertes ROM ist und die Simulation mit externen EPROMs einen erhöhten Aufwand erfordert. Ausnahmen hiervon bietet die Intel-8048-Familie, deren Ausbau und Leistungsfähigkeit bei kleinen und mittleren Systemen kaum Wünsche offenläßt.

2 Technologie

Die Fertigung eines Mikroprozessors erfolgt in LSI-Technik, das bedeutet eine Packungsdichte von etwa 500...10 000 Transistorfunktionen pro Chip. Die Größe eines solchen Chips entspricht etwa einem Quadrat mit 4...6 mm Kantenlänge.

Die ersten Mikroprozessor-Bausteine (1971) waren in PMOS-Technologie ausgeführt; sie lassen sich relativ einfach herstellen *(Tabelle 2)*, sind aber nur für geringe Arbeitsgeschwindigkeiten geeignet.

Heute sind andere MOS-Technologien vorherrschend, nämlich NMOS und CMOS; letztere hat eine hohe Störsicherheit bei extrem niedriger Verlustleistung, was beim mobilen Einsatz eines Mikroprozessor-Systems besonders wichtig sein kann. Erwähnt sei noch die SOS-Technologie, die von der Arbeitsgeschwindigkeit her den übrigen MOS-Technologien überlegen ist.

Durch die Vielfalt der Herstellungstechniken (es werden allein 8 verschiedene MOS-Technologien angewandt) entsteht ein Problem, das generell zu wenig beachtet wird: Die Produkte von Zweitlieferanten *(second source)* sind nicht immer austauschbar mit den Modellen anderer Hersteller! Wer auf volle Kompatibilität Wert legt, muß beachten, daß nur solche Bausteine problemlos austauschbar sind, deren Hersteller im Rahmen von Kooperationsverträgen Masken und Fertigungstechniken gemeinsam verwenden. Bei den kaskadierbaren Elementen ist vorwiegend die bipolare Technologie zu finden, da hier Wert auf hohe Arbeitsgeschwindigkeit gelegt wird. In naher Zukunft sind auf diesem Sektor bedeutende Fortschritte durch die ECL-Technik *(Emitter Coupled Logic)* zu erwarten.

Noch ganz in den Anfängen steckt die völlig neu entwickelte I^2L-Technik. Sie erlaubt die höchste Packungsdichte (Tabelle 2) und ermöglicht hohe Arbeitsgeschwindigkeiten bei extrem niedriger Verlustleistung. Diese Technik vereint damit die wesentlichen Vorteile der anderen Technologien, und sie wird aus diesem Grunde rasch an Bedeutung gewinnen [37, 39].

3 Interne Organisation

3.1 Aufbau

Schematischer Aufbau und interne Organisation der Zentraleinheit *(CPU)* sind in Bild 1 dargestellt. Der Datenfluß erfolgt über eine gemeinsame Sammelleitung *(Bus)*, die die Verbindung zwischen den abwechselnd sendenden und empfangenden Datenstationen herstellt. Der Bus enthält ebensoviele Leitungen wie Bits in einem Wort enthalten sind. Beson-

Tabelle 1. Häufig auftretende Abkürzungen und ihr voller Wortlaut

ACC	accumulator
ALPS	advanced logic processing system
ALU	arithmetic logic unit
AR	address register
ASCII	American standard code for inform. interchange
ASTRO	asynchronous/synchronous transmitter/receiver
BORAM	block oriented RAM
CP	control panel
CPU	central processing unit
CRT	cathode-ray tube
DMA	direct memory access
DR	data register
EAROM	electrically alterable ROM
ECL	emitter coupled logic
FIFO	first-in, first-out
FPLA	field programmable logic array
I/O	input/output
IOT	input/output-transfer
IR	instruction register
K	1024mal
LIFO	last-in, first-out
LSB	least significant bit
LSI	large scale integration
µP	microprocessor
MNOS	metal nitride oxide semiconductor
MOS	metal oxide semiconductor
MPS	microprocessor-system
MPU	microprocessor-unit
MSB	most significant bit
PIE	peripheral interface element
PDP	programmed data processing
PC	program counter
PLA	programmable logic array
PL/M	programming language for microprocessors
PL/1	programming language 1
PPS	parallel processing system
RALU	register and arithmetic logic unit
RAM	random access memory
ROM	read only memory
SOS	silicon-on-sapphire
UART	universal asynchronous/synchronous receiver/transmitter
WCS	writable control storage

Tabelle 2. Fertigungsaufwand bei unterschiedlichen Technologien, dargestellt am Beispiel eines Gatters mit vier Eingängen

	Chipfläche pro Gatter (in µm²)	Bauelemente-Typen	Maskenanzahl	Diffusionsvorgänge
Standard-TTL	34 000	3	7	4
CMOS	32 000	3	6	3
PMOS	6 800	2	4	1
NMOS	3 600	2	7	3
I^2L	3 100	1	4	2

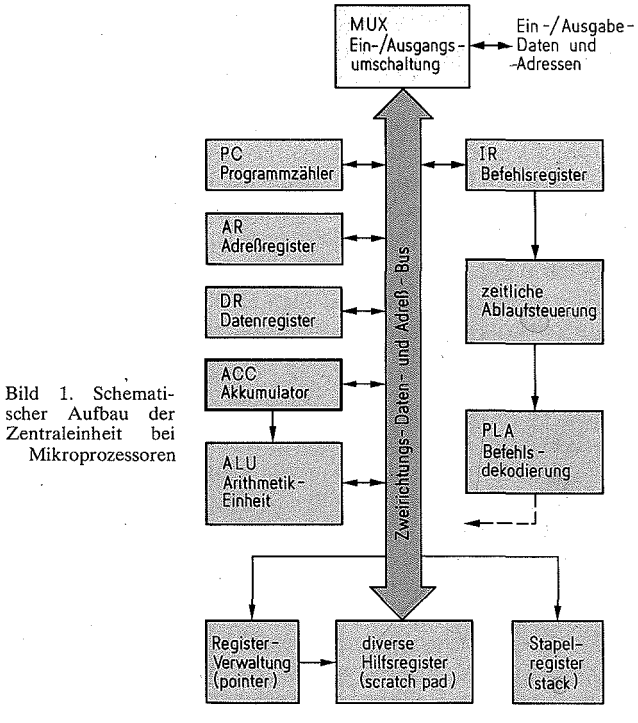
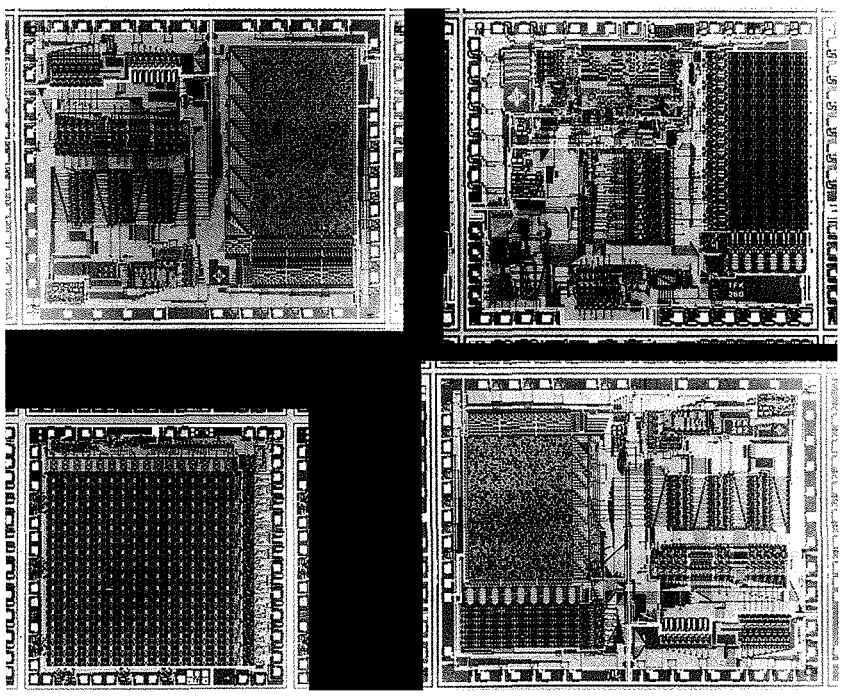


Bild 1. Schematischer Aufbau der Zentraleinheit bei Mikroprozessoren



Der Mikroprozessor CP 3-F von AEG-Telefunken, der erste europäische in PMOS-Technologie, hat mit seinem Konzept Schule gemacht: Der Typ F 8 von Fairchild ist ihm sehr ähnlich.
Das Bild zeigt die Rechen- und Steuereinheit (rechts oben), den Programmspeicher (links oben), den Programm-Datenspeicher (rechts unten) und den Datenspeicher (links unten)

ders bei Systemen mit kleiner Wortlänge ist aus diesem Grunde ein eigener Adreß-Bus mit höherer Bitzahl sinnvoll, um einen größeren Speicherbereich adressieren zu können. Zur Steuerung des Datenverkehrs mit externen Bausteinen (z. B. Speicher oder Interface-Schaltungen) ist ein Multiplexer vorhanden, der normalerweise von internen Steuersignalen gestellt wird. Einige Systeme erlauben aber auch einen externen Eingriff, was wichtig ist bei der Verarbeitung äußerer Programmunterbrechungen (siehe auch Abschnitt 8.4).

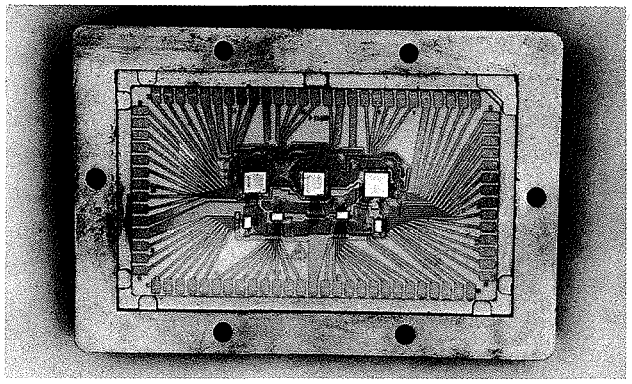
Von den fünf dargestellten Arbeitsregistern ist der *Akkumulator (ACC)* das zentrale Register, über das generell alle zu verarbeitenden Daten laufen. Bei Operationen der Arithmetik-Einheit *(ALU)* steht ein Operand daher immer im Akkumulator, wohin in der Regel auch das Ergebnis abgespeichert wird. Ein Bit zur Erweiterung des Akkumulators, das z. B. bei Ergebnis-Überläufen gebraucht wird, bezeichnet man mit *Link*.

Entsprechend den drei Bedeutungen, die ein Wort besitzen kann, existieren auch drei korrespondierende Register: Das *Adreßregister,* in dem die Adresse der gerade angesprochenen Speicherstelle steht, das *Datenregister* zur Zwischenspeicherung von Daten und das *Befehlsregister,* über das die Instruktionen zur Befehlsdecodierung gelangen.

Während die interne Verarbeitung eines Befehls abläuft, wird ein weiteres Register, das man als *Programmzähler* bezeichnet, um einen Schritt hochgezählt, bevor der neue Zählerstand zur Adressierung des nächstfolgenden Befehls ins Adreßregister übertragen wird. Zusätzlich sind weitere Hilfsregister vorhanden, die zur vorübergehenden Aufnahme von Daten dienen, wenn dies der zeitliche Ablauf erfordert; das tritt beispielsweise beim Austausch von Registerinhalten oder bei der Bildung von Zwischenergebnissen auf. Einige Prozessoren sind mit weiteren Arbeitsregistern ausgestattet, mit denen eine große Flexibilität bei der Programmierung erreicht wird. Eine solche Anordnung von Hilfsspeichern *(scratch pad memory)* erfordert eine eigene Verwaltung über eine eigene Speicheradressierung *(pointer)*, wenn die Anzahl der vorhandenen Hilfsregister zu groß wird; einige Standardmodelle besitzen 64 derartige Register! Zur Speicherung *(„Rettung")* aller aktuellen Registerinhalte bei externen Programmunterbrechungen dient ein Stapelregister *(stack)*; es gibt die zuletzt empfangenen Daten zuerst wieder aus (Stapelprinzip), was bei der Abarbeitung verschachtelter Unterbrechungen erforderlich ist.

3.2 Arbeitsweise

Zur abgekürzten Darstellung von Wortinhalten bedient man sich spezieller Zahlensysteme; damit bringt man die unübersichtliche Folge des binären Wortes in eine handliche, leicht überschaubare Form. Dazu sind besonders solche Zahlensysteme geeignet, deren Basis ein Vielfaches von 2 ist, z. B. das oktale (Basis 8) und das hexadezimale (Basis 16) System. Entsprechend bezeichnet man auch die Arbeitsweise eines Prozessors. Diese verkürzte Schreibweise hat wohlgemerkt nur für die Programmierung eine Bedeutung, wenn der Benutzer und das Übersetzungsprogramm diese Form verwenden; die Maschine selbst kennt nur die binäre Darstellungsweise im Maschinenwort.



Für ihren Tischrechner 9825 A hat die Firma Hewlett-Packard einen Mikroprozessor entwickelt, der mit keinem auf dem Markt befindlichen vergleichbar ist. Der rechte Chip allein stellt schon einen µP herkömmlicher Art dar; die beiden großen Chips links davon sind für mathematische Funktionen und die Ein-/Ausgabe-Steuerung zuständig. Für den Erwerb dieses Bausteins muß man allerdings 20 000 DM ausgeben, da er nur in Verbindung mit dem genannten Tischrechner erhältlich ist

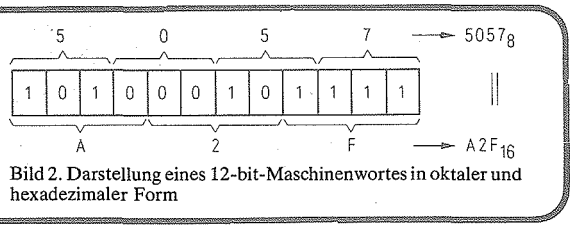


Bild 2. Darstellung eines 12-bit-Maschinenwortes in oktaler und hexadezimaler Form

Aus *Bild 2* geht die Umsetzung eines Maschinenwortes in die entsprechende oktale und hexadezimale Form hervor: Ohne auf einen Sinnzusammenhang der Digitalinformation zu achten, wird rein formal die Umwandlung vollzogen. So könnten etwa die ersten 5 Bits eine Adresse darstellen und insofern inhaltlich zusammengehören; dieser Zusammenhang ist nach der Umsetzung nicht mehr erkennbar. Für das Programmieren ist es eine unerläßliche Voraussetzung, sich mit dieser Darstellungsweise vertraut zu machen.

Arithmetische Operationen werden grundsätzlich im Zweier-Komplement *(two's complement)* durchgeführt. Die Überführung einer Binärzahl ins Zweier-Komplement geht so vor sich, daß man Bit für Bit invertiert und Eins hinzuzählt. Das höchstwertige Bit *(MSB)* repräsentiert das Vorzeichen einer Ganzzahl *(integer)*: Ist es gesetzt, bedeutet das ein negatives Vorzeichen [31, 32]. Rein formal kann also die Addition zweier positiver Zahlen zu einem negativen Ergebnis führen, wenn ein Übertrag in die höchste Stelle erfolgt. Das Nachvollziehen von Rechenvorgängen im Zweier-Komplement ist nur schwer möglich, weil es unserer gewohnten Denkweise nicht entspricht. Der Anwender muß solche Operationen nur in Ausnahmefällen durchführen.

4 Arbeitsphasen

4.1 Ablaufsteuerung

Durch den komplexen Systemaufbau ist eine sorgfältig abgestimmte zeitliche Ablaufsteuerung *(timing)* erforderlich. Dazu besitzen die Prozessoren einen quarzstabilisierten Taktgenerator, der einen sogenannten Mehrphasentakt erzeugt; das bedeutet, daß ein Arbeitstakt *(state)* an verschiedenen Stellen des Systems zu unterschiedlichen Zeiten eintrifft und wirkt, um beispielsweise Laufzeiten von Informationen zu berücksichtigen.

Man unterscheidet zwischen synchronem und asynchronem Betrieb; bei letzterem erfolgt die Synchronisierung extern, z. B. durch einen zentralen Taktgeber in einem größeren System. Im Sonderfall kann das so weit gehen, daß man den Takt fortnimmt und die interne Verarbeitung dadurch nur unterbricht, aber nicht stört; diese statische Betriebsweise ist eine wertvolle Eigenschaft eines Systems, da die Informationen auch bei Taktausfall erhalten bleiben und anschließend weiterverarbeitet werden können. Keinesfalls ist die Taktfrequenz ein absolutes Maß für die Arbeitsgeschwindigkeit eines Prozessors; es gibt Systeme, die bei langsamerem Takt einen Befehl schneller ausführen als solche mit höherer Taktfrequenz, weil sie bei der Befehlsausführung mit weniger Schritten auskommen.

4.2 Arbeitsgeschwindigkeit

Die allgemeine Angabe einer Zykluszeit *(cycle time)* in Datenblättern ist unzureichend; der Anwender muß zusätzlich exakt erfahren, für welchen Zyklus die Angabe gilt, weil nicht alle Befehle in derselben Zeit ausgeführt werden. Allgemein versteht man unter einem Befehlszyklus die Zeitspanne zum Holen, Decodieren und Ausführen eines Befehls; dabei kann man noch die Hol-Phase *(fetch cycle)* und Ausführungs-Phase *(execution cycle)* unterscheiden. Jede dieser Phasen setzt sich wiederum aus mehreren Arbeitstakten zusammen, die je nach System variieren.

Der Versuch, durch Angabe von typischen Additionszeiten ein Vergleichskriterium für die Arbeitsgeschwindigkeit von Mikroprozessoren zu schaffen, bleibt letztlich ohne Erfolg; zu unterschiedlich ist deren Arbeitsweise, als daß man einzelne Instruktionen miteinander vergleichen könnte. Daher lassen sich bestehende Normen auch nur bedingt und sehr allgemein auf die verschiedenen Hersteller-Angaben anwenden [33, 34].

Ein fundierter Systemvergleich ist nur mit geeigneten Testprogrammen (*benchmarks*; nicht zu verwechseln mit einer ähnlich klingenden Produktbezeichnung) möglich, die ganze Befehlsfolgen beinhalten. In diese Richtung zielt auch die Angabe des Datendurchsatzes *(throughput rate)* pro Zeiteinheit.

5 Wortorganisation

Es hängt ausschließlich von der Stellung im Arbeitsspeicher und von der Programmierung ab, ob ein Wort als Befehl oder als Operand aufgefaßt wird.

5.1 Befehlsstruktur

Jeder Befehl besteht aus einem Instruktionsteil *(operation code)* und einem Adreßteil *(address code)*; er vereint also die Angabe, was zu tun ist, mit dem Hinweis, womit etwas getan werden soll. Unabhängig davon besitzt der Befehl selbst eine Speicheradresse, unter der er zu erreichen ist; dies ist aber nicht mit der Adresse des Operanden zu verwechseln. Es sind drei Befehlsgruppen zu unterscheiden: speicherbezogene, registerbezogene und Ein-/Ausgabe-Befehle. Der Umfang des Befehlssatzes für ein bestimmtes System läßt einen Schluß auf dessen Leistungsfähigkeit zu.

5.2 Befehlsausführung

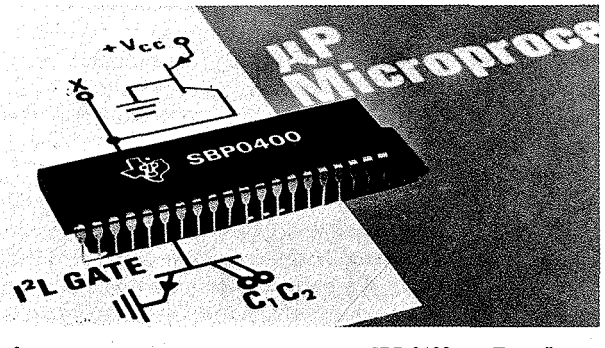
An einem vereinfachten Beispiel *(Bild 3)* sind schematisch die einzelnen Arbeitstakte zusammengestellt, die zur Ausführung eines einzigen Befehls notwendig sind. Es gibt umfangreiche Anweisungen, die über 20 interne Arbeitstakte erfordern.

6 Speicher

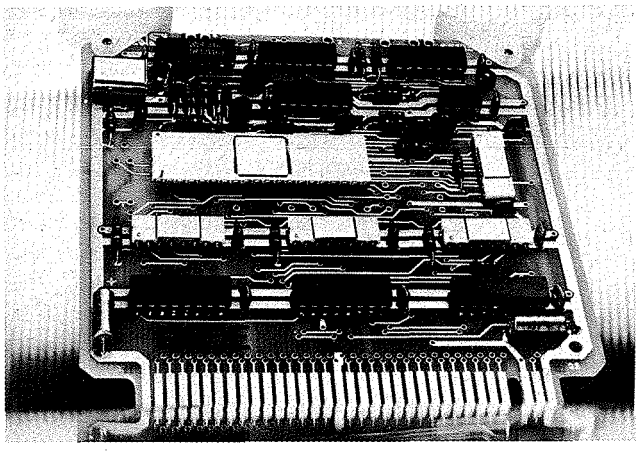
Erst im Zusammenspiel mit Speichern *(memory)* wird die Zentraleinheit eines Mikroprozessors zu einem arbeitsfähigen System. Es finden ausschließlich Halbleiterspeicher der unterschiedlichsten Technologien Verwendung.

6.1 Programmspeicher

Ziel einer Entwicklung ist es, ein arbeitsfähiges Programm für einen speziellen Anwendungsfall auszuarbeiten, um es



Die I²L-Technologie, in der der Mikroprozessor SBP 0400 von Texas Instruments aufgebaut ist, vereint die wesentlichen Vorteile anderer Technologien



Der Mikroprozessor PACE der Firma National Semiconductor, einer der ersten 16-bit-Typen, eignet sich durch die vermaschte Interrupt-Struktur besonders für die Prozeß-Steuerung

dann einem Lesespeicher *(ROM)* einzugeben. Dort bleibt es ständig erhalten und wird über den Programmzähler Schritt für Schritt aufgerufen. Die Programmierung der ROMs kann entweder mit einer Maske beim Hersteller oder mit einem speziellen Programmiergerät durch den Anwender selbst erfolgen. Die zweite Möglichkeit bietet sich bei kleineren Serien und in der Entwicklungsphase an, da dies rationeller ist. Der Programmiervorgang selbst ist irreversibel, lediglich die noch nicht programmierten Bits lassen sich noch nachträglich „einbrennen". Bei einigen speziellen Typen allerdings kann man den Inhalt durch UV-Bestrahlung löschen und anschließend neu eingeben. Besonders wirtschaftlich sind hierbei neu entwickelte ROMs in CMOS-Technologie, da ihre Verlustleistung um Größenordnungen niedriger liegt als bei den gängigen bipolaren Typen.

6.2 Arbeitsspeicher

Der Arbeitsspeicher ist wortorganisiert und als Schreib-/Lesespeicher *(RAM)* ausgeführt. Wichtigstes Kriterium hierbei ist die Zugriffszeit *(access time)* zu den Daten, da sie im wesentlichen die Arbeitsgeschwindigkeit eines Systems bestimmt. Es gibt statische RAMs, die ihre Information so lange halten, wie die Versorgungsspannung anliegt (interne Flipflops), und es gibt dynamische Typen, bei denen die Information in Form von Ladungsmengen gespeichert wird; hier ist für eine ständige „Auffrischung" *(refreshing)* der Daten zu sorgen (im Millisekundenbereich). Nach Ausfall des Taktes geht die Information verloren. Auch bei den RAMs bieten die statischen CMOS-Typen erhebliche Vorteile: Sie lassen sich durch kleine Akkumulatoren über einen langen Zeitraum puffern, d. h. sie behalten dann auch ohne die Versorgungsspannung des Systems ihre Information. In Zukunft sind RAMs in MNOS-Technologie zu erwarten, die eine Ladung auch ohne Refreshing extrem lange halten und sich damit wie nichtflüchtige *(non-volatile)* Speicher verhalten.

6.3 Befehlsdecodierung

Der Einsatz von *programmierbaren Logik-Einheiten (PLA)* bei der Befehlsdecodierung bringt gegenüber der Verwendung von ROMs erhebliche Vorteile: Während beim ROM jeder Eingangsadresse eine bestimmte Ausgangskonfiguration zugeordnet ist, kann man beim PLA die Kombination mehrerer Ausgangszustände (deren logische ODER-Verknüpfung) durch die entsprechende Adressierung erreichen [35]. Bei geeigneter Konzeption des Mikroprozessors lassen sich durch ein PLA sogar mehrere Befehle zusammenfassen und während eines einzigen Befehlszyklus' verarbeiten. Diese Möglichkeit, den Befehlsvorrat zu erweitern, geht in die Richtung der *Mikroprogrammierung,* bei der ganze Befehlsfolgen durch den Anwender festgelegt werden können.

7 Peripheriegeräte

Um mit dem Prozessor kommunizieren zu können, benötigt man ein extern anschließbares, peripheres Gerät. Im Normalfall ist das ein Datensichtgerät mit Tastatur, über das die Benutzer-Eingaben und Prozessor-Rückmeldungen ablaufen. Die Verwaltung dieser „primären" Peripherie erfolgt über fest in ROMs abgelegte Standard-Routinen, die die eigentliche Programmierarbeit des Anwenders nicht tangieren.

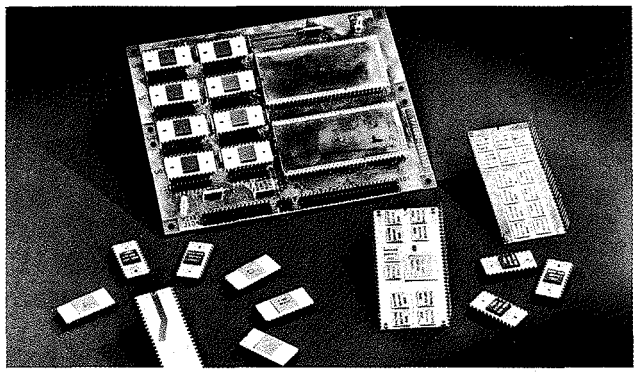
Eine wesentliche Hilfe, vor allem für den Entwicklungsingenieur ohne weitreichende Software-Erfahrung, ist eine Bedienkonsole *(control-panel).* Insbesondere während der Einarbeitung, aber auch bei späteren Tests, kann eine solche Einrichtung vorteilhaft sein; in der Regel ist sie über einen Vielfachstecker anschließbar.

Von Bedeutung sind ferner die Magnetbandkassette und die Floppy-disk-Einheit als Massenspeicher, sowie der Zeilendrucker und ein schneller Leser/Stanzer. Die beiden letztgenannten Geräte ermöglichen eine schnelle Datenausgabe, z. B. bei großen Datenmengen. Mit dem schnellen Leser kann man die Einlesezeit für einen Lochstreifen auf einige Sekunden reduzieren. Zum Vergleich: Ein Standard-Lochstreifenleser braucht für einen 4-K-Programmstreifen mehr als eine Viertelstunde. Alle schnellen Peripheriegeräte sind um ein Vielfaches teurer als die Standardausführungen. Zur Leitungsanpassung und Signalumsetzung gibt es spezielle Leitungstreiber/-empfänger-Bausteine *(UART),* die den seriellen Datenverkehr im asynchronen Betrieb ermöglichen.

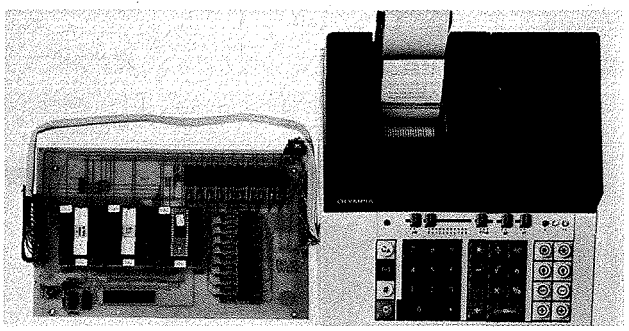
8 Programmierung

Die wichtigste Systemeigenschaft ist die Software-Ausstattung. Sie ist das einzige Medium zwischen Mensch und Maschine, und vom Komfort ihrer Organisation hängt die Bedienfreundlichkeit bei der Programmierung ab. Hier kann man zusätzliche Angebote an Leistungsfähigkeit gar nicht hoch genug einschätzen, weil mangelnde Ausstattung der mitgelieferten Betriebsprogramme ein spürbares Hindernis bei der Arbeit ist. Einige Systeme sind aufwärtskompatibel mit größeren Anlagen, d. h. sie verwenden dieselbe Symbolsprache wie z. B. ein Minicomputer. Bei bereits vorhandenen Rechnern sollte man auf solche Verträglichkeit achten, wenn man ein Mikroprozessor-System auswählt.

Die Systeme verwenden jeweils eigene Programmiersprachen, sogenannte *Assembler-Sprachen;* ihr logischer Aufbau



Eine Besonderheit stellen die Moduln der Firma Scientific Micro Systems dar: Auf einem Keramik-Substrat sind hier in Dickschicht-Technik Mikrocomputer-Teilsysteme zusammengefaßt



In der mittleren Daten- und Bürotechnik finden Mikroprozessoren ein weites Anwendungsfeld

und ihre Interpunktion (*syntax*) sind systemspezifisch. Das Erlernen solcher Formalismen ist eine Sache der Gewohnheit und bereitet keine prinzipiellen Schwierigkeiten.

8.1 Programmerstellung

Zur Realisierung eines Programms bestehen generell zwei Möglichkeiten: Verwendung des Mikroprozessor-Systems für das das Programm gedacht ist oder Einsatz einer Großrechenanlage. Wer zu einem Großcomputer keinen Zugang hat, kann sich über eine besondere Bedienkonsole *(terminal)* an eine solche Anlage anschließen, indem er die normale Post-Fernsprechleitung benutzt. Dieses Verfahren, bei dem mehrere verteilte Teilnehmer abwechselnd auf einen zentralen Rechner zugreifen, nennt man *Timesharing* [24]. Ein Großcomputer bietet komfortable Programmiermöglichkeiten und spart im Timesharing-Verfahren auch größere Investitionen. Jedoch ist für die Programmerstellung ein gut ausgebautes Mikroprozessor-System vorzuziehen, weil es den wirklichkeitsgetreuen Test unter praktischen Betriebsbedingungen ermöglicht. Bevor das Programm einwandfrei arbeitet, sind bis zu fünf Zwischenzustände zu durchlaufen, zu denen jeweils fertige Hilfsprogramme seitens der Hersteller angeboten werden.

Der Prozessor akzeptiert nur dann externe Daten, wenn er programmäßig dafür vorbereitet worden ist; ein Hilfsprogramm, das einerseits diesen Datenverkehr ermöglicht und andererseits verschiedene Modifizierungen eines Anwenderprogramms erlaubt, wird mit *Editor* bezeichnet. Wenn sich dieses Programm im Arbeitsspeicher des Prozessors befindet, genügt die Eingabe eines Steuerzeichens, z. B. „L", gefolgt von einem Startzeichen, z. B. „$", um (in diesem Fall) einen Lochstreifen einzulesen; zusätzlich sind gegebenenfalls noch Startadresse oder Adreßbereiche anzugeben. *Tabelle 3* enthält die Zusammenstellung einiger symbolischer Anweisungen, wie sie in ähnlicher Form bei einem angebotenen Editor

Tabelle 3. Symbolische Anweisungen an den Editor und ihre Bedeutung

B	*(breakpoint):* Markierung einer Stopp-Adresse im Programmablauf, um Zwischenzustände zu untersuchen
C	*(copy):* Duplizieren bestimmter Speicherbereiche
D	*(dump):* Ausstanzen bestimmt. Speicherbereiche auf Lochstreifen
E	*(execute):* Programmausführung von einer bestimmten Startadresse an
H	*(hexadecimal operation):* Addition oder Subtraktion von hexadezimalen Zahlen für die Adreßberechnung
L	*(load):* Einlesen eines Lochstreifens
M	*(modify):* Ändern eines Speicherinhaltes
O	*(octal operation):* Addition oder Subtraktion von oktalen Zahlen für die Adreßberechnung
T	*(type):* Ausdrucken bestimmter Speicherinhalte

Befehl: LOAD ACC, X	lade den Akkumulator mit dem Inhalt derjenigen Speicherstelle, die unter der Adresse X zu erreichen ist
Befehlsadresse: 1871	Adresse der Speicherstelle, die den Befehl enthält
Ausführung:	
1871 → AR	Befehlsadresse ins Adreßregister
(PC) = PC + 1	Programmzähler hochzählen
(1871) → DR	Inhalt der Speicherstelle 1871 (das ist der Befehl LOAD ACC, X) ins Datenregister
$(DR)_{INSTR}$ → IR	Instruktionsteil des Befehls aus dem Datenregister ins Befehlsregister und zur Dekodierung (Deutung des Befehls „Lade den Akkumulator")
$(DR)_{ADR}$ → AR	Adreßteil des Befehls (das ist X) aus dem Datenregister ins Adreßregister
(X) → DR	Inhalt der Speicherstelle X ins Datenregister
(DR) → ACC	(X) aus dem Datenregister in den Akkumulator

Vereinbarung: X bezeichnet die Speicherstelle selbst, (X) den Inhalt der Speicherstelle X

Bild 3. Schematische Darstellung einer vollständigen Befehlsausführung

existieren. Ein besonderes Hilfsprogramm *(routine)* zur Fehlererkennung nennt man *Debugging*. Wenn der Editor nicht fest in einem ROM vorhanden ist, muß das Laden des Lochstreifens, der das Editor-Programm enthält, auch ohne die Anweisung „L" möglich sein; denn solange der Editor nicht geladen ist, versteht die Maschine diese Anweisung nicht. Bei aller Nüchternheit im Umgang mit Rechnern gelangt man hier an einen Punkt, an dem man das System förmlich überlisten muß: Das gelingt entweder mit einer festverdrahteten Zusatzlogik, oder, weitaus eleganter, mit einigen Anweisungen am Anfang des Lochstreifens, die das Laden des Streifens bewirken, auf dem sie stehen. Die Amerikaner nennen das sehr treffend *Bootstrap-loader*. Diese Einrichtung ist im Lieferumfang enthalten. Solche Verwaltungsbefehle, die ohne Einfluß auf das eigentliche Programm bleiben, nennt man *Pseudo-Instruktionen*.

Mit einem weiteren Hilfsprogramm, dem *Assembler*, erfolgt dann schrittweise die Umsetzung des mnemotechnischen Programmcodes *(Quellenprogramm)* in die binäre Form der Maschinendarstellung. Wenn der Assembler auf dem Mikroprozessor-System selbst läuft, bezeichnet man ihn als *resident*; wird er dagegen auf anderen Anlagen realisiert, spricht man vom *Cross-Assembler*.

Das vom Editor ausgestanzte Quellenprogramm *(source tape)* wird in den Mikroprozessor, der mittlerweile den Assembler im Arbeitsspeicher hat, geladen. Daran schließen sich drei Programmläufe *(pass)* an, die zu unterschiedlichen Ergebnissen führen: Im ersten Lauf (pass one) werden die symbolischen Programmadressen *(label)* aufgelistet, und jeder einzelnen wird eine vom Assembler errechnete Speicherstelle zugewiesen; außerdem werden Syntax-Fehler erkannt und entsprechende Fehlermeldungen ausgedruckt. Sind noch Fehler enthalten, muß man erneut den Editor laden, das Programm korrigieren und einen berichtigten Streifen ausstanzen.

Erst nach einem fehlerfreien ersten Lauf kann im zweiten Lauf *(pass two)* die Ausgabe des Maschinen-Lochstreifens (*object tape*) erfolgen. Er enthält die Befehlsfolge des ursprünglich erstellten Quellenprogramms in einer Form, die die Maschine unmittelbar verarbeiten kann. Sind alle technischen Fehler aus dem Programm beseitigt, könnte man dieses Maschinenprogramm fest in ROMs programmieren und hätte damit das funktionsfähige Programm für den Mikroprozessor erstellt.

Ein dritter Lauf *(pass three)* schließlich dient zur vollständigen Dokumentation des Programms. Er führt zum Ausdruck des Quellenprogramms mit den mnemotechnischen Abkürzungen, zu denen parallel die entsprechenden Speicheradressen und Maschinenbefehle ausgegeben werden; die Befehlsdarstellung erfolgt in der systemeigenen Form, also oktal oder hexadezimal.

8.2 Adressierung

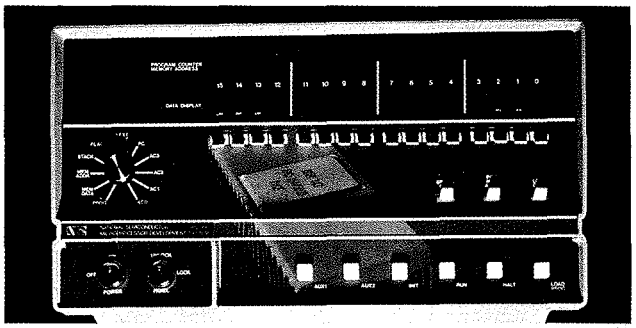
Wenn ein einziges Wort zur Speicheradressierung dienen sollte, wäre der ansprechbare Speicherumfang recht gering; ein 8-bit-Wort beispielsweise erlaubt nur die Adressierung von 256 Speicherstellen, während mit einer 16-bit-Wortlänge bereits 64 K angesprochen werden können. Deshalb sind besonders Prozessoren geringer Wortlänge so organisiert, daß Adressen aus zwei oder drei aufeinanderfolgenden Worten zusammengesetzt werden (Zwei- bzw. Drei-Wort-Befehle). Reicht der damit entstandene Rahmen noch nicht aus, erlauben einige Systeme die hardwaremäßige Erweiterung des Speichers *(memory extension)*. In diesem Fall wird eine zusätzliche Steuerlogik zur Adressierung erforderlich. Im einfachsten Fall weist eine Adresse unmittelbar auf die gewünschte Speicherstelle hin *(direkte Adressierung)*. Bei der *indirekten Adressierung* befindet sich in der angesprochenen Zelle nur die Adresse für diejenige Speicherstelle, auf die zugegriffen werden soll. Eine weitere Adressierungsart ist die *Adreßmodifikation*; hierbei wird die effektive Adresse durch verschiedene Operationen gewonnen, die vom Programm beeinflußbar sind. Eine Vielzahl von möglichen Adressierungsarten ist wünschenswert, da sie die Leistungsfähigkeit eines Systems erhöht.

8.3 Direkter Speicherzugriff

Normalerweise erfolgt das Einlesen externer Daten in den Speicher über die Zentraleinheit, das ist ein sehr langsamer Vorgang, weil für jedes Wort eine spezielle Routine ablaufen muß. Die Möglichkeit des direkten Speicherzugriffs *(DMA)* erlaubt den Datentransfer unmittelbar in den Arbeitsspeicher unter Umgehung des Mikroprozessors. Dieser wird während des Einlesens desaktiviert, und die Steuersignale für den Speicher kommen dann von der sendenden Stelle. Bei Systemen, die hohe Arbeitsgeschwindigkeiten haben sollen, ist diese Eigenschaft unerläßlich.

8.4 Externe Unterbrechungen

Bei einer Fülle von Anwendungen muß der Prozessor auf externe Meldungen reagieren, z. B. bei der Prozeßsteuerung. Dazu gibt es die Möglichkeit externer Programmunterbrechungen *(interrupt)*: Ein von außen angelegtes Signal hält das laufende Programm an und alle augenblicklichen Zustände der Zentraleinheit werden abgespeichert. Dann springt der Prozessor automatisch in ein vorbereitetes Unterprogramm *(interrupt routine)* und führt die gewünschte Operation aus, anschließend nimmt er mit den gespeicherten Informationen wieder die Verarbeitung des ursprünglichen Programms *(background program)* auf. Es werden auch Systeme mit verschachtelten Unterbrechungsmöglichkeiten *(multilevel interrupts)* angeboten; hier erhält eine Unterbrechung höherer Priorität den Vorrang vor einer anderen. Derartige vermaschte Unterbrechungen bezeichnet man mit *Nesting*; dabei müssen jeweils alle aktuellen Zustände der Zentraleinheit zwischengespeichert werden, damit auch eine unterbrochene Interrupt-Routine fortgesetzt werden kann. Können von verschiedenen externen Anschlüssen Unterbrechungssignale eintreffen, muß die unterbrechende Stelle lokalisiert werden. Das läßt sich entweder von einem Unterprogramm bewerkstelligen, oder das Unterbrechungssignal selbst führt eine Identifikation mit sich, die auf den Ursprung hindeutet *(vectored interrupt)*. Eine vergleichsweise langsame Art, auf externe Unterbrechungen einzugehen, stellt die ständige zyklische Abfrage aller möglichen Stellen dar, ob eine Unterbrechung gewünscht wird oder nicht *(polling)*.



Leistungsfähige Entwicklungssysteme schaffen die Voraussetzung für eine effektive Programmentwicklung

8.5 Zustandssignale

Die Zentraleinheit erzeugt in Abhängigkeit von bestimmten internen Zuständen entsprechende Signale *(flags)*, die dadurch entstehen, daß bei Auftreten des Zustandes ein zugehöriges Flipflop kippt. Diese Signale können vom Programm abgefragt und in logische Entscheidungen, z. B. Sprungbefehle, mit einbezogen werden.

9 Anwendungen

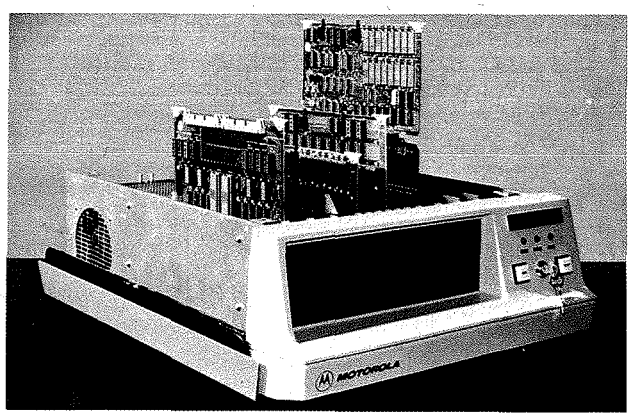
Der Einsatz von Mikroprozessoren ist nur dann sinnvoll, wenn die Arbeitsgeschwindigkeit ausreicht. In Verbindung mit einer Bus-Struktur bei Geräten oder Anlagen ist die Einbeziehung eines oder mehrerer Mikroprozessoren nahezu unerläßlich [38]. Bei umfangreicheren Aufgaben wird man ohnehin mehrere Prozessoren gleichzeitig einsetzen *(multiprocessing)*; der Materialpreis ist längst vernachlässigbar geworden gegenüber den Systemkosten, vor allem auf der Software-Seite.

9.1 Vor- und Nachteile

Der entscheidende Vorteil der Mikroprozessoren liegt in der hohen Flexibilität, verbunden mit einem vereinfachten Aufbau und erhöhter Zuverlässigkeit des Gerätes, in dem er eingesetzt wird. Daraus erwachsen aber auch wesentliche Nachteile: Die Gerätekonzeption wird schwerer durchschaubar, und Reparaturen oder Änderungen erfordern hochgradige Spezialisten. Es ist daher ein Irrglaube, anzunehmen, man könnte die Systemkosten durch Einsatz eines Mikroprozessors ohne weiteres senken; bevor man diese Materie so weit beherrscht wie etwa heute die Handhabung von TTL-Bausteinen, sind erhebliche Investitionen notwendig. Und die einfache Änderungsmöglichkeit des Geräteverhaltens durch Umprogrammieren, oft ohne Überlegung in den Raum gestellt, erfordert eben auch ihre Zeit und ist selbst vom Fachmann nicht mit der linken Hand abzutun. Es soll wohlgemerkt nicht gegen die Mikroprozessoren geredet werden, ganz im Gegenteil, aber es soll einmal deutlich gesagt sein, daß diese letztlich nur Hilfsmittel bleiben, die dem Anwender zunächst entsprechenden Einsatz abverlangen.

9.2 Steuerungsaufgaben

Bevorzugte Anwendungsgebiete sind Steuerungs- und Verwaltungsaufgaben. So bietet sich der Einsatz eines Mikro-



Das Entwicklungssystem der Firma Motorola ist für den Mikroprozessor MC 6800 vorgesehen und ermöglicht einen wirklichkeitsnahen Test mit umfangreicher Speicherbestückung

prozessors z. B. bei der Steuerung einer Elektronenstrahlröhre für ein Sichtgerät an oder dort, wo präzise Zeitabläufe zu steuern sind. Ebenso geeignet ist er für die Verwendung in Buchungsautomaten oder Registrierkassen, wo gleichzeitig zur Dateneingabe der aktuelle Stand bestimmter Zahlenkolonnen ermittelt und ausgegeben werden kann. Aber auch in komfortableren Meßgeräten hat der Mikroprozessor längst Einzug gefunden; dort übernimmt er beispielsweise die Steuerung der vom Bedienfeld eingegebenen Daten, führt die gewünschten Operationen aus und veranlaßt entsprechende Anzeigen oder Signalausgaben.

9.3 Rechnerbau

Es soll noch ein spezielles Anwendungsgebiet angesprochen werden, bei dem vorzugsweise die kaskadierbaren Prozessorbausteine zum Einsatz kommen: Der Bau von Rechenanlagen. Hierunter fallen Geräte von der Leistungsfähigkeit eines Minicomputers oder eines schnellen Prozeßrechners. Aber auch der gezielte Nachbau eines bestimmten Rechnertyps gehört hierzu. Dabei verhält sich das nachgebaute System genauso wie das kopierte, es ist also vollkommen gleich zu programmieren und zu bedienen. Geschieht diese Nachbildung nur programmäßig auf einem anderen Rechner, spricht man von *Emulation*. Das emulierende System muß in der Regel schneller sein als das nachgebildete, da zur Nachbildung eines Befehls u. U. mehrere neue Befehle erforderlich sind. Die Emulation gewinnt insofern an Bedeutung, als damit bestehende Programme unverändert auf einer neuen Anlage weiterbenutzt werden können. Das ist ein nicht zu unterschätzender Vorteil, weil damit die sehr aufwendigen Programmumstellungen entfallen.

10 Einstieg in die Praxis

Nach der Fülle der technischen Informationen stellt sich die Frage, wie man bei der persönlichen Einarbeitung erfolgversprechend vorgehen kann.

10.1 Seminare

In zunehmendem Maße bieten Hersteller und Vertriebsfirmen Schulungskurse zur Einarbeitung in die Mikroprozessor-Technik an. Diese sind dann erfolgversprechend, wenn bei ausreichend geringer Teilnehmerzahl eine individuelle Betreuung sichergestellt ist. Die Möglichkeit, selbst an den Prozessoren zu arbeiten, ist unabdingbare Voraussetzung dafür. Sogenannte Blitzkurse von eintägiger Dauer besitzen wenig Aussicht auf einen nachhaltigen Erfolg.

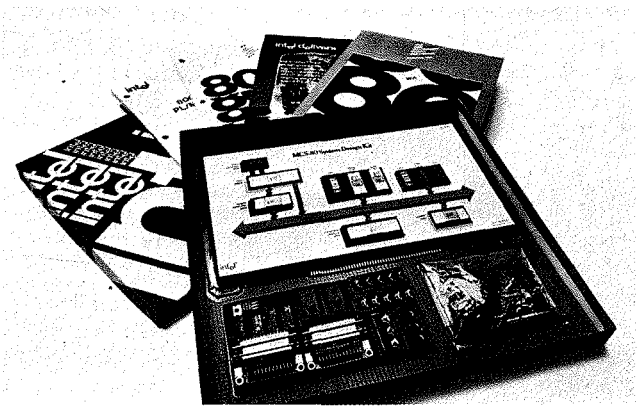
10.2 Eigene Erfahrung

Die im persönlichen Umgang mit dem Mikroprozessor gewonnene Erfahrung ist durch nichts zu ersetzen. Nur mit ihr kann man die Leistungsfähigkeit eines Systems vollständig beurteilen, was bei bloßem Literaturstudium nur unzureichend gelingt. Der Anfang ist leichter, als man gemeinhin glaubt: Heute sind die sogenannten Einkarten- und Hobby-Computer bereits in preisliche Größenordnungen vorgedrungen, die sie selbst für den Privatmann erschwinglich machen. Erste Arbeiten an derartigen Geräten führen auch dann zu einer fundierten Einarbeitung in die Materie, wenn es sich anfangs nur um spielerische Beschäftigung handelt.

10.3 Entwicklungssysteme und Mikrocomputer

Um dem Anwender die Handhabung eines Prozessors und dessen Programmierung zu erleichtern, werden funktionsfähige Mikrocomputer und vollständige Entwicklungssysteme angeboten. Letztere sind meist mit einem Typ des Mikroprozessors aufgebaut, für den die Programme erstellt werden sollen, und sie besitzen einen großen Speicher sowie Bedienelemente. Derartige Geräte sind besonders wertvoll für die Einarbeitung und Schulung. In dieser Ausbaustufe unterscheiden sie sich wesentlich von solchen Mikroprozessor-Systemen, die beispielsweise in ein Gerät eingebaut werden. Dort ist kein separates Bedienfeld für den Prozessor vorgesehen, und der Speicherumfang ist dem speziellen Anwendungsfall angepaßt.

Als primäre Auswahlkriterien kommen die Arbeitsgeschwindigkeit, die erforderlichen Versorgungsspannungen und der adressierbare Speicherbereich in Betracht. Hierbei ist zu klären, inwieweit sich der Mikroprozessor in eine vorhandene oder geplante Systemkonzeption einfügt. Von sehr weitreichender Bedeutung bei der Auswahl eines Modells ist die Programmiermöglichkeit, und zwar ausschließlich unter dem Gesichtspunkt der später zur Verfügung stehenden Hilfsmittel. Es ist deshalb besonders auf die Verwendbarkeit der angebotenen Programme auf bereits vorhandenen Rechenanlagen zu achten. Wenn diese Forderung nicht besteht, sollte man sich bei der Softwareauswahl im Zweifelsfall immer für das leistungsfähigere Angebot entscheiden, weil eine spätere Umstellung wegen der unverhältnismäßig hohen Kosten nur in den seltensten Fällen realisiert wird. Eine pauschale Zuordnung einzelner Wortlängen zu bestimmten Anwendungsfällen kann nicht getroffen werden. Ein Anhalt dazu ist in [26] zu finden. Im Einzelfall, besonders bei kleineren Stückzahlen, wird jedoch die Erfahrung mit einem bestimmten Modell eher ausschlaggebend für dessen Einsatz sein als die optimale Hardware-Anpassung.



Den Aufbau eines vollständigen Mikrocomputer-Systems innerhalb weniger Stunden ermöglicht ein Bausatz der Firma Intel. Er enthält alle erforderlichen Bauelemente, den Mikroprozessor MCS 8080 und Software-Handbücher

10.4 Fremdunterstützung

Die Firmen sind generell daran interessiert, ihre Kunden weitgehend bei der Arbeit mit dem Mikroprozessor zu unterstützen. Spezialisten beantworten auch Detailfragen rasch per Telefon oder beraten an Ort und Stelle. Dieses Entgegenkommen ist überall festgestellt worden, und man sollte bei Bedarf davon Gebrauch machen!

11 Zukunftsaussichten

In erster Linie sind fallende Preise dafür ausschlaggebend, daß man im Laufe der Zeit eine ständig wachsende Leistungsfähigkeit angeboten bekommt und sich diese auch zunutze macht. Das Leistungsspektrum wird sich derart verschieben, daß am unteren Ende die herkömmlichen 8-bit-Prozessoren den Platz einnehmen, den heute noch die billigen 4-bit-Typen innehaben. Als Standardwortlänge wird sich in absehbarer Zeit das 16-bit-Format durchsetzen, das in Verbindung mit sinkender spezifischer Verlustleistung zu einer Erhöhung der Verarbeitungsleistung um den Faktor 10...100 führt.

Gleichzeitig wird die Linie der Aufwärtskompatibilität aufgegeben oder zumindest teilweise verlassen; die 16-bit-Typen sind also nicht unbedingt mehr kompatibel mit einem 8-bit-Vorgängermodell, so wie sich dies besonders anschaulich in der Mikroprozessor-Reihe vom 4004 bis hin zum Z80 vollzogen hat.

In Zukunft ermöglichen preiswerte Halbleiterspeicher selbst den Einsatz umfangreichster Entwicklungsunterstützung in Form von fertigen ROMs, die einem Mikroprozessor bzw. Ein-/Chip-Mikrocomputer die Flexibilität eines heutigen Entwicklungssystems verleihen. Eine Abschätzung der Entwicklungstendenzen über die achtziger Jahre hinaus ist aus heutiger Sicht nicht möglich. Einsatzmöglichkeiten und Absatzgebiete für (nach heutigen Maßstäben gemessen) riesige Speicher zeichnen sich ab bei der synthetischen Spracherzeugung oder bei kompletten Sprach-Übersetzungs-Maschinen, um nur zwei der vielfältigen Anwendungsbeispiele zu nennen.

Auf jeden Fall ist es falsch, zu argumentieren, daß irgendwann einmal kein Markt mehr für die ständig steigende Leistungsfähigkeit von Mikrocomputer-Produkten vorhanden sein wird. Man braucht nur ein Jahrzehnt zurückzudenken, um sich an Rechenschieber und Logarithmentafel als unerläßliche Hilfsmittel für den Ingenieur zu erinnern. Hätte man damals zu prophezeien gewagt, daß es zehn Jahre später leistungsfähige Digitalrechner im Taschenformat geben würde, die sogar zum Gepäck von Schulkindern gehören, wäre man ohne viel Aufhebens als Phantast abgetan worden.

Unter diesem Aspekt sind die vage angedeuteten Trends sicherlich nicht zu hoch gegriffen.

Dipl.-Ing. Reinhard Gößler wurde in Thüringen geboren und wuchs in Berlin auf. Dem Studium an der TU Hannover folgte eine dreijährige Tätigkeit als Entwicklungsingenieur in der Industrie. Heute ist er tätig als fester Mitarbeiter im FRANZIS-Verlag und in der Mikrocomputer-Schulung. Außerdem führt er industrielle Mikrocomputer-Entwicklungsaufträge durch.
Hobby: Schnelle Stuttgarter Sportwagen
Telefon: (0421) 250347
ELEKTRONIK-Leser seit 1969

Literatur

Firmenneutrales Schrifttum, das sich gleichermaßen zur Einarbeitung in die Materie, zur Vertiefung des Stoffes und als Nachschlagewerk eignet:

[1] Microprocessors and Microcomputers. A Comprehensive Technical Introduction (Course Notes). Integrated Computer Systems, Culver City, Kalifornien (1975).
[2] *Blomeyer-Bartenstein, H. P.:* Mikroprozessoren und Mikrocomputer. Eine Einführung in die Grundlagen und Anwendungstechnik. Fachbuch der Siemens AG, München (1975).
[3] Mikrocomputer. Eine Zusammenfassung aktueller Aufsätze aus der Fachzeitschrift „Der Elektroniker". AT-Fachverlag GmbH, Stuttgart (1975).
[4] *Tireford, H.:* Vom Computer zum Mikroprozessor. Fachschrift der Firma Motorola, Taunusstein (1975).
[5] *Weissberger, A. J.:* MOS/LSI microprocessor selection. Electronic Design 1974, Heft 12, S. 100...104.
[6] *Lilen, H.:* Introduction à la micro-informatique: Du microprocesseur au micro-ordinateur. Société des Editions Radio. Frankreich (1975).
[7] Aufsatzsammlung über alle Themen, die im Zusammenhang mit dem Mikroprozessor stehen; auch bei mangelnden spanischen Sprachkenntnissen ein hilfreiches Nachschlagewerk: Mundo Electrónico 1975, Nr. 45 (Novemberheft).

Ein Standardwerk zur Einarbeitung in die PDP-8-Software, gleichzeitig das Benutzerhandbuch für die Programmierung:

[8] Introduction to Programming. Handbuch der Digital Equipment Corporation, USA. Vertrieb: Spezial-Electronic KG, München und Bückeburg (1970).

Komplette Literaturzusammenstellungen im Nachdruck:

[9] Microprocessors and Microcomputers. Reprints. Integrated Computer Systems, Culver City, Kalifornien (1975).
[10] Microprocessors and Microcomputers. Manufacturers' Literature. Integrated Computer Systems, Culver City, Kalifornien (1975).

Aufsätze, die sich aus allgemeiner Sicht mit der Materie befassen:

[11] *Lewandowski, R.:* Preparation: the key to success with microprocessors. Electronics 1975, 20. März, S. 101...106.
[12] *Bailey, S. J.:* Microprocessor: Candidate For Distributed Computing Control. Control Engineering 1974, März, S. 41...44.
[13] *Martinez, R.:* A Look at Trends in Microprocessor/Microcomputer Software Systems. Computer Design 1975, Nr. 6, S. 51...57.

Vergleichende Gegenüberstellungen und nähere Betrachtung einiger Modelle:

[14] *Groves, B.:* Microprocessors. Instruments & Control Systems 1975, März, S. 47...53.
[15] Automatique et informatique industrielles 1975, Oktoberheft.

Firmenschriften, die ein näheres Kennenlernen der vorgestellten Modelle erleichtern, ohne jeweils den vollständigen Unterlagensatz erwerben und durcharbeiten zu müssen:

[16] Intersil IM 6100-CMOS-12-bit-Microprocessor. Kostenlose Druckschrift der Firma Spezial-Electronic, München und Bückeburg.
[17] Intercept Prototyping System. Kostenlose Druckschrift der Firma Spezial-Electronic, München und Bückeburg.
[18] M 6800 Application Manual. Motorola, Taunusstein (1975).
[19] Exorciser data sheets. Kostenlose Druckschrift der Firma Motorola, Taunusstein.
[20] The F 8 from Mostek. *Braunes* und *grünes* Heft. Kostenlose Druckschriften der Firma Mostek, Bernhausen/Stuttgart.
[21] COSMAC Microkit. Operator's Manual. RCA (1975). Vertrieb: A. Neye Enatechnik, Quickborn/Hamburg.
[22] *Bass, J. E.:* Microcomputer-Based Communications Concentrator Systems. Kostenlose Druckschrift der Firma Rockwell (MCC 505). Vertrieb: COSMOS Electronic, München.
[23] The SMS Micro-Controller. System Description. Scientific Micro Systems (1974). Vertrieb: Munzig International, München.
[24] The SMS Micro-Controller. Timesharing User Guide. Scientific Micro Systems (1974). Vertrieb: Munzig International, München.

Schnell verfügbare Aufsätze, die sich ausgezeichnet für eine erste Einarbeitung eignen:

[25] *Neunert, H.:* Systemtechnik mit Mikrocomputern. ELEKTRONIK 1974, H. 10, S. 391...395.
[26] *Habgood, D.:* Wie wär's mit einem Mikroprozessor? ELEKTRONIK 1974, H. 10, S. 379...382.
[27] *Fertl, W.:* MPS 10 – Ein modularer Mikrocomputer. ELEKTRONIK 1974, H. 10, S. 383...386.
[28] *Münchrath, R.:* Tendenzen der Minicomputer-Entwicklung. ELEKTRONIK 1973, H. 10, S. 345...348.
[29] *Scheytt, W.:* Minicomputer. ELEKTRONIK 1975, H. 10, S. 82...90.
[30] *Dieckmann, H. W.:* Mikroprozessor. Computer statt festverdrahteter Logik. FUNKSCHAU 1975, H. 26, S. 53...56.

Weitere Quellenhinweise:

[31] Code-Übersicht. Arbeitsblatt Nr. 69. ELEKTRONIK 1972, H. 9, S. 327...328.
[32] Codes für Analog/Digital- und Digital/Analog-Umsetzer. Arbeitsblatt Nr. 93. ELEKTRONIK 1975, H. 12, S. 87...90.
[33] Informationsverarbeitung. Begriffe. DIN 44 300. Beuth-Vertrieb, Berlin und Köln (1972).
[34] Prozeßrechensysteme. Begriffe. DIN 66 201. Beuth-Vertrieb, Berlin und Köln (1971).
[35] *Wuttke, G.:* Keine Angst vor PLAs. Elektronik-Journal 1975, H. 11, S. 6.
[36] *Schmid, H.:* Monolithic Processors. Computer Design 1974. H. 10, S. 87...95.
[37] *Horton, Englade, McGee:* I^2L takes bipolar integration a significant step forward. Electronics 1975. Erstes Februarheft. S. 83...90.
[38] *Pannach, A.:* Interface-Entwicklung für den IEC-Bus. ELEKTRONIK 1975, H. 12, S. 61...64.
[39] *Huse, H.:* I^2L: Die Technologie und ihre Anwendung in einem schnellen 4-bit-Mikroprozessorelement. ELEKTRONIK 1976, H. 2, S. 79...82.

Obwohl der Mikroprozessor, hervorgegangen aus dem Wunsch nach einer universellen, komplexen Standard-Schaltung, in der Bauelementeentwicklung vorläufig einen gewissen Endpunkt setzte, geht die stürmische Weiterentwicklung der Halbleitertechnologie ungebrochen weiter. Und zwar wird sich – entsprechend dem Moore'schen Gesetz – bis in die 80er Jahre die Anzahl der Funktionen auf einem Chip pro Jahr verdoppeln. Das bedeutet, daß die Mikroprozessoren noch komplexer werden (z. B. höhere Speicherkapazität, zusätzliche Peripherieschaltungen). Schließlich dürfte eine neue Generation von 8- und 16-bit-Mikroprozessoren auf den Markt kommen, die hinsichtlich Architektur und Befehlssatz besser den höheren Programmiersprachen angepaßt sind.

Dr. Hans Hoffmann

Entwicklungstendenzen bei Mikroprozessoren

Technologie und Anwendung

1 Programmierbare Systeme und Steuereinheiten

Programmierbare Systeme, die universell einsetzbar sind, sind aus der Datenverarbeitung allgemein bekannt. Im *Bild 1* ist das Beispiel einer Datenverarbeitungsanlage mit ihren verschiedenen Funktionseinheiten schematisch dargestellt. Außer der zentralen Recheneinheit und dem Hauptspeicher für das Speichern von Programm und Daten sind an das System auch die verschiedenen peripheren Einheiten angeschlossen. Im allgemeinen benötigt man bei umfangreichen Datenverarbeitungsanlagen und anderen Systemen je nach Komplexität der Peripherie-Geräte unterschiedliche Steuereinheiten. Moderne Anlagen zeichnen sich dadurch aus, daß die Steuereinheiten der elektromechanischen Peripherie, wie z. B. Schnelldrucker oder Plattenspeicher, mittels programmierbarer Prozessoren aufgebaut werden.

Die Verwendung programmierbarer Prozessoren als Steuereinheiten ergibt außer einer Standardisierung der Hardware eine bessere Anpassungsmöglichkeit an das Verbindungssystem der Funktionseinheiten, im allgemeinen eine *Bus-Struktur*. Umfangreichere Steuereinheiten können bei steigendem Integrationsgrad kostengünstiger hergestellt werden oder bei gleichen Kosten mehr Funktionen übernehmen. Dies führt zu einer immer stärkeren Entlastung der Zentraleinheit. Hierbei darf nicht übersehen werden, daß die Anforderungen an einen Steuerprozessor für sehr schnelle Plattenspeicher hinsichtlich Komplexität und Geschwindigkeit in der gleichen Größenordnung liegen wie für die Zentraleinheit, während es andererseits aus System-Gesichtspunkten manchmal nicht lohnt, überhaupt Steuerprozessoren einzusetzen, da die zentrale Recheneinheit z. B. während eines Programm-Lade-Vorganges die Ansteuerung selbst vornehmen kann. In anderen Bereichen der Ein/Ausgabe-Geräte kann es wiederum kostengünstiger sein, in einem Gerät mit unterschiedlichen Funktionen – z. B. in einem intelligenten Terminal – mehrere Prozessoren einzusetzen, um mit ihnen einzelne, abgrenzbare Funktionen zu realisieren.

Die in Bild 1 dargestellten Strukturen, wie sie von den Datenverarbeitungsanlagen vorgegeben worden sind, kann man jetzt, wo Prozessoren als Einzel-Bauelemente in Form von Mikroprozessoren erhältlich sind, auf andere Steuerfunktionen außerhalb der Datenverarbeitung übertragen und damit kostengünstigere Systeme aufbauen. Bei etwas komplexeren Anlagen kann es sich als vorteilhaft erweisen, wenn ein Mikroprozessor (als zentrale Recheneinheit) die Steuerung anderer Mikroprozessoren als Steuerprozessoren übernimmt, oder – bei einfacheren Systemen – ein Mikroprozessor auch die Steuerung der speziellen Peripherie-Bausteine durchführt.

In *Bild 2* ist dargestellt, wie man die Gedanken und Strukturen, die in der Datenverarbeitung üblich sind, durch das Bauelement Mikroprozessor auch auf Geräte und Systeme der Konsum-Elektronik im weitesten Sinne übertragen kann. Das bisherige Fernsehgerät wird durch verschiedene zusätzliche Geräte erwei-

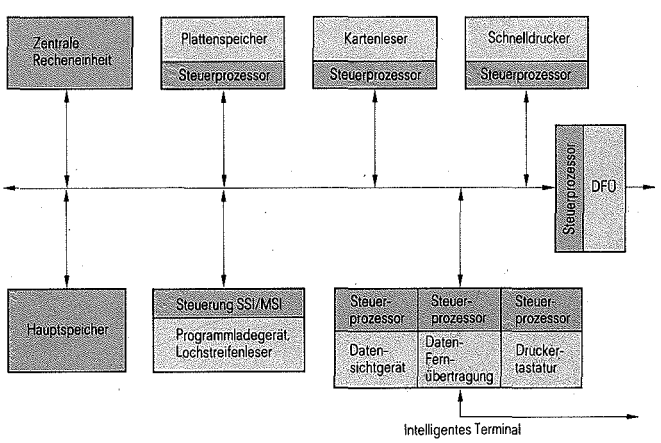


Bild 1. Beispiel eines Datenverarbeitungssystems

tert. Ein Zentralrechner mit entsprechendem Speicher für Daten und Programm steuert über eine Bus-Struktur die jeweiligen Geräte wie z. B. Digital-Kassette, Spiele, digitale Uhr, Bildschirmtext (Viewdata) und Videotext (Teletext). Die komplexeren Geräte werden wiederum durch kleine Steuer-Prozessoren gesteuert, wobei diese Prozessoren die Anpassung der gerätespezifischen Elektronik und Mechanik an die Bus-Struktur des Bildgerätes vornehmen. Das Fernsehgerät, so wie man es heute kennt, wird dann in diesem umfangreichen System eigentlich nur noch für die bildliche Darstellung der vielfältigen, von außen angebotenen Informationen benutzt.

Eine Abschwächung der technologischen Fortschritte in der Halbleiter-Industrie ist z. Zt. nicht abzusehen, so daß sowohl Fertigungstechnik wie auch Schaltungstechnik und Prozeßtechnologie weiterhin dafür sorgen werden, daß sich die äquivalenten Funktionen, die pro Chip realisiert werden können, bis auf weiteres alle zwei Jahre verdoppeln werden und daß die Kosten pro Funktion stark sinken.

2 Mikroprozessoren – Klassen und Anwendungen

Eine scharfe Abgrenzung des Anwendungsbereiches von Mikroprozessoren ist nicht möglich, weil die Einsatzgebiete fortlaufend zunehmen. Sie reichen von Prozessoren in der Datenverarbeitung als zentrale Recheneinheit und in Steuereinheiten bis hinunter zum Tankstellen-Steuer- und Rechengerät, vom Spezialrechner für Datenfernübertragungen bis zum Taxameter oder vom Code-Umsetzer eines Fernbedienungsgerätes für Fernseh-Empfänger bis zum Einsatz als Schach-Partner.

Für die unterschiedlichen Anforderungen dieses weiten Anwendungsbereiches unterscheidet man im wesentlichen vier Klassen von Mikroprozessoren:

- Kaskadierbare Prozessor-Elemente
- Schnelle Steuerprozessoren
- 4-/8-/16-bit-Prozessoren
- 4-/8-/16-bit-Computer

Hierbei unterscheiden sich die einzelnen Gruppen hinsichtlich der eingesetzten Technologie, ihrer Flexibilität und des Integrationsgrades. Die wichtigsten Merkmale der verschiedenen Typen sind in der *Tabelle* zusammengefaßt.

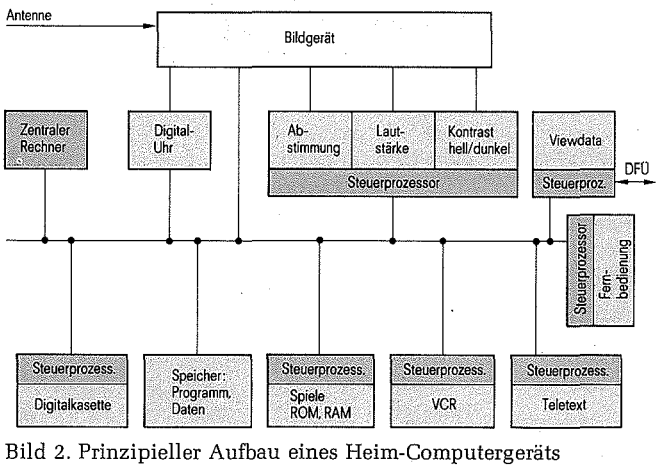


Bild 2. Prinzipieller Aufbau eines Heim-Computergeräts

In der Spalte a) der Tabelle sind einige Eigenschaften der aus kaskadierbaren Elementen aufgebauten Prozessoren aufgeführt. Diese Mikroprozessoren bieten durch die Möglichkeit, die Elemente zu kaskadieren, d. h. aufeinanderzustapeln und damit die Wortbreite der Rechner den gewünschten Anforderungen anzupassen, und durch die Möglichkeit, diese durch ein Mikroprogramm anzusteuern, große Freiheiten in der System-Architektur. Beides zusammen erlaubt einen kostengünstigen und sehr flexiblen Aufbau von Rechensystemen bestehender Rechner (Vorteil: vorhandene Software kann weiterverwendet werden) und den Aufbau von speziellen Steuereinheiten für hohe Anforderungen, z. B. für schnelle Plattenspeicher oder ganz spezifische Steuer-Prozessoren.

Da mit Hilfe der Mikroprogrammierung der einzelne Befehl aus dem Anwender-Programm in einzelne Schritte aufgelöst wird, ist der Befehlsvorrat eines so aufgebauten Rechners erst durch das Mikroprogramm festgelegt. Hauptanwendungsgebiet dieser Mikroprozessoren ist der Aufbau von Mini-Rechnern. Die 2900-Familie gilt als repräsentativ für diese Bauelemente.

Im Gegensatz dazu sind schnelle Steuer-Prozessoren (s. Tabelle, Spalte b) mit einem festen Befehlsvorrat versehen. Sie sind speziell konzipiert für einen Geschwindigkeitsbereich, der zwischen MOS-Prozessoren und den eben beschriebenen kaskadierbaren Prozessor-Elementen liegt. Der Befehlsvorrat ist angepaßt an Steuerfunktionen, d. h. nicht arithmetisch orientiert. Mit 40 mm^2 Kristall-Fläche und Zweilagen-Verdrahtung auf dem Kristall stellt einer dieser kommerziell verfügbaren Prozessoren (8X300) die obere Grenze des heute technologisch Erreichbaren dar.

Die Spalte c) in der Tabelle enthält die Eigenschaften des 16-bit-Mikroprozessors aufgezeigt. Wesentliches Merkmal – und damit auch wesentlicher Anwendungsbereich – ist die Verarbeitung von 16-bit-Daten, die bei Prozeßrechner-Anwendungen bedeutsam werden. Außerdem werden solche Prozessoren noch bei der Datenverarbeitung in Büromaschinen und Datenfernverarbeitung eingesetzt. Beispiele für diese Mikroprozessoren sind 9900 und PACE.

Einige wichtige Eigenschaften der 8-bit-Mikroprozessoren sind in Spalte d) angegeben. Diese Klasse stellt eigentlich die Art von Mikroprozessoren dar, die heute als Mikroprozessor schlechthin bezeichnet wird. Seine Anwendungen reichen von einfachen industriellen Steuerungen über Peripherie-Steuerungen in der Datenverarbeitung bis zum Einsatz im Fernsehgerät als Spieleprozessor. In diesem Bereich wird eine sehr breite Palette angeboten, nur um einige zu nennen: Z 80, 8080, 2650, 6800.

Der Einsatz dieser Mikroprozessoren wird im wesentlichen noch unterstützt durch die sog. „Entwicklungssysteme". Dies sind kleine Computer-Systeme, aus Mikroprozessoren aufgebaut, und in der oberen Ausbaustufe mit einem Plattenspeicher (Floppy Disk), Datensichtgerät und Drucker ausgerüstet. Entsprechende Software steht zur Verfügung, um sowohl die Software für das Anwenderprogramm zu entwickeln als auch die Fehlersuche zu erleichtern. Weiterhin

ermöglichen diese Systeme den eigentlichen Integrationstest, d. h. das Zusammenspiel der Software mit der neu erstellten Hardware zu vereinfachen.

Die Spalte e) der Tabelle enthält die Eigenschaften der unteren Klasse der Mikroprozessoren. Die Wortbreite beträgt 4 oder auch 8 bit. Als Technologie wird teilweise noch die langsamere P-MOS-Technologie benutzt. Die 4-bit-Versionen werden ebenso wie die P-Kanal-MOS-Technik nicht mehr wesentlich für Neuentwicklungen benutzt. Diese einfachen Prozessoren werden für Steuerungen im Konsumer- und Industrie-Bereich eingesetzt, im wesentlichen als Ersatz von Elektromechanik oder von kleinen SSI/MSI-Steuerungen. Beispiele für diese Prozessoren sind 4040, SC/MP. Der Trend jedoch geht dahin, daß man diese kleinen Prozessoren inklusive Hauptspeicher mit Festwertspeicher für Programme und Schreib/Lese-Speicher für Daten auf einem Chip unterbringt (Einchip-Systeme/Mikrocomputer).

In Spalte f) sind schließlich einige Eigenschaften von Mikrocomputern angegeben. Sie ergeben für den Geräte-Hersteller nicht nur eine Verbesserung des Preis-/Leistungsverhältnisses, sondern durch die geringere Anzahl von Gehäusen, die verarbeitet werden müssen, absolut gesehen eine Verringerung der Kosten, so daß man erwarten kann, daß diese Einchip-Mikrocomputer wiederum neue Applikationsfelder eröffnen werden, und nicht nur einen Ersatz von TTL-Logik darstellen. Beispiele für diese Klasse sind TMS 1000, 8048 und 9940.

Besonders hervorzuheben sind die vorhandenen bzw. angekündigten Einchip-Mikrocomputer mit „EPROM (*Electrically Programmable Read Only Memory*)", einem löschbaren und wiedereinschreibbaren Festwertspeicher auf dem Chip (8748) [5]. Mit diesen Bauelementen kann eine sehr effektive Entwicklung durchgeführt werden, wodurch eine Null-Serie, die im allgemeinen erst die restlichen Fehler des Systems erkennen läßt, etwa 8...12 Wochen früher anlaufen kann. Sind jedoch nur kleine Serien geplant, wie sie z. B. in der Meßtechnik oder allgemein in der industriellen Steuerung vorkommen können, so sind diese Bauelemente aufgrund der Flexibilität in der Programmierung hierfür besonders geeignet. Der Einsatz in kleinen Serien wird erfolgen, sobald die Preise dieser Produkte stark fallen, was sehr wahrscheinlich zu erwarten ist.

Von der Schaltungstechnik her gesehen werden neue Einchip-Computer in N-MOS-Technologie ausgeführt. Die Schnittstelle zur Außenwelt ist TTL-kompatibel. Durch die neuen Möglichkeiten der Ionenimplantation verwendet man sowohl „enhancement-" als auch „depletion-load" MOS-Transistoren auf einem Chip, wobei dynamische und statische Schaltungstechnik zum Zuge kommt mit nur einer Versorgungsspannung von 5 V.

Um an einem Beispiel aufzuzeigen, welche Wege die Halbleiter-Industrie geht, um die benötigte Chipfläche zu verkleinern, sei auf *Bild 3* hingewiesen. Die lokale Oxidation ist in letzter Zeit oft in der Literatur erwähnt worden. Sie führt nicht nur bei bipolaren Transistoren sondern auch bei MOS-Transistoren zu einer Verkleinerung der Fläche und der Verdrahtungskapazitäten [6, 7].

3 Peripherie-Schaltungen

Bei den Einsatzgebieten von Mikroprozessoren, d. h. von programmierbaren Bauelementen, muß man sich immer vor Augen halten, daß die Verarbeitung sequentiell vorgenommen wird, da die einzelnen Befehle des Programms sequentiell abgearbeitet werden.

Tabelle der Eigenschaften verschiedener Mikroprozessoren

a) Prozessoren aus kaskadierbaren Prozessorelementen
- kaskadierbare Prozessorelemente (RALU), 4 bit breit
- sehr schnell < 100 ns
- mikroprogrammierbar (Mikroassembler)
- werden durch komplexe Mikroprogramm-Steuereinheiten ergänzt
- benötigen Ein/Ausgabe-Schaltungen
- bipolare Technologie

b) Steuerprozessoren (Sequenzer)
- Byte-orientiert
- Bit-orientierter Befehlsvorrat
- optimiert für Bit-
  - Auswahl
  - Testen
  - Setzen
  - Schieben
  - Verknüpfungen
- hohe Geschwindigkeiten
- bipolare Technologie
- Assembler

c) Mikroprozessoren der „oberen" Klasse
- 16 bit Wortbreite
- Befehlsvorrat von Minirechnern
- schnell
- gute DMA-Möglichkeiten
- großer Adreßraum
- gute Interrupt-Struktur
- N-MOS-Technologie

d) Mikroprozessoren der „mittleren" Klasse
- 8 bit Wortbreite
- relativ vielseitiger Befehlsvorrat
- relativ vielseitiger arithmetischer Befehlsvorrat
- mittelschnell
- DMA-Möglichkeiten
- kleiner Adreßraum
- Interrupt-Struktur
- N-MOS-Technologie
- Assembler

e) Mikroprozessoren der „unteren" Klasse
- 4 oder 8 bit Wortbreite
- einfacher Befehlsvorrat
- begrenzter arithmetischer Befehlsvorrat
- nicht besonders schnell
- nicht speicherintensiv (kein DMA)
- kleiner Adreßraum
- einfache Interrupt-Möglichkeiten
- N-MOS-(P-MOS)Technologie

f) Mikrocomputer
- 4, 8 oder 16 bit Wortbreite
- einfacherer Befehlsvorrat
- begrenzter arithmetischer Befehlsvorrat
- nicht besonders schnell
- 1...2 KByte Speicher auf dem Chip
- maskenprogrammiert
- einfache Interrupt-Möglichkeiten
- N-MOS-(P-MOS)Technologie
- EPROM-Version/Emulator
- Assembler

Hierin liegt einerseits die große Flexibilität in der Anwendung von Mikroprozessoren, andererseits aber die Beschränkung der Verarbeitungsgeschwindigkeit, die kleiner ist als bei vergleichbaren anwendungsspezifisch aufgebauten Lösungen [8]. In *Bild 4* ist dieser Zusammenhang gezeigt.

Im Bild sind noch weitere Bauelemente eingezeichnet, die zur Lösung spezifischer Aufgaben für größere Stückzahlen zur Entlastung und Ergänzung von Mikroprozessor-orientierten Systemen entwickelt worden sind. Diese für die jeweiligen Mikroprozessor-Familien konzipierten Schaltungen, vielfach auch „Peripherie-Schaltungen" genannt, dienen der Ergänzung und Erweiterung des Anwendungsbereiches dieser Mikroprozessor-orientierten Systeme. Einige dieser Bauelemente sind zur Erreichung einer höheren Geschwindigkeit festverdrahtet, andere sind wiederum programmierbar aufgebaut [9]. Bei der Einführung der Mikroprozessoren wurden zunächst relativ einfache Zusatzschaltungen benötigt, z. B. Takt-Bausteine, um die anfangs vorhandenen Unzulänglichkeiten der Prozessoren auszugleichen, während neuere Peripherie-Schaltungen durch autonomes Abarbeiten sehr komplizierter Funktionen die Mikroprozessor-Systeme ergänzen. Diese Peripherie-Schaltungen übertreffen z. T. an Komplexität den eigentlichen Mikroprozessor, z. B. Datenübertragungscontroller.

Die Mikroprozessoren selbst sind mit Hilfe dieser Bausteine zu Mikroprozessor-Familien und -Systemen ausgebaut worden. Sie sind im allgemeinen so konzipiert worden, daß sie ohne weitere SSI/MSI-Bausteine unmittelbar an den Mikroprozessor angeschlossen werden können. In ihnen können z. B. folgende Funktionselemente enthalten sein:

- Vervielfachen der Ein/Ausgabe-Anschlüsse
- Programmierbare Zeitglieder

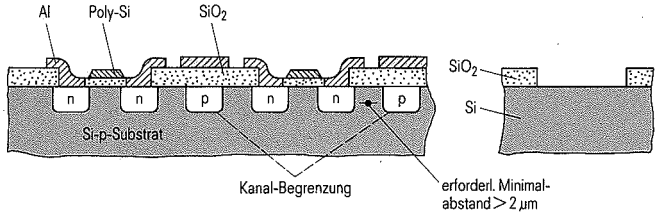


Bild 3a. Querschnitt zweier herkömmlicher N-MOS-Transistoren. Hohe und steile SiO_2-Kanten ergeben Probleme bei Al-Bahnen. Die P-dotierten Gebiete zwischen den Transistoren verhindern Ausbreitung parasitärer N-Kanäle auf Kosten der Packungsdichte

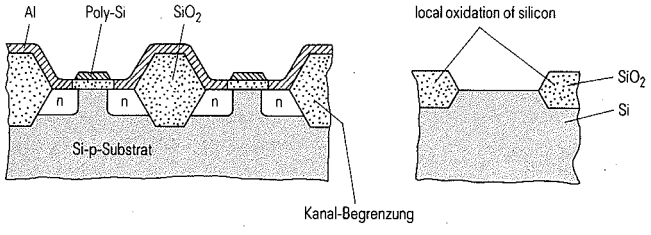
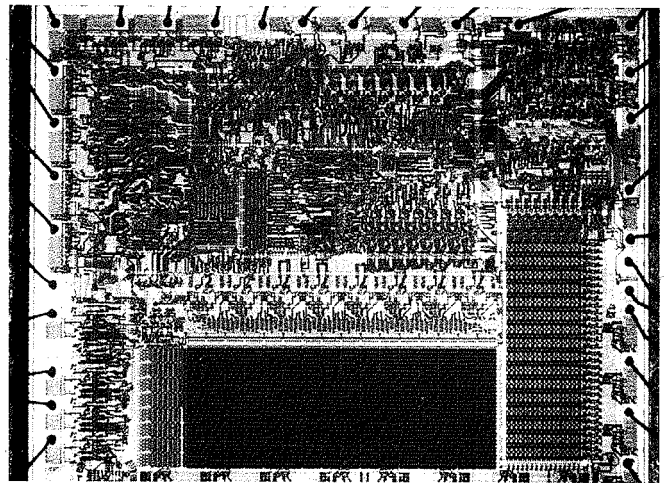


Bild 3b. Querschnitt durch zwei N-MOS-Transistoren in LOCOS-Technologie. Das SiO_2 wächst zum Teil in das Silizium hinein und unter die Maskierung. Dies ergibt kleinere und schräg verlaufende Stufen. Wegen der größeren Oxid-Dicke bilden sich keine parasitären Transistoren, deshalb kleinere zulässige Abstände; gegenüber Bild 3a rund 30 % weniger Flächenbedarf



Die µP-Peripherie-Bausteine sind oft dichter „gepackt" als die Mikroprozessoren selbst

- Serien-Parallel-Umsetzer mit Unterstützung von Standardprozeduren zur Verwendung in Datenfernübertragungsanlagen
- Festwert- und Schreib/Lesespeicher

Hinzu kommen noch spezifische Bausteine für besondere Anwendungen.

Beispielhaft für die verschiedenen Peripherie-Bausteine werden vier Bausteine kurz erläutert:

a) In *Bild 5* ist ein Ein/Ausgabe-Erweiterungsbaustein zusammen mit einem Einchip-Mikrocomputer gezeigt. Ohne zusätzliche Decodierungsaufwendungen, d. h. ohne zusätzliche Gatter-Bausteine lassen sich an den 8048 vier Erweiterungsbausteine anschließen, wobei jeder Baustein wiederum 4 x 4-Ein/Ausgabe-Anschlüsse ermöglicht. Durch die Einschränkung auf Gruppen zu je 4 bit wird die Bearbeitung von Dezimalzahlen betont. Besonderes Kennzeichen dieses Bausteins ist seine Programmierbarkeit. Die beiden Funktionen, Eingabe oder Ausgabe, werden während des Programmablaufs (d. h. dynamisch) durch den Mikrocomputer festgelegt. Ebenso können die Daten innerhalb des Bausteins mit weiteren Daten aus dem Prozessor durch UND- oder ODER-Funktionen verknüpft werden.

b) In *Bild 6* ist ein mikroprogrammierbarer Doppel-Prozessor dargestellt mit einer speziellen Architektur für die Steuerung serieller Daten. Dieser Baustein unterstützt den eigentlichen System-Prozessor durch autonomes Abarbeiten komplexer Befehle, wobei die Daten 8 bit parallel auf dem System-Bus vorliegen und seriell abgegeben oder aufgenommen werden. Mit einer derartigen Konzeption ist bereits ein Systembaustein definiert worden, der die komplexe Steuerung von sowohl Floppy-Disk- als auch von Datenfernübertragungs-Systemen mit ihren z. T. recht komplexen Prozeduren durchführt. Die unterschiedlichen Anforderungen werden durch Mikroprogrammunterschiede realisiert, die beim Entwurf und der Herstellung des Bausteins berücksichtigt werden müssen.

c) Bekanntlich kommt es bei der Minimierung der Kosten im wesentlichen darauf an, die Anzahl der Bausteine pro System klein zu halten [7]. In *Bild 7*

ist ein Baustein dargestellt, der mehrere bisher noch getrennt aufgebaute Baugruppen wie Festwertspeicher, Schreib/Lesespeicher, Taktgenerator zu einem einzigen zusammengefaßt. Wesentlich an diesem Baustein ist jedoch, daß man einerseits über die programmierbare Gatter-Matrix die Anfangsadressen der Speicher auf dem Chip festlegen und andererseits gleichzeitig Adressen speziellen Ausgangsanschlüssen zuordnen kann, wodurch zusätzliche Decodierlogik für weitere periphere Bausteine eingespart wird.

d) Ein Beispiel für eine „festverdrahtete" Lösung einer speziellen komplexen Funktion, nämlich das Darstellen und Bewegen von Figuren auf dem Schirm eines Fernsehgerätes für Spiele, ist in *Bild 8* dargestellt. Wegen der hohen Geschwindigkeitsanforderung müssen diese Funktionen parallel bearbeitet werden und führten deshalb zu einer festverdrahteten Lösung auf dem Kristall. Zusammen mit Zählern, die den Stand des Video-Strahls (horizontal und vertikal) angeben, befinden sich in dieser Schaltung Speicher, in die die Muster der Spielfiguren, des Hintergrundes und die jeweilige Farbe eingeschrieben werden. Gleichzeitig werden in diesen Speichern die jeweiligen Ortskoordinaten der Spielfiguren vom Mikroprozessor her gespeichert. Kollision mit dem Hintergrund oder anderen Figuren werden durch Koinzidenz der Video-Signale festgestellt und zur Weiterverarbeitung an den Mikroprozessor abgegeben.

4 Trends

Über die zukünftigen Entwicklungen kann man eines sagen: Die stürmische Weiterentwicklung der Halbleiter-Technologie ist ungebrochen. Der Flächenbedarf der einzelnen Bauelemente wird weiterhin sinken und die Größe der einzelnen Chips wird weiterhin wachsen (*Bild 9*). Bis 1980 erwartet man eine Verkleinerung der Minimalabstände auf dem Chip von jetzt etwa 5 µm auf etwa 2 µm und eine Ausdehnung der bearbeiteten Chip-Fläche von zur Zeit etwa 40 mm² auf ca. 140 mm², wobei man zunächst nicht gleichzeitig große Flächen und Minimalabstände von 2 µm realisieren wird. Man wird daher weiterhin versuchen, Bauelemente mit einem noch größeren Funktionsumfang zu integrieren mit folgenden Konsequenzen:

- Kurzfristig wird man eine Weiterentwicklung der Mikroprozessoren und Mikrocomputer in der Form erleben, daß die Mikrocomputer mit einer wachsenden Anzahl von Speicherelementen ausgerüstet und neue kostengünstige Anwendungen durch weitere Integration von peripheren Schaltungen ermöglicht werden.
- Es wird eine neue Generation von 8- und 16-bit-Mikroprozessoren entwickelt werden, deren Architektur und Befehlssatz besser an die höheren Programmiersprachen und an die Anwendungen, z. B. Textverarbeitung, angepaßt sind.
- Die Anzahl der Speicherelemente pro Chip wird größer und die einzelnen Speicherfunktionen billiger. Die Programmausführung kann dann dadurch beschleunigt werden, daß heute sequentiell ausgeführte Programmabschnitte durch Auslesen von Tabellenwerten in Speichern ersetzt werden.
- In zunehmendem Maße werden hochspezielle, komplexe Systemkomponenten auch von den Halb-

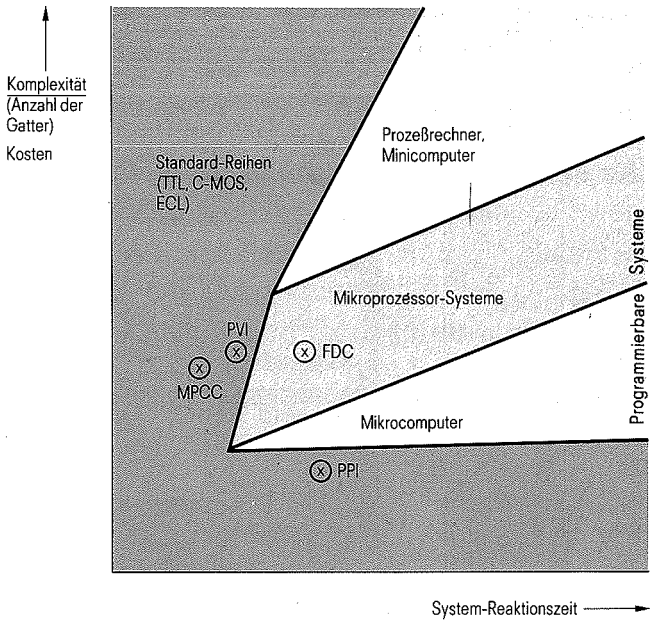


Bild 4. Anwendungsbereiche von Computern und LSI-Peripherieschaltungen (FDC = Floppy Disk Controller, MPCC = Multi Protocol Communication Controller, PVI = Peripheral Video Interface, PPI = Programmable Peripheral Interface)

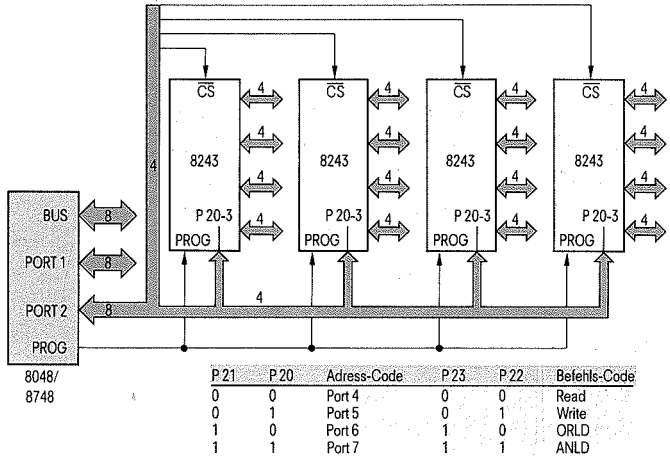
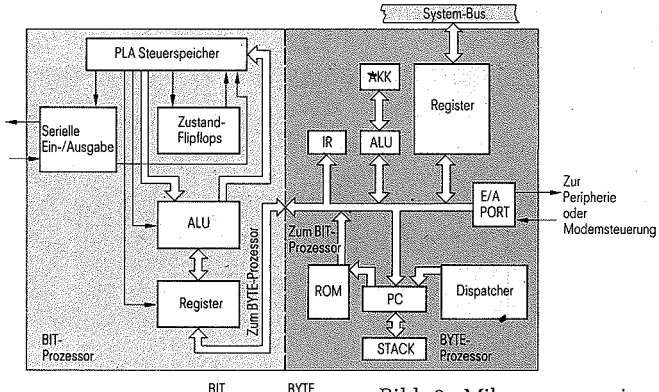
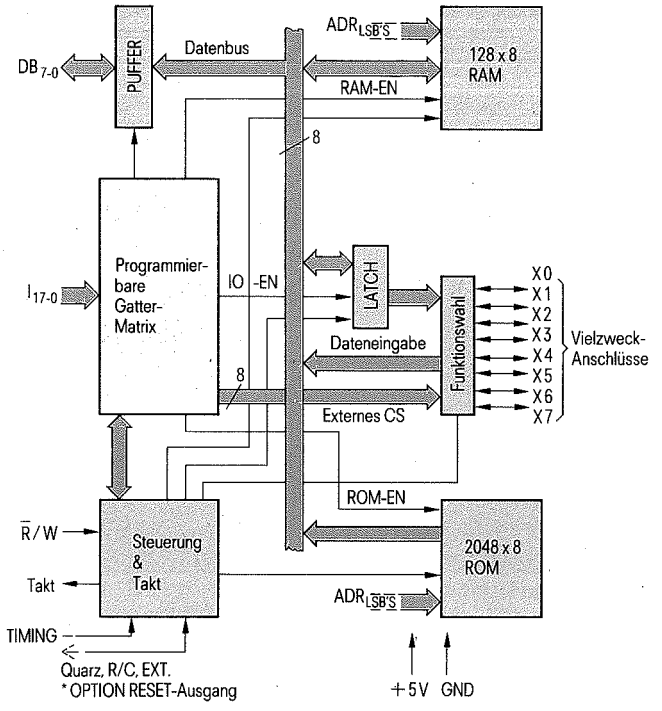


Bild 5. System mit vier Ein-/Ausgabe-Erweiterungsbausteinen



	BIT	BYTE
Anzahl der Befehle	./.	46
Befehlsausführungszeit	250 ns	1,25 µs
Wortbreite des Mikrobefehls	33	8
Größe des Mikroprogrammsp.	3300	6912
Anzahl der Bauelemente	4600	13.500

Bild 6. Mikroprogrammierbarer Doppelprozessor für Datenübertragungsaufgaben



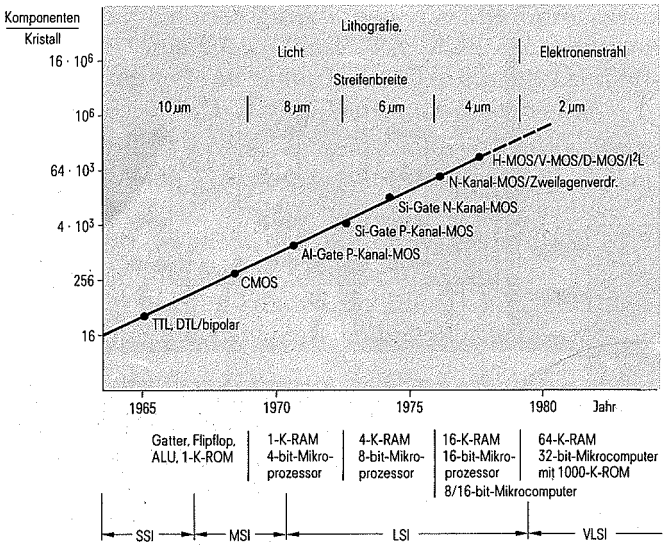
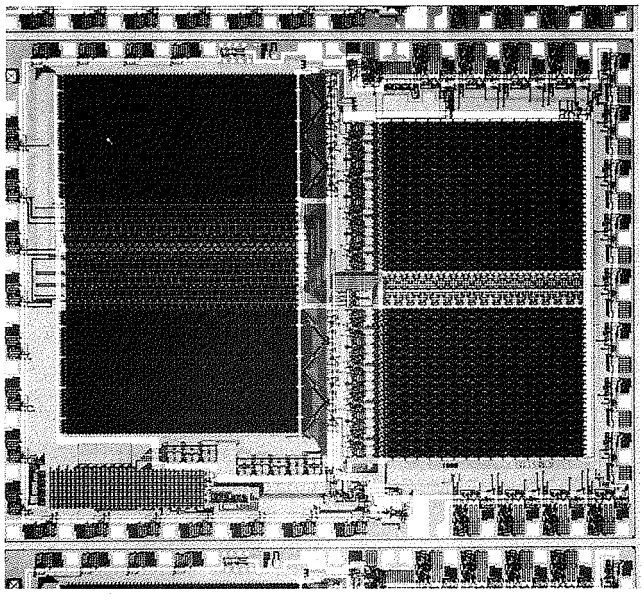


Bild 9. Technologische Entwicklung bei integrierten Schaltungen



Kristallfoto des LSI-Peripheriebausteins 2656

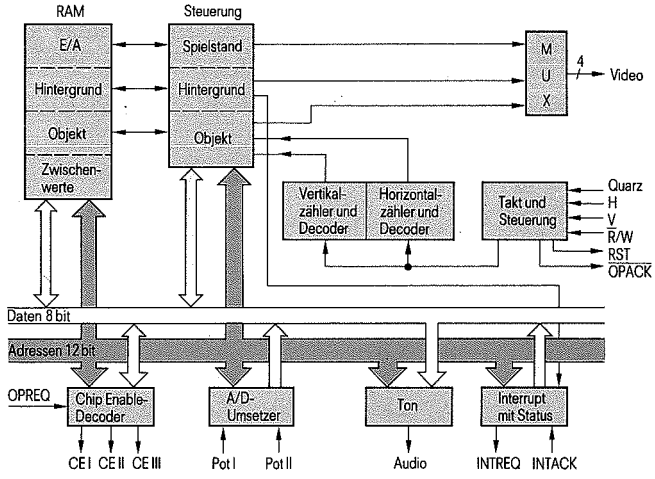


Bild 8. Programmierbarer Video-Interfacebaustein

◀ Bild 7. System- und Speicherbaustein Typ 2656

leiter-Herstellern so entwickelt, daß ein allgemein verwendbarer Basisbaustein durch eine Art Mikroprogrammierung auf seine endgültige Aufgabe hin festgelegt wird.

- Die elektrisch veränderbaren Eigenschaften von MOS-Transistoren, wie sie in den EPROMs verwendet werden, führen in der Zukunft zu einem weiteren Grad der Flexibilität, wobei der gleiche Preisverfall durchlaufen wird, wie er in der Halbleiter-Industrie bisher üblich war.
- Da man die Verlustleistung pro Fläche im Silizium nicht erhöhen kann, die Bauelemente-Dichte jedoch zunehmen soll, wird man nicht umhin können, mit der Einführung dieser kleinen Strukturen auch die Versorgungsspannung von bisher 5 V auf etwa 3 V herabzusetzen. Die Spannung von 5 V war bisher durch die eingeführte TTL-Schaltungsfamilie vorgegeben. Durch die Ablösung dieser Familie und die bessere Beherrschung der MOS-Technik ist man nicht mehr an den Wert 5 V gebunden.
- Die zukünftigen Entwicklungsplätze müssen diesen Entwicklungen Rechnung tragen. Einerseits muß die Hardware-Seite genügend flexibel ausgelegt sein, andererseits bedarf es einer hinreichenden Software-Unterstützung, um dem Anwender die Umsetzung seines Anwendungs-Know-how's in die umfangreicher werdende Anwender-Software zu erleichtern.

Literatur

a) allgemeine Literaturhinweise

[1] *Moore, G.:* Progress in Digital Integrated Electronics. Device Conference, Washington 75.

[2] *Lipovski, G. J.:* Further remarks on the microprocessor of the future. ACM, Computer Architecture News, Vol. 6 Number 3, August 1977.

[3] *Linvill, J., Hogan, C. L.:* Intellectual and economic fuel for the electronics revolution. Science, 18.3.1977.

b) spezielle Literaturhinweise

[4] *Verhofstadt, Peter W. J.:* Microprocessor architecture in perspective. Tagungsberichte Mikroelektronik 1976 in München.

[5] *Stamm, D. et al:* A single chip, highly integrated, user programmable microcomputer. ISSCC (1977).

[6] *Apples, J. A.* et al: Local oxidation of silicon and its applications. Philips Research Reports, Vol. 25, Number 2 (April 1970).

[7] *Pahley, R.* et al: H-MOS scales traditional devices to higher performance level. Electronics, August 18, 1977.

[8] *Hoffmann, H.:* Festverdrahtete Logik contra Mikroprozessoren. etz b, Heft 15, Jahrgang 28 (1976).

[9] *Glenn, L.* et al: A dual processor serial data controller chip. ISSCC (1977).

Hermann Schmid,
MSEE

Obwohl es Mikroprozessoren erst seit knapp acht Jahren gibt, ist das Angebot an monolithischen Prøzessor-, Speicher- oder Hilfsbausteinen bereits sehr umfangreich, außerdem werden ständig neue Bauteile vorgestellt. Daher wird es immer schwieriger, zu erfassen, was diese Bausteine leisten und wie gut sie arbeiten. Dieser Überblick behandelt monolithische CPU-Bausteine und befaßt sich mit Mehrchip-Prozessoren und Einchip-Computern.

Monolithische Mikrocomputer-Bausteine

Eine Übersicht

1 Wo stehen wir heute, wohin geht die Entwicklung?

Monolithische (Mikro-)Prozessoren gibt es erst seit kurzem, trotzdem haben diese bereits die Elektronik und die mit ihr verbundenen Industriezweige revolutioniert. Was vor nicht allzu langer Zeit mit dem CPU-Baustein 8008 als Lösungsversuch für ein spezielles Problem begann, hat sich zu einer Lawine monolithischer Prozessor-, Speicher- und Hilfsbausteine entwickelt, die einen Markt mit einem Volumen von Milliarden DM repräsentieren.

Aus dieser ersten Generation (8008) ging Mitte der siebziger Jahre die zweite Generation (8080, 6800 usw.) hervor, die schon über den zehnfachen Durchsatz verfügte. Mittlerweile sind wir Zeugen der Geburt einer neuen Generation von Mikroprozessoren, zu der die Typen 8086, Z8000 und andere gehören, deren Leistung nochmals um eine Größenordnung höher liegt. Zusätzlich wurden zahlreiche Speicher-Bausteine in verschiedenen Ausführungen, eine Vielzahl monolithischer Ein/Ausgabe- und Peripherie-Steuerbausteine, sowie eine große Zahl der verschiedensten Hilfs-Verarbeitungs-Bausteine entwickelt.

Trotzdem ist die Mikro-Revolution noch keineswegs vorüber. Im Gegenteil, sie hat gerade erst begonnen. Was wir bis jetzt gesehen haben, ist nur die Spitze des Eisbergs, die wirkliche Revolution dagegen steckt noch in ihren Anfängen.

Schon in Kürze werden wesentlich größere, kompliziertere und leistungsfähigere Bausteine auf den Markt kommen. Unter den Halbleiterherstellern gibt es offensichtlich nur geringe Zweifel, daß Mitte der achtziger Jahre VLSI-Schaltungen erhältlich sein werden, die über eine Million aktive Elemente enthalten. Mit Bausteinen dieser Dimension, die immer noch um Größenordnungen von den theoretischen Grenzen entfernt sind [1], liegt die Anzahl und Komplexität der Verarbeitungsfunktionen, die sich implementieren lassen, schon nahezu jenseits des Vorstellungsvermögens: Möglich werden dann beispielsweise Megabit-Speicher, Mainframe-Computer mit großem Programm- und Datenspeicher, oder auch Mehrfach-Prozessoren (z. B. ein Vierfach-PDP-11) usw. Als Folge davon werden Abmessungen und Kosten digitaler Verarbeitungs-Schaltungen praktisch gegen Null gehen.

Im Gegensatz dazu werden spezielle digitale Schaltungen, analoge Präzisions-Ein/Ausgabeschaltungen, sowie Treiber für hohe Spannungen, Ströme oder Leistungen diesem Trend nicht folgen, oder allenfalls wesentlich langsamer.

Das wesentliche Problem bleibt jedoch vor allem die Software, deren Kosten oft die Hardwarekosten um den Faktor zehn übersteigen können. Die Ursache dafür ist wohl weitgehend in der Tatsache zu suchen, daß Mikroprozessoren bis jetzt vor allem von Hardware-Spezialisten entwickelt wurden, mit dem Ziel, den Hardware-Aufwand auf das Minimum zu reduzieren. Daß sich auf diese Weise das erklärte Ziel einer Minimisierung der Gesamtkosten des Systems nicht erreichen läßt, wird inzwischen praktisch von niemanden mehr bestritten.

Ein weiterer Grund ist die Tatsache, daß die meisten der gegenwärtig erhältlichen Mikroprozessoren Kopien bereits vorhandener Minicomputer sind, die nach dem von John von Neumann entwickelten Prinzip der sequentiellen Instruktionsausführung in der CPU arbeiten. Es muß in diesem Zusammenhang jedoch darauf hingewiesen werden, daß von Neumann diese Arbeitsweise konzipierte, als aktive Bauelemente noch langsam und teuer und lediglich die Verdrahtung schnell und billig war. Durch die Verwendung von LSI- oder VLSI-Schaltungen hat sich diese Situation jedoch ins Gegenteil verkehrt. Die Folge davon ist, daß die Notwendigkeit zur Verringerung der Anzahl der aktiven Bauelemente, oder allgemein der Hardware, nicht mehr länger besteht.

Dagegen ist es jedoch besonders wichtig, die Kosten für die Software-Entwicklung, die Dokumentation,

ihre Überprüfung und die Programmpflege zu reduzieren. Fernerhin wird leicht einsetzbare Software benötigt, um diese Technologie einem großen Anwenderkreis zugänglich zu machen. Zukünftige Software muß dem Anwender die Komplexität und Kompliziertheit transparent machen, oder anders gesagt, die ganze verwickelte Hardware vor ihm verbergen.

Die Software befindet sich heute dort, wo die Hardware vor einem Jahrzehnt stand: Auf dem SSI-Niveau. Die zahlreichen höherentwickelten Programmsprachen, die zur Zeit für Mikrocomputer eingeführt werden, heben die Software auf die MSI-Ebene an. Jedoch sind auch bei der Software noch wesentlich höhere Integrationsgrade erforderlich, wenn Mikroprozessoren eines Tages von so großen Verbrauchergruppen, wie sie z. B. Hausfrauen oder Schüler darstellen, verwendet werden sollen. Im Augenblick ist jedoch kein Durchbruch in dieser Hinsicht zu erkennen.

Im Bemühen, gegenwärtig übliche Verfahren bei der Entwicklung von Mikroprozessoren umzudrehen, hat ein kleineres Unternehmen in Kalifornien (Western Digital) kürzlich eine revolutionäre Lösungsmöglichkeit angekündigt: Einen Mikroprozessor, der auf der Basis einer Höheren Programmiersprache (PASCAL) entwickelt wurde. Die „PASCAL-Mikro-Maschine" ist damit der erste Prozessor, der Instruktionen direkt in einer Höheren Sprache ausführen kann.

Inzwischen stellt auch das Konzept des einzelnen, zentralen Prozessors hoher Leistung nicht mehr bei allen Problemen die günstigste Lösung dar. Stattdessen wird die Verwendung von Multi-Prozessorsystemen vorgeschlagen. Es ist nämlich nicht nur wesentlich wirtschaftlicher, viele kleine Prozessoren anstelle eines großen aufzubauen, sondern es ist auch einfacher, kleine und unabhängige Programme zu entwickeln, statt eines sehr umfangreichen Programms mit vielen zueinander in Beziehung stehenden Modulen. In Zukunft werden daher in verstärktem Umfang Systeme mit paralleler oder verteilter Verarbeitung Verwendung finden. Es gibt eine Reihe von Gründen dafür, warum diese Systeme in der Vergangenheit nicht zahlreicher verwendet wurden. Zunächst einmal standen die erforderlichen monolithischen Einchip-Prozessoren mit ihren Speicher- und Ein/Ausgabeschaltungen noch nicht zur Verfügung, darüber hinaus gab es aber auch die zum Betrieb solcher Multi-Prozessor-Systeme erforderliche Software noch nicht. Ein weiterer Grund ist schließlich darin zu sehen, daß für das Problem des Datenverkehrs zwischen diesen Prozessoren bis jetzt noch keine ideale Lösung gefunden wurde.

Offenbar ist die „Mikroprozessor-Revolution" bisher doch mehr eine „Evolution" gewesen. Der in den letzten Jahren erzielte Fortschritt war mehr oder weniger vorhersagbar, als sich die Integrationsdichte von SSI über MSI zu LSI weiterentwickelte. Sobald jedoch erst einmal die Möglichkeit besteht, zuverlässig VLSI-Schaltungen zu produzieren, bestehen keine Beschränkungen mehr hinsichtlich der Integrationsdichte, sondern vielmehr nur noch im Hinblick auf die Software. Sobald auch dieses Problem gelöst ist, sind Grenzen nur noch in bezug auf die Anwendungen zu sehen. Dann wird die eigentliche Revolution beginnen, denn spätestens ab diesem Zeitpunkt wird sich nicht mehr genau vorhersagen lassen, was sich ereignen wird. Der Erfolg wesentlicher technischer Entwicklungen hängt dann nicht länger von der Fähigkeit des Ingenieurs zu ihrer Verwirklichung ab, sondern von den Launen und Wünschen großer Verbrauchergruppen.

Hier soll ein Überblick über die jüngsten Entwicklungen auf dem Gebiet der Prozessor-Hardware gegeben werden. Wegen der großen Zahl neu vorgestellter oder gerade in Entwicklung befindlicher monolithischer Bausteine, werden nur die wichtigsten Prozessoren mit ihren wesentlichen Merkmalen und der Angabe ihres momentanen Entwicklungsstandes beschrieben.

2 Hochintegrierte Bausteine (VLSI)

VLSI ist das Verfahren zur gleichzeitigen Herstellung von Millionen Transistoren auf einem dünnen Silizium-Wafer, unter Verwendung lithographischer Herstellungs- und chemischer Verarbeitungsprozesse. Lithographische Herstellung umfaßt die Erzeugung topologischer Bauelemente- und Schaltungsmuster im Maßstab 1000:1 – eins für jeden Verarbeitungsschritt –, ihre Verkleinerung und die Übertragung auf einen Silizium-Wafer (5...10 cm Durchmesser), so daß ausgewählte Flächen anschließend der chemischen Bearbeitung unterzogen werden können. Der große Wafer wird anschließend in kleinere Chips zerschnitten, von denen jeder die gewünschten VLSI-Schaltungen enthält. Zum Schluß werden die Chips in der jeweiligen Gehäuseform eingekapselt und getestet.

Praktische und wirtschaftliche Gesichtspunkte beschränken zur Zeit die gesamte Verlustleistung von VLSI-Chips auf weniger als 1 W und die Abmessungen auf weniger als 7,5 x 7,5 mm^2, ganz unabhängig davon, wieviele aktive Elemente sich auf dem Chip befinden. Um eine Erhöhung der Dichte zu erreichen, müssen die Verlustleistung je Element sowie seine Größe proportional zur Anzahl der aktiven Elemente auf dem Chip abnehmen. Während der letzten zehn Jahre war eine Verringerung der Elementgröße und der Verlustleistung auf ein Tausendstel zu beobachten. Gleichzeitig reduzierte sich die Ausbreitungsverzögerungszeit auf ein Hundertstel ihres ursprünglichen Wertes, so daß das Produkt aus Verlustleistung und Ausbreitungsverzögerungszeit um fünf Größenordnungen abnahm.

Im gleichen Zeitraum hat sich die Dichte integrierter Schaltungen, entsprechend der Regel von Moore-Noyce, Jahr für Jahr verdoppelt. Dieser Prozeß wird sich noch einige Zeit fortsetzen, so daß Mitte der achtziger Jahre die Herstellung von Chips mit bis zu einer Million aktiver Elemente möglich sein dürfte (*Bild 1*). Da nur ein geringes Anwachsen der Chipfläche zu erwarten ist, muß eine Erhöhung der Integrationsdichte im wesentlichen durch eine Reduzierung der Abmessungen der aktiven Elemente erzielt werden.

Durch eine Verkleinerung bzw. ein „Einschrumpfen" aller drei Dimensionen eines aktiven Elementes verringern sich nicht nur seine Abmessungen und die zugehörigen parasitären Parameter (im wesentlichen die Kapazität), sondern es ergibt sich auch eine Verstärkung des elektrischen Feldes zwischen den verschiedenen Elektroden, sofern nicht gleichzeitig die Versorgungsspannungen reduziert werden. Ferner ist festzustellen, daß sich der Strom, die Kapazitäten und die Ausbreitungsverzögerungszeit proportional verringern, während die Fläche und die Verlustleistung quadratisch abnehmen. Das Produkt aus Verlustleistung und Ausbreitungsverzögerungszeit nimmt sogar proportional zur dritten Potenz der Verkleinerungskonstanten K ab, wie es die *Tabelle 1* zeigt.

Während des letzten Jahrzehnts wurden viele neue Herstellungsprozesse für Halbleiter entwickelt, mehr als 15 bipolare und über 30 MOS-Verfahren, die alle zu deutlich höheren Chip-Dichten und beträchtlich gesteigerten Leistungen führten [2]. Trotzdem sind die heutigen VLSI-Schaltungen noch immer weit vom Ideal entfernt: Sie sind zu langsam, verbrauchen zu viel Leistung und sind nicht genügend hochintegriert, um die Integration von einer Million aktiver Bauelemente auf einem Chip zu ermöglichen.

Bild 2 verdeutlicht, wo die drei meistverwendeten LSI-Techniken zur Zeit im Geschwindigkeits-Leistungsbereich liegen (durchgezogene Linien) und wo sie sich vermutlich in wenigen Jahren befinden werden (gepunktete Linien). Es werden zahlreiche Anstrengungen zur Reduzierung des Produktes aus Leistung und Verzögerungszeit – einer sehr zweckmäßigen Größe zur Kennzeichnung der erzielten Verbesserung – bei diesen Herstellungs-Technologien unternommen. Darüber hinaus werden aber auch viele neue Techniken entwickelt, da die gegenwärtig bekannten wenig Anlaß zur Hoffnung geben, dieses Produkt jemals auf einen Wert 0,1 pJ zu drücken. Am vielversprechendsten scheint die Verwendung von Gallium-Arsenid-Material zu sein. Eine Reihe amerikanischer und japanischer Hersteller untersuchen zur Zeit VLSI-Techniken unter Verwendung von GaAs. Um eine Million aktive Elemente auf einem Chip zu integrieren, ist jedoch immer noch eine Verringerung des Produktes aus Leistung und Ausbreitungsverzögerung auf ein Hundertstel erforderlich.

Die Möglichkeiten zur Herstellung von VLSI-Schaltungen durch photolithographische Prozesse sind dann voll ausgeschöpft, wenn die Linienbreite oder kleinste auflösbare Länge in der Größenordnung der Wellenlänge des sichtbaren Lichts liegt (ungefähr 0,4...0,7 µm). Bevor darüber hinaus eine weitere Steigerung der Bauelemente-Dichte möglich ist, müssen neue Techniken entwickelt werden, die eine Reduzierung der Linienbreite auf weniger als 1 µm erlauben.

Elektronenstrahl-Lithographie (*EBL*) und Röntgenstrahl-Lithographie (*XRL*) könnten diese Möglichkeit bieten, zunächst noch durch Belichtung der Silizium-Wafer mit Hilfe von Masken, später durch direkte Belichtung.

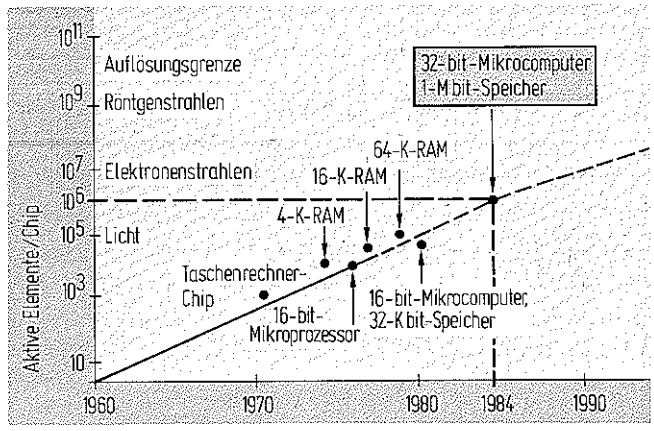


Bild 1. Das Diagramm von Texas Instruments zeigt die seit 1960 stetige LSI-Wachstumsrate (Verdopplung im Zeitraum von jeweils zwei Jahren) und sagt Megabit-Speicher sowie monolithische 32-bit-Prozessoren (mit CPU, Speicher, Ein/Ausgabe) für 1984 voraus

In jüngster Zeit erzielte Fortschritte sowohl bei der EBL- [3] als auch der XRL-Technik [4], haben die meisten Halbleiter-Hersteller davon überzeugt, daß diese Verfahren in den nächsten Jahren rasch ausreifen und ihre gegenwärtigen Probleme überwinden werden, und zwar besonders im Hinblick auf die Geschwindigkeit, die noch hohen Kosten und die Justage-Genauigkeit.

In der Submikron-Technologie steckt die Möglichkeit zur Steigerung der Chip-Dichte um den Faktor zehn, bei gleichzeitiger Reduzierung des Leistungs-Verzögerungszeit-Produktes auf ein Hundertstel, wodurch Chips mit einer Million aktiver Bauelemente möglich würden. Damit dürften Entwickler die Möglichkeit haben, auch die komplexesten Funktionen auf einem einzigen Chip zu implementieren, so daß die Abmessungen und Kosten digitaler Hardware äußerst niedrige Werte erreichen werden.

Das große Angebot an monolithischer Verarbeitungs-Hardware kann in Prozessor-, Speicher- und Hilfsbausteine unterteilt werden. Jeder digital arbeitende Computer benötigt diese Komponenten. Single-Board-Computer verwenden sie als getrennte Bausteine, monolithische Computer integrieren sie auf einem einzigen Chip.

Die ideale Lösung für dieses Überangebot wäre ein Einchip-Prozessor mit hoher Arbeitsgeschwindigkeit und Auflösung, sowie einem Speicher, der für die meisten Anwendungen groß genug ist. Derartige universelle Prozessor-Elemente sind zur Zeit aber weder praktisch noch wirtschaftlich. Die Hersteller monolithischer Prozessor-Hardware müssen ihre Produkte

Tabelle 1

Schaltungsparameter	reduziert um
Strom I	K
Kapazität WL/t_{ox}	K
Verzögerungszeit je Gatter UC/I	K
Fläche	K^2
Verlustleistung UI	K^2
Geschwindigkeits-Leistungs-produkt (UC/I) · (UI)	K^3

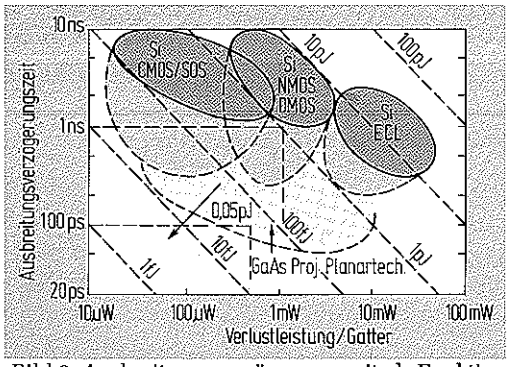


Bild 2. Ausbreitungsverzögerungszeit als Funktion der Verlustleistung bei gebräuchlichen MOS-, ECL- und GaAs-Verfahren; die durchgezogenen Linien gelten für zur Zeit übliche Prozesse, die gestrichelten für zukünftige Verfahren

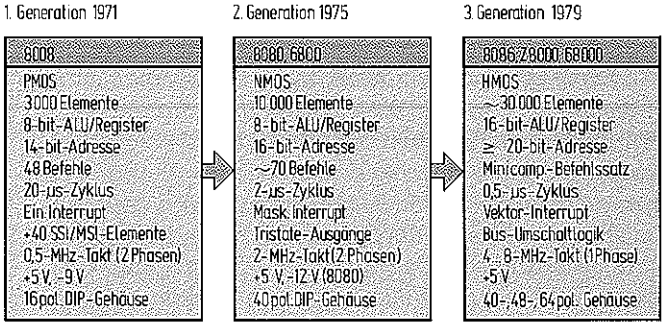


Bild 3. Drei CPU-Generationen zeigen einen deutlich gestiegenen Integrationsgrad sowie höhere Chipdichte und Arbeitsgeschwindigkeit, außerdem eine Verfeinerung des Befehlssatzes

daher für spezielle Anwendungen auslegen und ausgewählte Parameter für die preisgünstigste Lösung optimieren. Monolithische Verarbeitungsbausteine können in Einchip-CPUs mit festem Instruktionssatz, mikroprogrammierbare Mehrchip-CPUs (*n-bit-Slices*) und monolithische bzw. Einchip-Prozessoren (mit CPU, Speicher, Ein/Ausgabe) unterteilt werden.

3 Einchip-Zentraleinheiten (CPU)

Die CPU führt arithmetische und logische Operationen aus und verschiebt Datenworte. Sie besteht aus der Arithmetik-Logik-Einheit, internen Registern, dem Befehlsdecodierer und der Steuer-Logik, sowie dem internen Datenbus mit Multiplexern und Ein/Ausgabe-Puffern. Die Verbindung zwischen der CPU, dem Speicher und der Umwelt erfolgt über getrennte oder auch im Multiplexverfahren betriebene Adreß-, Daten- und Steuerbusse. Gegenwärtig erhältliche Einchip-CPUs haben feste Instruktionssätze und sind daher vom Anwender nicht mikroprogrammierbar. Bis jetzt wurden drei CPU-Generationen vorgestellt (*Bild 3*).

Die erste CPU-Generation in PMOS-Technik (8008, 4004) ist durch Zykluszeiten von 20 µs und einfache Instruktionssätze gekennzeichnet, erfordert viele Hilfsschaltungen und benötigte bis zu drei verschiedene Versorgungsspannungen. Oft war der Taktgenerator größer und teurer als die CPU.

Die zweite CPU-Generation in NMOS-Technik (6800, 8080) bietet eine auf ein Zehntel verkürzte Zykluszeit (2 µs) sowie leistungsfähigere Instruktionssätze und benötigt außerdem weniger Hilfsschaltungen.

Die Mikroprozessoren Z 80 oder 8085 beispielsweise müssen eigentlich als Zwischenschritt von der zweiten zu der zu erwartenden dritten Generation angesehen werden. Diese Bausteine verwenden eine verbesserte NMOS-Technologie, arbeiten mit der doppelten Geschwindigkeit des Typs 8080 (1 µs), verfügen über umfangreiche Instruktionssätze (158 Instruktionen beim Z 80), enthalten den Taktgenerator auf dem Chip und benötigen nur eine einzelne Versorgungsspannung von +5 V, sowie praktisch keine Hilfs-Hardware.

Die nachfolgend aufgeführten Eigenschaften der gegenwärtig erhältlichen 8-bit-Einchip-CPUs [5] sind als typisch anzusehen:

- NMOS-Technologie,
- Zykluszeiten von 1...2 µs (Registeraddition),
- zwischen 50 und 80 Makroinstruktionen (aber nur selten besteht die Möglichkeit zur Zwei-Byte-Multiplikation oder Division),
- getrennte Adreß- und Datenbusse (nur der 8085 hat einen im Multiplex betriebenen Bus),
- registerorientiert (wenige CPUs bieten speicherorientierte Architekturen),
- 40polige DIL-Gehäuse.

Die dritte CPU-Generation wird gegenwärtig vorgestellt. Sie ist vor allem durch 16-bit-CPUs mit der Leistung von Minicomputern gekennzeichnet. Bereits jetzt ist eine beträchtliche Anzahl erhältlich (*Tabelle 2*), einige weitere sollen in Kürze eingeführt werden.

Zwischen 8-bit- und 16-bit-CPUs die Grenze zu ziehen ist nicht ganz einfach. Einige der 16-bit-Bausteine beispielsweise haben nur eine 8-bit- (CP1600) oder gar 4-bit-Arithmetik-Logik-Einheit (9440), andere dagegen nur einen 8-bit-Datenbus (TMS 9980, 6809). Trotzdem werden sie als 16-bit-CPUs angesehen, da sie vorzugsweise zur Verarbeitung von 16-bit-Worten entwickelt wurden.

3.1 TMS 9900

Die ausgereifteste, zur Zeit erhältliche 16-bit-CPU ist der Typ TMS 9900 von Texas Instruments [6]. Zu seinen herausragendsten Eigenschaften gehören:

- fortschrittliche Speicher-Architektur für eine wirkungsvolle Handhabung von Interrupts,
- Instruktionsworte von 16 bit Länge erhöhen die Verarbeitungsgeschwindigkeit und die Programmier-Flexibilität,
- vollständiger Minicomputer-Instruktionssatz, einschließlich einer 17-µs-Multiplikation und einer 33-µs-Division (3-MHz-Takt),

- getrennte Daten-, Adreß-, Ein/Ausgabe-, Interrupt- und Kontroll-Bus-Strukturen, die zahlreiche externe Komponenten überflüssig machen,
- 16 Prioritäts-Interrupt-Ebenen,
- direkte und DMA-Ein/Ausgabemöglichkeiten,
- serielle Ein/Ausgabe zur wirkungsvollen Bitmanipulation.

Die Geschwindigkeit des TMS 9900 ist jedoch relativ niedrig; eine Register-Register-Addition dauert beispielsweise 4,7 µs, von einem Register zum Speicher sogar 7,6 µs.

3.2 Der Typ 9440

Obwohl seine Arithmetik-Einheit nur für 4 bit ausgelegt ist, führt der Typ 9440 (*Bild 4*) den gesamten Instruktionssatz des Minicomputers Nova aus, jedoch ohne Multiplikation und Division [7]. Eine direkte 16-bit-Addition vom Speicher zum Akkumulator erfolgt, bei einer Taktfrequenz von 10 MHz, in nur 4,7 µs. Diese hohe Arbeitsgeschwindigkeit wird durch Fairchilds „*Isoplanar Integrated Injection Logic*" (I^3L) ermöglicht, die pro Gatter eine Laufzeitverzögerung von nur 5 ns aufweist.

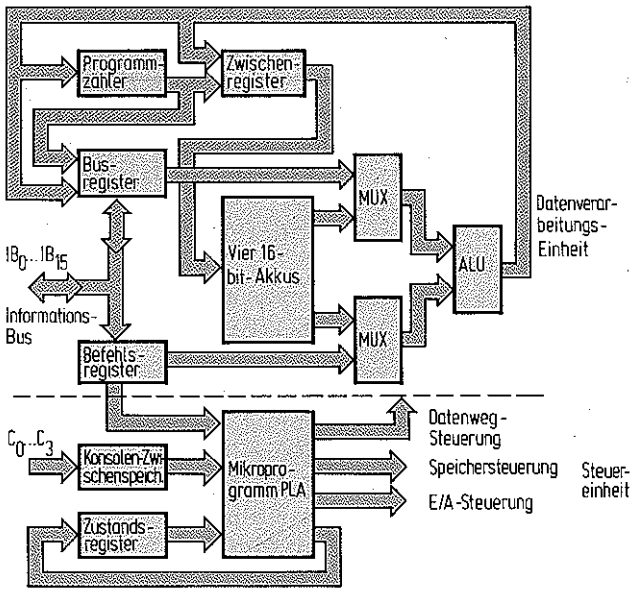


Bild 4. Bipolare 16-bit-CPU 9940 von Fairchild. Die Datenweg-Einheit enthält vier 16-bit-Akkumulatoren, vier spezielle 16-bit-Register, eine 4-bit-ALU sowie eine Kombination von 4- und 16-bit-Bussen. Die Kontrolleinheit besteht im wesentlichen aus einem maskenprogrammierbaren Logik-Array (PLA), in dem n 24-bit-Worte gespeichert werden können

Tabelle 2. Vergleichsübersicht: 16-bit-Einchip-CPU-Bausteine

Hersteller	Typ	Verfahren	Versorgungsspannung (V)	Taktfrequenz Anz. Taktph.	Wortlänge CPU	Wortlänge Datenbus	Ausführungszeiten Instruktion Fetch	Ausführungszeiten Reg.-Reg. Addition	Anzahl der Instruktionen	Busse	Bemerkungen
Data General	MN601	NMOS	+5, +10, +14	8.3 MHz/2	16	16	1.8	2.4		1	LSI NOVA
Fairchild	9440	I^3L	+5, I_{inj}	10 MHz/1	4	16	0.5	1.5	NOVA	1	emuliert die Micro-NOVA. Ausführung nach MIL-Spezifikationen erhältlich
	9445	I^3L	+5, I_{inj}	10 MHz/1	16	16		0.4	NOVA	1	
Ferranti	F100-L	Bipolar	+5	11 MHz	16	16				1	wird nur in Europa angeboten
General Inst.	CP1600A	NMOS	–3, +5, +12	4 MHz/2	8	16	2.4	7.2	87	1	erhältlich seit 1975
HP	MC^2	SOS	+12	8 MHz/1	16	16	0.375(?)	0.875	34 Klassen	2	nicht auf dem freien Markt erhältlich
Intel	8086	HMOS	+5	5 MHz/1	16	16		0.8*	133	1	zehnfacher Durchsatz im Vergleich mit der CPU 8080
Motorola	6809	NMOS	+5	5 MHz/1	8	8		0.4*	72	2	sowohl 8- als auch 16-bit-Betrieb
	68000	HMOS	+5	8 MHz/1	32	32**/ 16	0.5		61	2	arbeitet intern mit 32-bit-CPU
National	PACE	PMOS	+5, –12	1 MHz/2	16	16		16	45	2	niedr. Geschwindigk.
	INS-8900	NMOS	+5	4 MHz/1	16	16	2,0	4.0	46	2	
	8096	ZMOS	+5	5 MHz	16	16				1	
TI	TMS-9900	NMOS	12, +5, –5	3 MHz/4	16	16	1.3	4.6	67	2	seit 1976 im 64poligen DIL-Gehäuse erhältlich
	SBP-9900	I^2L	+5, I_{inj}	3 MHz/1	16	16	1.3	4.6	67	2	Ausführung nach MIL-Spezifikationen erhältlich
	TMS-9980	NMOS	+5	2.5 MHz/4	16	8	4.3	9.1	67	2	40poliges DIL-Gehäuse
Toshiba	TLCS-16	NMOS	+5	2 MHz/1	16	16					wird seit 1976 hergestellt aber nicht auf dem freien Markt angeboten
Zilog	Z8000	NMOS	+5	4 MHz/1	16	16		0.75*	110	1	ähnlich LSI PDP 11

* Zykluszeit; I_{inj} = Injections-Strom
** Intern

Die dritte Generation der 16-bit-Bipolar-Typen ist der Mikroprozessor 9445 von Fairchild. Die I^3L-Technik vereint die Eigenschaften bipolarer Schaltungen und die Bauelementedichte der VLSI-Technik. Der Instruktionssatz ist umfangreicher als der des Prozessors 9440, wobei dessen Befehle kompatibel und weitere leistungsfähige Instruktionen möglich sind. Die wichtigsten Eigenschaften der CPU 9445 sind:

- parallel mikroprogrammierte Architektur,
- elf 16-bit-Register, wovon fünf durch Software erreichbar sind,
- Möglichkeit zum Multiprozessorbetrieb,
- direkte Adressierung von 128 KByte, bis zu 16 MByte durch Memory-Mapping möglich,
- statischer Betrieb (0...15 MHz Taktfrequenz),
- einfache 5-V-Betriebsspannung,
- leistungsfähiger Befehlssatz mit
  - umfangreiche Stack-Manipulationsmöglichkeiten,
  - Byte-Manipulationsbefehle,
  - Multiplikation, Division mit und ohne Vorzeichen,
  - Doppelwort-Befehle (32 bit),
  - Trap-Befehl,
  - erweiterte Steuerbefehle für Speicherverwaltung und Multiprozessorbetrieb,
- sieben Adressierungsarten
- direkter Zugriff auf bis zu 62 programmierte E/A-Anschlüsse,
- typische Ausführungszeiten bei 12 MHz sind:

– arithmetische/logische Funktion	0,33 µs
– Sprung/Sprung in Subroutine	0,33 µs
– Multiplikation (16 x 16 bit mit 32-bit-Ergebnis)	3,0 µs
– Division (32 : 16 bit)	3,66 µs
– Addition/Subtraktion mit Fließkomma	40,0 µs
– Multiplikation/Division mit Fließkomma	50,0 µs

3.3 *Der Typ MC²*

Die sehr schnelle 16-bit-Einchip-CPU MC^2 von Hewlett-Packard [2] arbeitet mit einer Taktfrequenz von 8 MHz und führt eine Addition (Register zu Register) in 875 ns aus [8]. Der für den Betrieb als Controller entwickelte CMOS/SOS-Chip enthält auf einer Fläche mit den Abmessungen 5,7 x 5,9 mm^2 10 000 Transistoren, bei einer Verlustleistung von nur 350 mW. Kennzeichnend für ihn sind folgende wesentliche Merkmale:

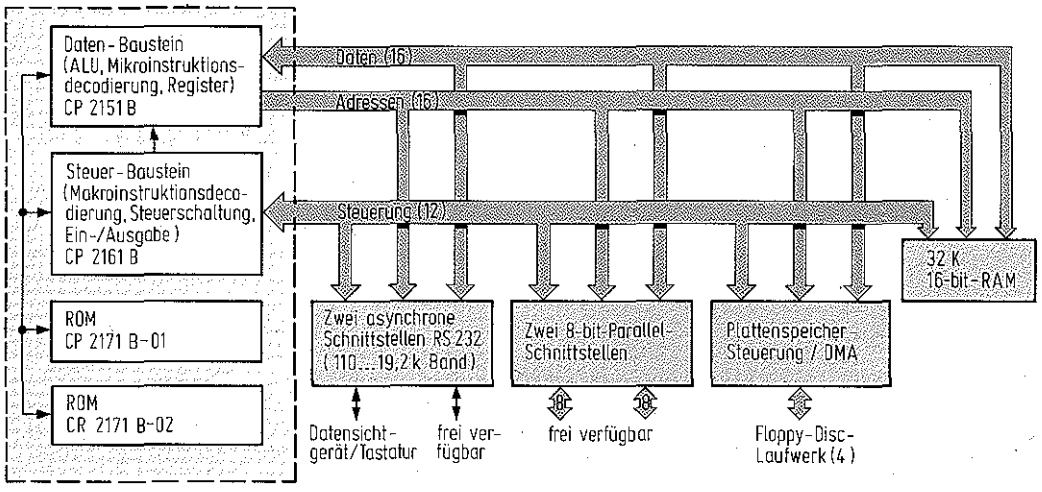
- acht allgemeine 16-bit-Register,
- getrennte Adreß- und Datenbusse,
- asynchrone Speicher- und Ein/Ausgabe-Operationen,
- völlig statischer Betrieb,
- 34 Instruktionsklassen, alle mit einer Breite von 16 bit,
- 48poliges quadratisches Keramikgehäuse.

Zur Zeit ist der Typ MC^2 noch nicht allgemein erhältlich.

3.4 *Der Typ 8086*

Der Typ 8086 von Intel war die erste der neuen 16-bit-CPUs. Kennzeichnend für sie ist ein gegenüber dem Typ 8080A-1 um den Faktor zehn höherer Durchsatz. Diese Verbesserung wird durch die Kombination der höheren Taktfrequenz (5...8 MHz) mit einem breiteren Datenwort, einer verbesserten Architektur sowie leistungsfähigerer Software erreicht [9]. Die Eigenschaften dieses in HMOS-Technik ausgeführten Chips, der auf einer Fläche von 5,7 x 5,7 mm^2 29 000 Transistorfunktionen enthält, sind beeindruckend:

- 16-bit-Arithmetik-Einheit, gekoppelt mit zwölf 16-bit-Registern, die arithmetische 8- oder 16-bit-Operationen mit binären bzw. BCD-Operanden (mit oder ohne Vorzeichen) oder auch im Zweierkomplement ausführen kann,
- Möglichkeit zur Erzeugung von 20-bit-Adressen zur Adressierung von maximal 1 MByte (direkt mit Offset, indirekt über Basis- oder Indexregister bzw. als Summe von beiden),
- umfangreicher und leistungsfähiger Instruktionssatz (einschließlich Multiplikation und Division mit und ohne Vorzeichen); auf Assembler-Ebene besteht Kompatibilität mit der Software des Typs 8080,
- einfache Implementation von Minicomputer-ähnlichen Operationen, wie beispielsweise: Positionsunabhängiger Code, dynamisch verschiebbare Programme, wiedereintrittsfähiger Code, Byte-String-Operationen und Bit-Manipulationen,



Aufbau des PME-Systems, des ersten Mikrorechners, der mit einer höheren Programmiersprache arbeitet (besprochen in Abschnitt 3.8)

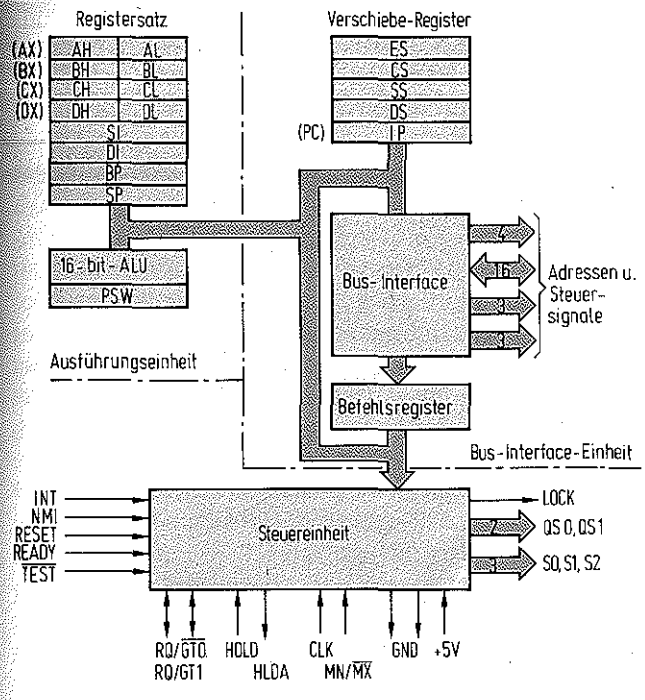


Bild 5. 16-bit-NMOS-CPU 8086 von Intel. Die Bus-Interface-Einheit (BIU) übernimmt den internen/externen Datenverkehr und erzeugt die Adressen. Die Ausführungseinheit (EU) unterwirft die gewählten Operanden den gewünschten logischen oder arithmetischen Operationen, während die Steuereinheit (CU) die Instruktionen decodiert und die Arbeit der BIU und der EU steuert

- Soft- und Hardware-Möglichkeiten zur Implementation von Multiprozessor-Systemen mit einer speziellen Bus-Zuteilungslogik,
- Hardware-Unterstützung für die Benutzung der leistungsfähigen, höheren Programmsprache PLM-86,
- leistungsfähige Interrupt-Struktur mit 256 Service-Code-Speicherplätzen für maskierbare und nichtmaskierbare Interrupts.

Die CPU 8086 (*Bild* 5) hat eine leistungsfähige Register-Architektur mit drei voneinander nahezu unabhängigen Bereichen: Die Bus-Interface-Einheit, die Ausführungs-Einheit und die Steuer-Einheit. Die Bus-Interface-Einheit erzeugt Adressen, gibt sie an den Speicher weiter, empfängt Instruktionen und Daten und überträgt sie zu den entsprechenden Registern. Die Ausführungs-Einheit wählt die gewünschten Operanden aus, verbindet sie mit der Arithmetik-Logik-Einheit, führt die erforderlichen Operationen durch und speichert das Ergebnis im vorgesehenen Register. Die Steuereinheit decodiert die Instruktion, fragt den internen Status, sowie die externen Kontroll-Eingänge ab und erzeugt daraus alle benötigten Steuersignale. Von besonderem Interesse sind das 6-Byte-Instruktionsregister, sowie die Trennung der acht allgemeinen Register von den vier Segment-Registern. Das spezielle Instruktions-Register (*instruction queue*), das man sonst in dieser Form gewöhnlich nur in größeren Computern findet, ermöglicht es, Instruktionen schon im voraus aus dem Speicher zu holen, um die Wartezeit zwischen den Befehls-Holphasen zu beseitigen. Die vier Segment-Register ermöglichen die die Verschiebung von Programmen und die Erweiterung der Adressenkapazität der CPU.

3.5 M6809

Der Typ 6809 von Motorola ist mit einer 8-bit-ALU, sowie einem ebenfalls 8 bit breiten externen Datenbus ausgestattet. Daher muß dieser Baustein als 8-bit-CPU mit umfangreichen 16-bit-Verarbeitungsmöglichkeiten angesehen werden [10]. Der Prozessor enthält eine 8-bit-ALU, vier 8-bit- und fünf 16-bit-Register, sowie einen internen 16-bit-Datenbus (*Bild* 6). Extern sind ein 16-bit-Adreßbus sowie ein 8-bit-Datenbus vorgesehen, so daß sich vollständige Kompatibilität mit dem Bussystem des Typs 6800 ergibt. Die beiden Akkumulatoren A und B können zu einer Länge von 16 Bit vereint werden, wenn Operationen mit 16-bit-Worten ausgeführt werden sollen. Ein zusätzliches Index-Register Y erweitert die Indizierungsmöglichkeiten des Prozessors, während ein zweiter „Anwender"-Stapelspeicher die Techniken des Daten-Managements verbessert.

Der Instruktionssatz des 6809 ist im Hinblick auf den Source-Code kompatibel mit Programmen für den Typ 6800, jedoch wesentlich leistungsfähiger. Dies wurde durch eine Kombination zahlreicher Befehle der CPU 6800 zu mehr allgemein gehaltenen Instruktionen, sowie sechzehn zusätzliche Instruktionen für 16-bit-Operanden (16X) erreicht. Einige der 16X-Instruktionen manipulieren Daten im Akkumulator, andere bieten die Möglichkeit zur Erzeugung oder Änderung von Adressen in den Indexregistern oder im Stapelspeicher-Zeiger. Weiterhin sind ein Befehl zur Multiplikation zweier 8-bit-Operanden, sowie eine

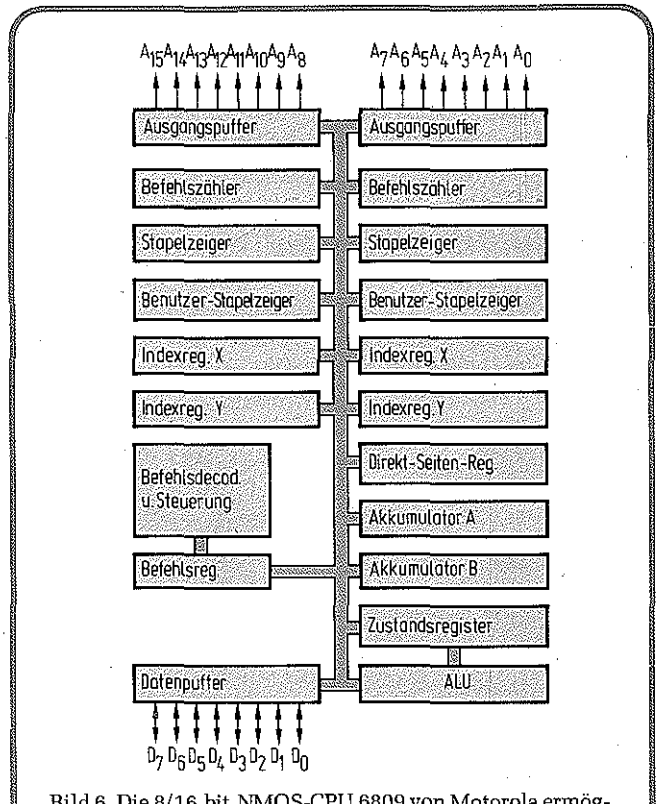


Bild 6. Die 8/16-bit-NMOS-CPU 6809 von Motorola ermöglicht arithmetische 8- und 16-bit-Operationen. Durch Beibehaltung der grundsätzlichen Architektur der CPU 6800 sowie der Software-Kompatibilität unterstützt die gewählte Konzeption vor allem das Arbeiten mit höheren Programmiersprachen

spezielle „Sync"-Instruktion vorgesehen, mit der eine sehr schnelle Synchronisation von Hard- und Software möglich ist.

3.6 Z8000

Während es sich beim Typ 8086 von Intel um den letzten Baustein der Familie 8080 handeln soll (der noch mit der CPU 8080 kompatibel ist), stellt der Prozessor Z8000 von Zilog die erste Komponente einer neuen leistungsfähigen CPU-Familie dar, die nicht mehr zum Z80 kompatibel ist. Der Typ Z8000 wird in zwei Varianten geliefert: Als 40polige Ausführung, die in der Lage ist, 64-KByte-Speicher zu adressieren, sowie als 48polige Version, mit der 128 Speichersegmente zu je 64 KByte adressiert werden können (insgesamt 8 MByte) [11].

Beide Ausführungen verfügen über eine sehr flexible Speicher-Architektur (*Bild 7*) mit 16 Allzweck-Registern, die sich alle als Akkumulator verwenden lassen. Außerdem kann der Programmierer alle Register (außer R-15) als Index- oder Basis-Register bzw. als Speicher-Zeiger für indirekte Adressierung einsetzen.

Der Z8000 ist auch für zwei verschiedene Betriebsarten eingerichtet, „System" und „Normal", die eine Trennung des Betriebssystems von den Anwenderprogrammen bewirken. Dazu verfügt jede Betriebsart u. a. über einen eigenen Stapelspeicher, um z. B. bevorrechtigte Instruktionen von der normalen Programmierung zu trennen.

Bei einer Taktfrequenz von 4 MHz und einer Dauer von 3...70 Taktzyklen je Instruktion, reichen die Ausführungszeiten von 0,75 µs für eine Register-Register-Addition, bis zu 17,5 µs für eine 16 x 16-bit-Multiplikation. Beim Vergleich der Ausführungszeiten dieser CPU mit denen eines bekannten Minicomputers läßt sich zeigen [1], daß der Typ Z8000 alle Bewertungs-Programmtests schneller ausführt, ausgenommen die Multiplikation.

Diese erhöhte Geschwindigkeit ist u. a. auch auf den sehr wirkungsvollen Instruktionssatz zurückzuführen, der viele neue und leistungsfähige Instruktionen umfaßt, wie zum Beispiel: Multiplikation und Division (mit und ohne Vorzeichen), Laden und Speichern mit doppelter Genauigkeit, sowie Instruktionen mit variablem Inkrement/Dekrement. Alle Befehle lassen sich auf Bits oder Bytes, sowie Worte mit einfacher und doppelter Länge anwenden. Vorgesehen sind weiterhin acht Adressierungsarten, von denen sich die fünf wichtigsten (Register, Register indirekt, unmittelbar, direkte Adresse und indizierte Adresse) bei praktisch allen 110 grundlegenden Instruktionen sowie deren 418 Kombinationen anwenden lassen. Der Instruktionssatz enthält zahlreiche Befehle, wie sie bei Minicomputern zu finden sind, beispielsweise String-Manipulationen, dynamische Verschiebung und verbesserte Adressen-Erzeugung. Für Multiprozessor-Betrieb steht sowohl besondere Software („*test bus status*") als auch spezielle Hardware (Ein/Ausgabe-Leitungen) zur Verfügung.

Der N-Kanal-Chip mit den Abmessungen 5,3 x 6,4 mm^2 enthält ungefähr 17 500 Transistoren und ist damit einer der Chips mit der höchsten Packungsdichte. AMD (Advanced Micro Devices) hat von Zilog eine Lizenz als Zweitlieferant erhalten.

3.7 M68000

Im Hinblick auf die Länge der Register, den Umfang des Adressenbereichs, die Adressierungsarten und die Zahl der aktiven Bauelemente übertrifft der Typ M68000 von Motorola alle anderen entsprechenden CPU-Bausteine. Er muß als die bisher bei weitem anspruchsvollste monolithische CPU-Entwicklung angesehen werden. Hätte man den internen 32-bit-Datenbus herausgeführt, wäre der Prozessor 68000 eine vollwertige 32-bit-CPU.

Der 68000 hat eine „orthogonale" Architektur. Darunter ist zu verstehen, daß jeder Befehl auf alle vorhandenen Register unter Verwendung einer der 14 Adressierungsarten bezogen werden kann, wobei sechs Operanden-Typen (Bit, BCD-Nibble, Byte, ASCII-Zeichen, sowie Worte mit einfacher und doppelter Wortlänge) möglich sind.

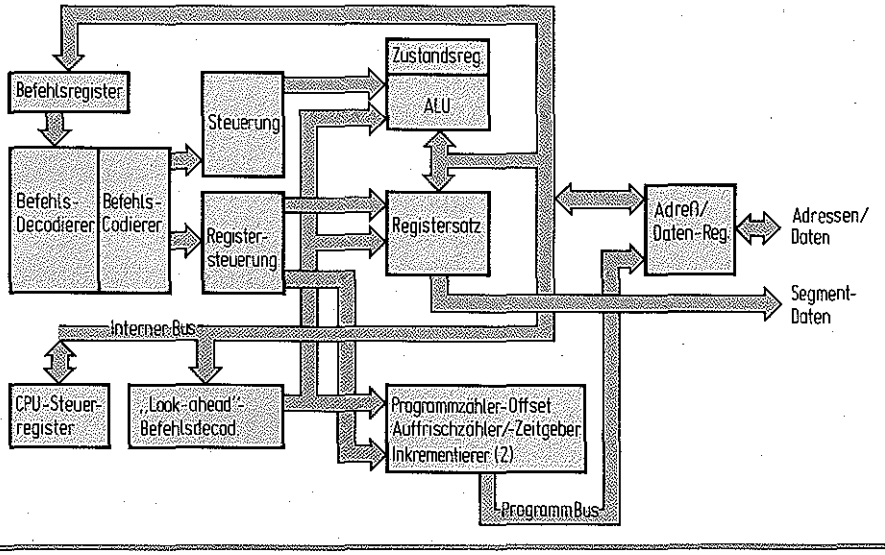


Bild 7. Die 16-bit-NMOS-CPU Z8000 von Zilog hat 16 Allzweckregister, die jeweils als Akkumulatoren oder auch (außer R15) als Index-, Basis- und Zeigerregister eingesetzt werden können. Die Pipeline-Architektur beginnt mit der Befehlsausführung, sobald das Instruktionsregister einen Befehl erhält. Dadurch wird, in Verbindung mit dem Look-Ahead Instruktions-Decoder, der Durchsatz beträchtlich gesteigert

Die sechzehn 32-bit-Register (*Bild 8*) sind in acht Datenregister, sowie acht Register zur Adressen-Modifikation unterteilt. Zusätzlich sind noch ein 16-bit-Status-Register sowie ein Programmzähler mit einer Länge von 24 bit vorgesehen, mit dem die CPU bis zu 16 Millionen Speicherplätze adressieren kann. Die ALU ist ebenfalls für 32 bit ausgelegt, so daß die gleichzeitige Addition von zwei 32-bit-Worten möglich ist.

Die im HMOS-Prozeß hergestellte CPU 68000 erreicht bei einer Taktfrequenz von 8 MHz den mehr als zehnfachen Durchsatz des Types 6800. Ein Vergleich [12] zeigt, daß die Adressierung beim 68000 nahezu doppelt so schnell wie beim Minicomputer PDP-11/45 und allgemein 1,5mal schneller als beim Typ Z8000 erfolgt. Die Register-Adressierung nimmt nur 0,5 µs, indirekte oder unmittelbare Adressierung 1 µs in Anspruch, bei einem Maschinenzyklus von 250 ns Dauer.

Der neue Baustein wurde speziell für die Verwendung höherer Programmiersprachen wie BASIC, FORTRAN, COBOL und PASCAL entwickelt. Die 61 Instruktionen (über 1000 Kombinationen) umfassen Multiplikation/Division mit und ohne Vorzeichen, Laden und Speichern von Worten mit doppelter Länge und bieten Unterstützung bei der positionsunabhängigen Programmierung, dem Speicher-Management und ausführlichen Software-Tests.

Der Chip enthält auf einer Fläche von 7,5 x 7,5 mm^2 insgesamt 75 000 aktive Elemente, die mit einer einzigen Versorgungsspannung von 5 V arbeiten. Er ist in einem 64poligen DIL-Gehäuse untergebracht, da Adreß- und Datenbus getrennt herausgeführt wurden. Muster sollen zur Jahresmitte 1979 erhältlich sein.

3.8 PASCAL Micro-Engine

Alle bisher gebauten digitalen Computer wurden so konzipiert, daß der Hardwareaufwand möglichst klein blieb und sie mit vielen Programmiersprachen arbeiten konnten. Diese Computer-Entwicklungen basieren auf dem Prinzip der sequentiellen Programmausführung, bei dem jeder Programmschritt aus einer Instruktion besteht, die den Betrieb der CPU steuert. Zur Beschleunigung der Programmierung wurden Assembler- und höhere Programmiersprachen entwickelt. Ein in diesen Sprachen geschriebenes Programm muß jedoch zunächst mit Hilfe eines Assemblers, Compilers oder Interpreters in die Maschinensprache übersetzt werden, wodurch der Aufwand an zusätzlich erforderlicher Hilfs-Software stark anwächst, außerdem steigt die Anzahl der möglichen Fehlerquellen.

Im Bemühen, die Programmierung zu vereinfachen, die Hilfs-Software zu reduzieren, den Arbeitsablauf des Computers zu beschleunigen und den Software-Engpaß zu überwinden, brach Western Digital mit dieser alten Tradition und zwar durch die Entwicklung eines Mikroprozessors auf der Basis der Programmiersprache PASCAL [13].

Die „PASCAL micro-engine" (PME) besteht aus vier 40poligen DIL-Bausteinen, die eine spezielle mikroprogrammierte Version des Prozessors MCP-1600 sind. Der Arithmetik-Chip enthält den Mikro-Instruk-

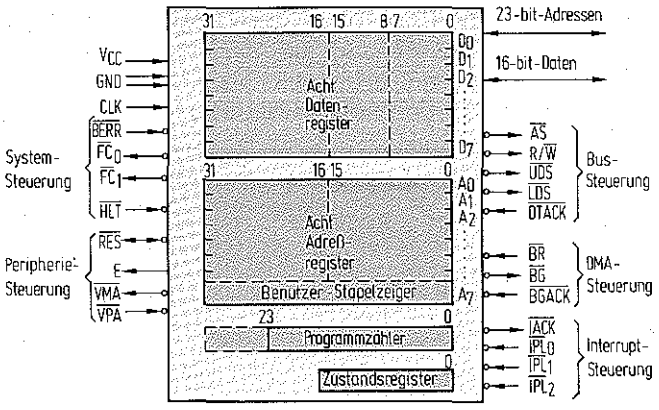


Bild 8. Die 16/32-bit-CPU 68000 von Motorola hat eine orthogonale Architektur. Sie ermöglicht die Anwendung aller 61 Instruktionen auf jedes der sechzehn 32-bit-Register, unter Benutzung jeder der 14 Adressierungsarten. Als Operanden sind Bits, BCD-Nibbles, Worte mit einfacher oder doppelter Länge sowie Wort-Strings möglich

tions-Decoder, eine 16-bit-ALU, sowie einen internen Registersatz. Der Mikro-Sequenzer-Chip umfaßt den Makro-Instruktions-Decodierer, den Mikro-Instruktions-Zähler, sowie die Steuerung für die gesamte Schaltung und die Ein/Ausgabe. Die beiden anderen Chips sind ROMs mit 512 x 22 bit Speicherkapazität; welche die Mikro-Instruktionen enthalten. Die vier Chips bilden einen Stapelspeicher-orientierten 16-bit-Prozessor, der 128 KByte Speicherplatz adressieren kann, vier Interrupt-Ebenen vorsieht und direkten Speicherzugriff steuern kann.

Der PME-Prozessor setzt zunächst das PASCAL-Quellenprogramm in einen zwischengeschalteten P-Code um. Anschließend wird der P-Code direkt vom Host-System ausgeführt, und zwar interpretativ. Der Prozessor kann, wie jeder andere Minicomputer, für beliebige Aufgaben eingesetzt werden, von Anwendungen als Echtzeit-Controller bis hin zum Betrieb in kleinen kaufmännischen Systemen.

Die Chips sind als Satz oder als fertig aufgebautes System mit der Bezeichnung PME erhältlich. Dieses System ist mit 64 KByte RAM, zwei RS-232-Schnittstellen, zwei parallelen Ein/Ausgabe-Toren, sowie einem Floppy-Disk-Controller ausgestattet. Als Software wird das UCSD-PASCAL-Betriebssystem (UCSD = *University of California at San Diego*) mitgeliefert, das einen Compiler, ein Datei-Verarbeitungsprogramm, einen Bildschirm-Editor, ein Fehlersuchprogramm und ein Programmpaket für graphische Darstellungen umfaßt.

Das PME-System kündigt das Ende der von-Neumann-Architektur und den Beginn eines neuen Entwicklungsabschnittes an, in dem die Software die Architektur des Prozessors bestimmt. Eine ausgezeichnete Darstellung der Arbeitsweise eines Computers mit direkter Programmausführung ist [14] zu entnehmen.

4 Mehrchip-CPUs

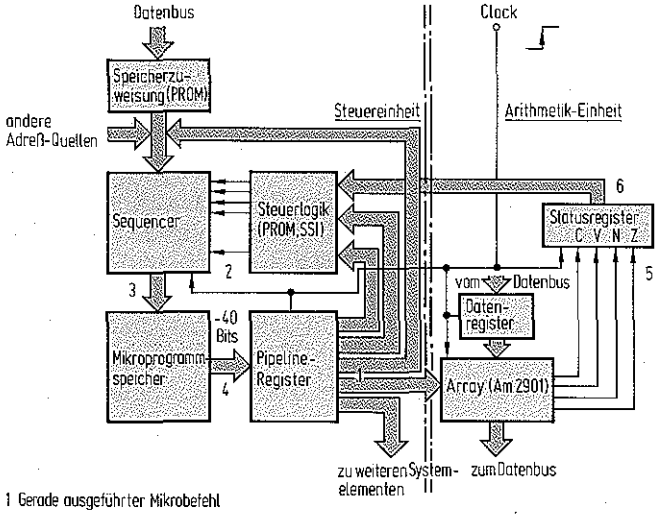
Ein beträchtlicher Teil neuer Mikroprozessorentwicklungen konzentriert sich darauf, CPUs mit der Leistung von Minicomputern zu produzieren. Der

Grund für diese Bemühungen ist nicht darin zu sehen, den gegenwärtigen Minicomputer-Markt an sich zu reißen, sondern vielmehr neue Märkte zu eröffnen, wie zum Beispiel für Heimcomputer und kleine kaufmännische Systeme.

Bis jetzt waren die Eigenschaften der leistungsfähigsten Einchip-CPUs höchstens mit Minicomputern vom unteren Ende der Skala vergleichbar. Im Gegensatz dazu können Mehrchip-CPUs die gleiche oder sogar eine höhere Leistung als Minicomputer erbringen. Eine solche Mehrchip-CPU läßt sich in zwei Bestandteile unterteilen: Die Arithmetik- und Logikeinheit (*ALU*) sowie die Steuereinheit (*CU*). Der Aufbau einer 16-bit-CPU (*Bild 9*) erfordert damit zwischen 30 und 50 SSI-, MSI- und LSI-Schaltungen, je nach den gewünschten Funktionen und der Arbeitsgeschwindigkeit.

4.1 *ALU*

Die ALU, aufgebaut mit vier kaskadierten 4-bit-Slice-Bausteinen, führt mit 16-bit-Operanden die vorgeschriebenen arithmetischen oder logischen Funktionen aus. Der Betrieb der bekannten Bit-Slice-ALU 2901 wird von der Steuereinheit durch neun Signalleitungen festgelegt, von denen die drei ersten die ALU-Quelle (extern, RAM oder Q-Register), die nächsten drei die ALU-Funktion und die drei letzten das Ziel der von der ALU gelieferten Ergebnisse bestimmen (extern, RAM oder Q-Register). Die Register im ALU-Zwischenspeicher werden durch zwei 4-bit-Adressen bestimmt. Die CU erzeugt diese neun ALU-Steuersignale, die 8 Adressen und weitere Signale aus dem 8-bit-Operationscode der Instruktion, den Statussignalen der ALU (*Zero, Sign, Overflow* usw.) sowie den verschiedenen Steuer- und Test-Eingangssignalen.



1 Gerade ausgeführter Mikrobefehl
2 Sequencer-Steuerleitungen, wählen „Quelle" der nächsten Mikrobefehlsadresse
3 Nächste Mikrobefehlsadresse
4 Nächster Mikrobefehl
5 Zustandsbits des gegenwärtigen Mikrobefehls
6 Zustandsbits des vorherigen Mikrobefehls

Bild 9. Die bipolare Bit-Slice-CPU AMD 2900 reduziert die Anzahl der für einen Minicomputer benötigten Komponenten und ermöglicht Software-Kompatibilität durch Emulation des vorhandenen Instruktionssatzes. Die ALU enthält vier Bit-Slice-Bausteine 2901 mit je 4 bit sowie Status- und Datenregister; die Steuereinheit besteht aus dem Mikroprogramm-ROM, der Sequenzer- und Steuerlogik, dem Pipeline-Register und dem Speicher-Zuweisungs-ROM. Die eingekreisten Ziffern geben die Reihenfolge der einzelnen Verarbeitungsschritte an

Die Steuereinheit enthält ein Speicherzuweisungs-ROM, den Sequenzer, das Pipeline-Register und die Steuerlogik. Der Schrittgeber (*Sequencer*) wählt das ROM, das Pipeline-Register oder eine externe Quelle als Eingabe und erzeugt daraus, in Übereinstimmung mit den internen und externen Steuersignalen, die jeweilige Mikroprogramm-Speicheradresse. Der Mikroprogramm-Speicher enthält die Mikroinstruktions-Worte, die gewöhnlich mehr als 40 bit lang sind. Das Pipeline-Register nimmt jede Mikroinstruktion für die Dauer eines Taktzyklus auf, während die CU die nächste Mikroinstruktion erzeugt. Dadurch wird die Arbeitsgeschwindigkeit erhöht.

Die wesentliche Aufgabe der Bit-Slice-Prozessoren ist darin zu sehen, die Anzahl der erforderlichen Komponenten für einen Minicomputer zu reduzieren, ohne daß jedoch Einbußen hinsichtlich der Software-Kompatibilität oder der Geschwindigkeit in Kauf genommen werden müssen. Eine wesentliche Eigenschaft von Bit-Slice-CPUs ist ihre Fähigkeit, vorhandene Minicomputer-Hard- und -Software emulieren zu können. Bit-Slice-Prozessoren sind mikroprogrammierbar. Das bedeutet, daß der Entwickler eine vorhandene Maschine und ihren Makro-Instruktionssatz durch ein entsprechendes Mikroprogramm nachbilden kann.

Die meisten Halbleiter-Hersteller bieten Bit-Slice-CPU-Elemente entweder in Low-Power-Schottky-, ECL-, I^2L- oder CMOS-Technologie an. Eine in jüngster Zeit erstellte Übersicht über Bit-Slice-ALU- und Steuerbausteine ergab [15]:

a) Die Wortlänge der ALU variiert zwischen 2 bit (Intel 3000) und 8 bit (Fairchild ECL-ADIU).
b) Die Anzahl der ALU-Instruktionen liegt zwischen 8 (beim IMP 4, 8, 16) und 24 000 (TI 481).
c) Die maximale Taktfrequenz liegt zwischen 5 und 20 MHz.
d) Die Zahl der ALU-Register variiert zwischen 0 (Motorola 10800) und 20 (IMP 4, 8, 16).
e) Die Anzahl der Befehle des Sequenzers schwankt zwischen 4 (Fairchild 9406) und 100 (IMP 4, 8, 16).
f) Die Tiefe des Schrittgeber-Stapelspeichers liegt zwischen 0 (Intel 3000) und 16 x 4 bit (Fairchild 9406).

4.2 *Wo liegt die Zukunft der Mehrchip-CPUs?*

Auf Grund der äußerst hohen Software-Investitionen vieler Minicomputer-Hersteller ist anzunehmen, daß Bit-Slice-Prozessoren, mit ihrer einzigartigen Fähigkeit, vorhandene Software zu emulieren, noch für einige Zeit auf dem Markt sein werden. Das wird auch durch die immer noch unternommenen Bemühungen zur Entwicklung verbesserter Bausteine demonstriert. Die meisten Hersteller stellen zur Zeit Bit-Slice-Bausteine mit nahezu der doppelten Geschwindigkeit der ursprünglichen Ausführungen vor. Fairchild und Motorola haben gerade jetzt sehr schnelle 8-bit-ALU-Bau-

steine in ECL-Technik angekündigt, einschließlich der zugehörigen Hilfsschaltungen. Diese neue ECL-Familie wurde für Hochleistungs-Zentraleinheiten konzipiert, die in der Lage sind, zwei 64-bit-Worte in 25 ns zu addieren. Die Entwicklung wurde von Sperry Univac unterstützt, jedoch sind die Komponenten in Kürze allgemein erhältlich. Zusätzlich zu diesem 8-bit-ALU-Slice- und Daten-Interface-Baustein (*ADIU*) werden beide Hersteller ein Störungs-Netzwerk (*MFN*), eine programmierbare Interface-Einheit (*PIU*) sowie einen Stapelspeicher mit Zweifachzugriff (*DAS*) anbieten. Die beiden letzten Bausteine sollen zu einem späteren Zeitpunkt in diesem Jahr vorgestellt werden.

5 Einchip-Prozessoren

Der monolithische (Einchip-)Prozessor (MP) [16, 17] vereint CPU, Speicher und Ein/Ausgabe auf einem Chip. Er umfaßt alle Funktionen, die sich normalerweise auf einer Platine finden. Beispielsweise hat der TMS 9940 nahezu die gleichen Eigenschaften wie der Einplatinen-Computer 990/4 von Texas Instruments [11]. Es ist jedoch ersichtlich, daß der Chip nicht ganz so leistungsfähig ist, da er mit nur 128 Byte RAM und 2 KByte ROM ausgestattet ist, während auf der Platine 1 KByte RAM und Fassungen für 4 KByte ROM vorgesehen sind. Diese Situation kann sich aber sehr rasch ändern, da in wenigen Jahren auf einem Chip bis zu 64 KByte RAM oder ROM, getrennt oder gemischt, möglich sein werden, wobei sich der gesamte Speicher mit einem 16-bit-Wort adressieren läßt [18].

Monolithische Prozessoren haben folgende Vorteile:

- Abmessungen, Gewicht und Preis sind so gering wie möglich.
- Es sind keine Verbindungsleitungen erforderlich, außer zu den Ein/Ausgabebausteinen.
- Die Datenübertragung zwischen CPU, Speicher und Ein/Ausgabetor ist optimiert.
- Bus-Treiber und Empfänger werden nicht benötigt.
- Es sind neue Architekturen möglich, da keine Notwendigkeit mehr für ein standardgemäßes Speicher-Interface besteht.
- Die Arbeitsgeschwindigkeit ist erhöht.

4-bit-Prozessoren wie den TMS 1000 und den PPS-4 gibt es bereits seit mehreren Jahren, so daß sie den größten Anteil am Mikrocomputer-Markt haben. Trotzdem besteht inzwischen größeres Interesse an 8-bit-Prozessoren, von denen der erste von Intel und Mostek 1976 vorgestellt wurde. Seitdem ist nicht nur die Zahl der angebotenen Typen erheblich gestiegen, sondern auch ihre Leistungsdaten und sonstigen Eigenschaften haben sich erheblich verbessert. *Bild 10* zeigt die verschiedenen Richtungen, in die sich monolithische Prozessoren weiterentwickelt haben: Größerer Speicherumfang, EPROM-Speicher, leistungsfähigere CPUs, serielle Ein/Ausgabe, programmierbare Ein/Ausgabe, sehr hohe Arbeitsgeschwindigkeit, „On-Chip"-A/D- bzw. D/A-Umsetzer, niedrigere Kosten.

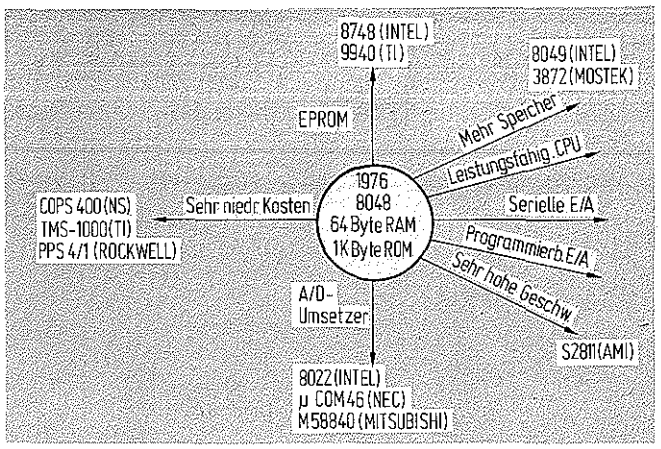


Bild 10. Die Trends bei monolithischen Prozessoren zeigen, in welche Richtungen sie sich während der letzten drei Jahre weiterentwickelt haben, vom preiswerten „COPS" von National bis zum Minicomputer auf einem Chip, dem Typ 9940

Eines Tages sind sicherlich alle Prozessoren „monolithisch", oder etwas genauer gesagt, alle werden Speicher und Ein/Ausgabeschaltungen enthalten. Diese Einchip-Prozessoren dürften bei zukünftigen Anwendungen in Produkten mit hohen Stückzahlen und niedrigen Kosten oder auch in leistungsfähigen Multiprozessor-Systemen die größte Rolle spielen. Zur Zeit bieten acht Halbleiter-Hersteller sieben verschiedene Einchip-Prozessor-Familien an, von denen allein die Familie „48" von Intel acht verschiedene Mitglieder hat. Die vielversprechendsten Entwicklungen in diesem Bereich sind nachfolgend beschrieben.

5.1 *Die Typen 3870/3872 bis 3876*

Mit dem Typ F8 von Fairchild und seiner einzigartigen Architektur, die Speicher und Ein/Ausgabetore auf dem Chip enthält, begann die Entwicklung monolithischer Prozessoren (MP). Der F8 ist allerdings kein wirklich monolithischer Prozessor, da er noch in zwei Chips unterteilt ist. Den Typ 3850 (CPU und RAM) sowie den Typ 3851 (Programmzähler und ROM). Der Prozessor MK 3870 von Mostek dagegen ist die erste tatsächlich monolithische Implementation des F8. Er ist mit ihm hinsichtlich der Software und der Befehlsausführungszeit kompatibel und verfügt über 64 Byte RAM, 2 KByte ROM, 32 programmierbare Ein/Ausgabeleitungen sowie einen 4-MHz-Taktoszillator und ist für Vektorinterrupt eingerichtet. Zur Versorgung ist lediglich eine einzige Spannung von +5 V erforderlich.

Mostek erwartet 1979 einen Absatz von nahezu 3 Millionen Prozessoren des Typs 3870 [19]. Wegen der großen Nachfrage nach diesen Chips hat das Unternehmen eine Reihe neuer Versionen des 3870 entwickelt, die bald vorgestellt werden sollen:

- 3872: Eine Version des 3870 mit doppelter Programmspeicherkapazität – Muster sind bereits erhältlich.
- 3873: Ein 3870 mit A/D- und D/A-Umsetzer auf dem Chip – geplant für Anfang 1979.
- 3874: Eine EPROM-Ausführung des 3870 – geplant für Mitte 1979.

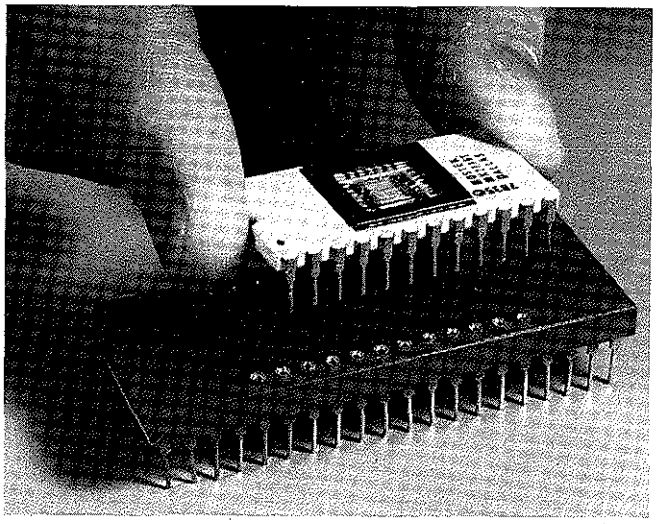


Bild 11. Für Prototypen und Kleinserien ist diese Version des Einchip-Mikrocomputers 3870 vorgesehen, in die Oberseite des 3874 von Mostek können verschiedene Typen EPROMs eingesteckt werden

- 3875: Eine Abwandlung des 3876 mit serieller Ein/Ausgabe – geplant für Mitte 1979.
- 3876: Eine Version des 3870 mit doppelter RAM-Kapazität – geplant für Mitte 1979.

Der Typ 3874 (*Bild 11*) bietet eine sehr zweckmäßige Lösung für die Prototypenentwicklung und Kleinserien-Anwendung. Es handelt sich bei dieser Ausführung um einen 3870 mit einer zusätzlichen Steckfassung, in die das gewünschte EPROM (2708, 2716 oder 2732) im „Huckepack"-Verfahren eingesetzt werden kann. Dies dürfte nicht nur eine sehr preiswerte Lösung sein, sondern sie ermöglicht es auch, mit Hilfe des EPROMs die Anforderungen der jeweiligen Applikationen zu spezifizieren.

5.2 Die Typen 8048/8748/8021/8022/8049

Diese schnell wachsende Familie von Einchip-Prozessoren wurde für einen weiten Bereich von Verarbeitungs-, Controller-, Peripherie-, Ein/Ausgabe- und Interface-Anwendungen konzipiert. Die einzelnen Mitglieder der Familie unterscheiden sich hinsichtlich der Art des ROM-Speichers (elektrisch oder maskenprogrammierbar), der Gehäuseabmessungen, des Instruktionssatzes und der Ein/Ausgabe-Architektur. Alle Bausteine dieser Familie sind durch eine konventionelle aber leistungsfähige Architektur, einen sehr klaren Aufbau, wirkungsvolle Ausnutzung des Programmspeichers sowie einen Instruktionssatz gekennzeichnet, der im wesentlichen aus Ein-Byte-Instruktionen (2,5 µs) besteht.

5.3 Der Typ 8048

Der Typ 8048 [20] ist die Basisausführung. Verglichen mit dem MK 3870 verfügt er nur über die halbe Programmspeicher-Kapazität, hat weniger (nur 27) Ein/Ausgabeleitungen und ist geringfügig langsamer. Dafür sind jedoch sowohl sein Speicher als auch die Anzahl der Ein/Ausgabetore durch eine Reihe spezieller Speicher- und Ein/Ausgabe-Bausteine (8243, 8155, 8355, 8755) erweiterbar. Sie enthalten auf ihrem Chip Adreß-Latch-Schaltungen, die einen direkten Anschluß an den im Multiplex betriebenen Adressen/Daten-Bus des 8048 ermöglichen.

Die meisten bisher erhältlichen Einchip-Prozessoren (3870, 3872, 8048, 8021, 8041, 8049, 6501 und 6600) haben maskenprogrammierbare ROM-Speicher, so daß für jede Anwendung eine spezielle Maske für die Metallisierung entwickelt werden muß. Daraus ergibt sich die Konsequenz, daß sich alle monolithischen Prozessoren mit maskenprogrammierbaren ROMs nur für den Einsatz in Serien mit großen Stückzahlen eignen und außerdem für die Prototypenentwicklung zusätzliche Schaltungen benötigt werden.

5.4 Der Typ 8748

Der Typ 8748 ist mit einem 1 K x 8 bit-Programmspeicher ausgestattet, der elektrisch programmiert und durch Bestrahlung mit UV-Licht wieder gelöscht werden kann, ähnlich wie der EPROM-Speicher 2708 von Intel [17]. Dadurch lassen sich alle beim Typ 8048 erwähnten Probleme hinsichtlich des ROM-Speichers umgehen. Der Typ 8748 ist damit der erste monolithische Prozessor, der folgende Eigenschaften hat:

- Programmiermöglichkeit durch den Anwender, mit Hilfe konventioneller ROM-Programmiergeräte
- einfache Prototypenentwicklung
- Ersatz des Prototyps durch den Typ 8048 für die Massenfertigung möglich
- Einsatzmöglichkeit in kleinen Stückzahlen.

5.5 Der Typ 8021

Der Prozessor 8021, der über nur 21 Ein/Ausgabeleitungen verfügt und mit einem Teil des Instruktions-

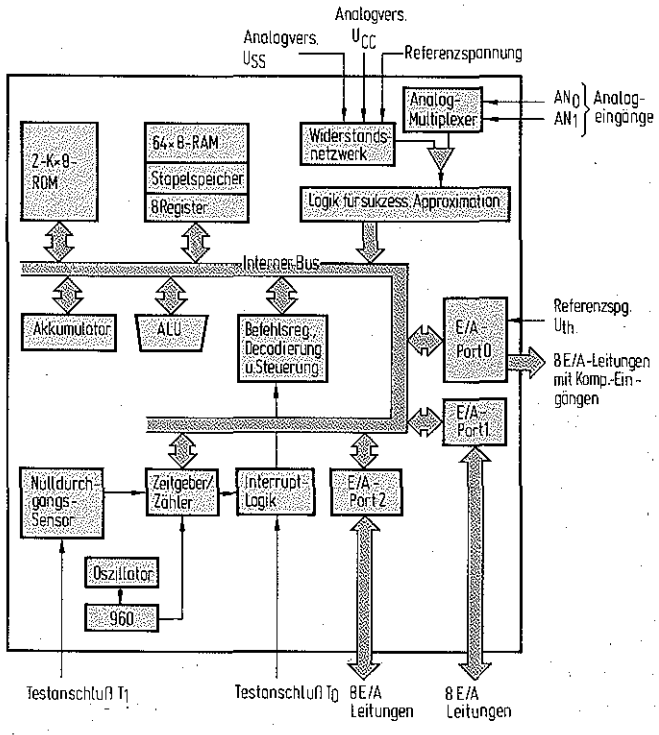


Bild 12. Der Typ 8022 von Intel ist der erste monolithische Prozessor mit einem integrierten 8-bit-A/D-Umsetzer, der eine Genauigkeit von $^1/_2$ LSB, absolute Monotonität und einen analogen 2-Kanal-Multiplexer aufweist

satzes des Typs 8048 ausgestattet ist, wurde vor allem für Niedrigpreis-Anwendungen optimiert. Er arbeitet mit niedrigerer Geschwindigkeit (10 µs Zykluszeit), weist aber andererseits technische Merkmale auf, die der 8048 nicht hat. Beispielsweise ist er mit einem Nulldurchgangs-Detektor und zwei Ein/Ausgabeleitungen ausgestattet, die für Sinkströme bis zu 7 mA ausgelegt sind.

5.6 *Der Typ 6801*

Der Baustein 8049 ist eine erweiterte Version des Typs 8048 mit einer 8-bit-CPU, 2048 Byte ROM (maskenprogrammierbar), 128 Byte RAM-Speicher, einem 8-bit-Zeitgeber/Zähler, einem Taktgeber und 27 Ein/Ausgabeleitungen auf einem Chip. Eine Erweiterung des Speichers und der Ein/Ausgabe ist durch Multiplexbetrieb des Tors 0 (zur Ausgabe von Daten und Adressen) und die Benutzung von vier Leitungen des Tors 2 für die höchstwertigen Adreßbits möglich. Der Typ 8049, der sowohl ein leistungsfähiger Controller als auch ein Prozessor ist, verfügt über umfangreiche Möglichkeiten zur Bit-Manipulation und ist für binäre und BCD-Arithmetik eingerichtet. Sein Instruktionssatz umfaßt über 90 Befehle, von denen 70 % eine Länge von nur einem Byte haben. Keine Instruktion ist länger als 2 Byte. Der Prozessor 8049 wird im 40poligen Gehäuse geliefert und ist pinkompatibel mit dem 8048 und dem 8748.

5.7 *Der Typ 8022*

Monolithische Prozessoren werden häufig in Regelungen eingesetzt, bei denen die Eingangssignale in Form analoger Spannungen vorliegen und die Ausgangssignale analoge Stellglieder steuern müssen. Konventionelle Lösungen verwenden getrennte A/D- und D/A-Umsetzer gemeinsam mit Mikroprozessoren. Dabei sind jedoch die Umsetzer meist entschieden teurer als der eigentliche Prozessor, wodurch die Verwendung monolithischer Prozessoren häufig verhindert wird. Um hier zu einer kostengünstigen Lösung zu gelangen, hat Intel einen monolithischen Prozessor entwickelt, der einen 8-bit-A/D-Umsetzer auf seinem Chip enthält (*Bild 12*). Der Prozessor des Typs 8022 entspricht der im 8021 verwendeten Ausführung (64 Byte RAM, 1 KByte ROM, 70 Instruktionen). Sein 8-bit-A/D-Umsetzer arbeitet mit sukzessiver Approximation und führt eine Umwandlung in 40 µs durch, wobei die erreichte Genauigkeit bei $^1/_2$ LSB liegt. Zusätzlich wurde der Umsetzer mit einem 2-Kanal-Multiplexer und einem Chopper-stabilisierten Komparator ausgestattet. Monotones Verhalten wird garantiert. Der 8022 wird im 40poligen DIL-Gehäuse geliefert, benötigt nur eine einzelne Versorgungsspannung und befindet sich bereits in der Produktionsphase.

Die Kombination präziser Analog- mit digitalen LSI-Schaltungen auf einem Chip bringt zahlreiche Probleme mit sich. Intel hat sie gelöst und damit einen weiteren, bedeutenden Fortschritt in der Entwicklung monolithischer Prozessoren erzielt. Ähnliche Bausteine werden auch von NEC und Mitsubishi erhältlich sein (siehe auch 5.12).

5.8 *Der Typ 6801*

Durch die Kombination der Schaltungen von acht Chips der Familie M 6800 von Motorola (6800, 6810, $1^1/_2$ x 6821, 2 x 6830, $^1/_3$ x 6840, 6850 und 6875), bei gleichzeitiger Beibehaltung der System-Architektur und der Software-Kompatibilität, verringert der Einchip-Prozessor 6801 (*Bild 13*) nicht nur die Hardware-Kosten eines Systems, sondern ermöglicht auch eine beträchtliche Leistungssteigerung [10]. Die CPU-Architektur entspricht im wesentlichen derjenigen des Typs 6800, ihre Arbeitsweise wurde jedoch durch ein zusätzliches Register zur Zwischenspeicherung und einen internen 16-bit-Datenbus verbessert. Die Ausführungszeit vieler Instruktionen konnte verkürzt werden, gleichzeitig wurden zehn neue Instruktionen hinzugefügt: Sechs mit doppelter Genauigkeit (Laden, Speichern, Addieren, Subtrahieren, Links- und Rechtsverschieben), drei zur Manipulation des Index-Registers (X), (X → Stack, Stack → X, B → X), sowie ein Befehl zur 8 x 8-bit-Multiplikation, ohne Berücksichtigung des Vorzeichens. Zusätzlich verfügt der Typ 6801 über einen sehr vielseitigen 16-bit-Zeitgeber mit drei Funktionen, ein serielles Ein/Ausgabe-Tor ähnlich dem ACIA-Baustein (6850), sowie vier Mehrfunktions-Ein/Ausgangs-Tore, deren Betriebsweisen folgendermaßen programmiert werden können:

1) als Allzweck-Ein/Ausgangs-Tore,
2) als getrennte oder im Multiplex-Verfahren betriebene Adreß- und Daten-Busse,
3) als Steuer-Bus,
4) als serieller Ein/Ausgabe-Bus.

Schließlich erhält der 6801 auch noch die erforderlichen Steuerschaltungen für drei verschiedene Arten des Multi-Prozessor-Betriebs: Serielle Datenübertragung, parallele Datenübertragung im Handshake-Verfahren sowie parallele Datenübertragung mit programmierbarer Peripheriesteuerung.

Der Einchip-Prozessor 6801 wird somit durch folgende Eigenschaften charakterisiert:

Er ist

- leistungsfähiger als ein entsprechendes System mit Bausteinen der Familie 6800, dabei aber
- System- und Software-kompatibel, bietet
- Betriebsmöglichkeit mit verschiedenen Ein/Ausgabe-Konfigurationen, wobei in einer Betriebsart vollständige Kompatibilität mit der nicht in Zeit-Multiplex betriebenen Bus-Struktur des Typs 6800 besteht, und kann als
- universelles Prozessor-Element eingesetzt werden, das sich in verschiedenen System-Anordnungen betreiben läßt.

5.9 *Der Typ 9940*

Der Typ Texas Instruments 9940, von dem jetzt die ersten Muster erhältlich sind, wurde als „Minicomputer auf einem Chip" bezeichnet, da er der erste Einchip-Prozessor ist, der 16-bit-Operanden verarbeiten kann und einen leistungsfähigen Instruktionssatz mit Befehlen für Multiplikation und Division aufweist. Seine Geschwindigkeit entspricht derjenigen der CPU

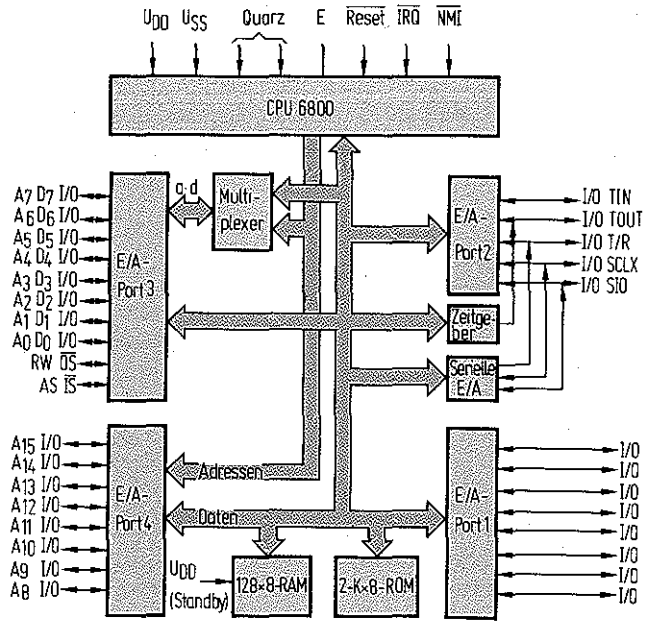


Bild 13. Der monolithische 8-bit-NMOS-Prozessor 6801 von Motorola enthält auf einem Chip das Äquivalent zu acht integrierten Schaltungen der Familie 6800, verfügt über eine programmierbare Ein/Ausgabe-Architektur, arbeitet in verschiedenen Systemanordnungen und ist in Hinsicht auf das System, den Bus und die Software zur Familie 6800 kompatibel

TMS 9900; eine 16-bit-Addition von Arbeits-Register zu Arbeits-Register dauert beispielsweise 4,0 µs.

Der Typ 9940 von Texas Instruments (*Bild 14*) ist das beste Beispiel dafür, welche Fortschritte bei den zur Zeit angebotenen monolithischen Prozessoren noch zu erwarten sind. Mit seiner 16-bit-CPU, 128 Byte RAM und 2 KByte ROM, 32 programmierbaren Ein/Ausgabe-Anschlüssen, der Interruptlogik und dem ebenfalls auf dem Chip befindlichen Takt- und Zeitgeber, übertrifft der mit einer Versorgungsspannung von 5 V arbeitende NMOS-Prozessor die Leistung jedes anderen zur Zeit erhältlichen monolithischen Prozessors. Die zunächst erhältliche Ausführung hat ein maskenprogrammierbares ROM, jedoch sollen bald eine EPROM-Version und eine weitere Ausführung mit einem maskenprogrammierbaren ROM mit 4 KByte Speicherkapazität folgen.

Der Prozessor 9940 hat die fortschrittliche Speicher-Speicher-Architektur des Typs 9900 sowie den gleichen Instruktionssatz. Zusätzlich zu seinen vier 8-bit-Ein/Ausgangstoren ist der 9940 mit einer als *„Communication Register Unit (CRU)"* bezeichneten Schaltung ausgestattet. Sie dient nicht nur zur seriellen Ein/Ausgabe, sondern bietet auch die Möglichkeit zur einfachen Bit-Manipulation (Setzen, Löschen, Testen) und zur Erweiterung der 32 Ein/Ausgabeleitungen auf insgesamt 256 (mit externer Hardware). Die serielle Daten-Übertragung wird jedoch durch die niedrige Geschwindigkeit der software-gesteuerten Verschiebe-Operation beeinträchtigt. Die Eingabe eines 16-bit-Wortes beispielsweise erfordert 56 Taktzyklen zu je 0,4 µs, entsprechend einer Gesamtdauer von 22,5 µs. Das ist sehr langsam, verglichen mit den seriellen Übertragungsraten von 1 Mbit/s, bzw. der bei Parallel-Betrieb erforderlichen 1 µs bei anderen monolithischen Prozessoren.

5.10 Der Typ Z8

Im Bemühen, einen möglichst großen Anteil des Marktes für 8-bit-Mikroprozessoren an sich zu ziehen, hat Zilog den Z8 so konzipiert, daß er entweder als Ein/Ausgabe-intensiver oder als Speicher-intensiver Prozessor arbeiten kann. Der Schlüssel zu dieser ehrgeizigen Zielsetzung liegt in der programmierbaren Ein/Ausgabe-Struktur – ähnlich wie beim Prozessor 6801 von Motorola –, die es ermöglicht, daß einige der allgemeinen Ein/Ausgabeleitungen in einen im Multiplexverfahren betriebenen Adreß/Datenbus umgewandelt werden, sowie in dem getrennten Adreßbereich für die internen und den externen Speicher [23].

Der Z8 (Bild 15) enthält eine 8-bit-ALU, 128 Byte RAM (erweiterbar auf 62 KByte), 2 KByte ROM (erweiterbar auf 64 KByte), zwei Zeitgeber/Zähler, einen asynchronen Empfänger/Sender sowie 32 Ein/Ausgabeleitungen. Die 124 RAM-Speicherplätze und die vier Ein/Ausgabeleitungen sind so organisiert, daß sie als 9 Gruppen zu je 16 Registern behandelt werden können, von denen sich jedes als Akkumulator, Index-Register oder Zeiger einsetzen läßt. Durch diese Lösung erübrigen sich nicht nur spezielle Ein/Ausgabe-Routinen, so daß Ein/Ausgabe-Daten direkt durch jede Instruktion manipuliert werden können, sondern beschleunigt Ein/Ausgabe-Operationen auch beträchtlich. Das Lesen von Daten an einem Tor, oder umgekehrt das Abspeichern, erfolgt um nahezu eine Größenordnung schneller als beim Typ 9940. Darüber hinaus kann der Z8 Register in der gleichen Gruppe mit nur vier Bit adressieren, so daß hierfür lediglich Ein-Byte-Instruktionen erforderlich sind.

Die Geschwindigkeit und Wirtschaftlichkeit des Prozessors Z8 ergibt sich vor allem aus seinem äußerst leistungsfähigen Instruktionssatz, der 47 Basis-Befehle und neun Adressierungsarten zu 129 Instruktionen kombiniert. Die meisten Instruktionen werden in

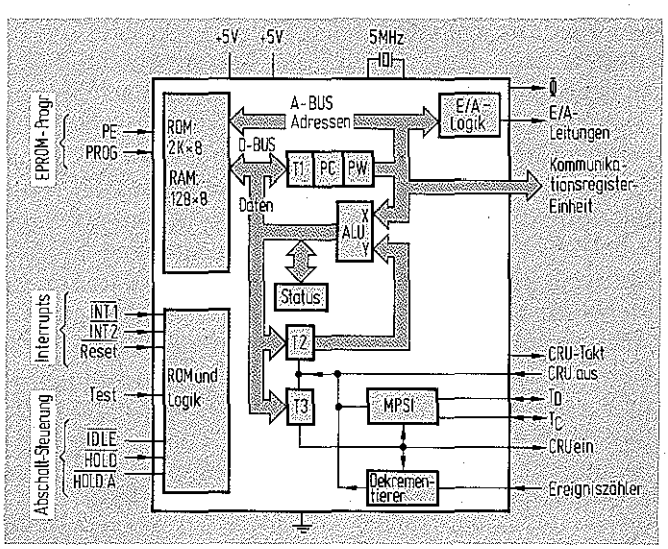


Bild 14. Der monolithische 16-bit-Prozessor TMS-9940 von Texas Instruments arbeitet mit 16-bit-Operanden und enthält 128 Byte RAM sowie 2 KByte ROM, einen Echtzeit-Zähler und vier 8-bit-Ein/Ausgabetore. Er verfügt über den vollständigen Instruktionssatz des Typs 9900, einschließlich der Instruktionen zur Multiplikation und Division und hat eine hochentwickelte Speicher-Speicher-Architektur

6 bis 10 Taktzyklen zu je 250 ns ausgeführt, entsprechend einer Zeit von 1,5...2,5 µs. Viele Instruktionen haben eine Länge von nur einem Byte, woraus eine außerordentlich hohe Packungsdichte bei der Codierung des Programms resultiert. Dadurch wird nicht nur der Speicherbedarf, sondern auch die Ausführungszeit erheblich verringert.

Im Hinblick auf die Bedeutung von Multiprozessor-Systemen bzw. von Netzen mit verteilter Verarbeitungsleistung wurde der Prozessor Z8 sowohl mit der Möglichkeit für Bit-parallele- als auch Bit-serielle Datenübertragung ausgestattet. Paralleler Datenaustausch erfolgt im Handshake-Verfahren über Tor 2. Der serielle Datenverkehr wird von einem UART mit Übertragungsraten von bis zu 62 kbit abgewickelt. ohne daß die CPU dabei mitzuwirken braucht.

Wie es scheint, vereint dieser leistungsfähige, monolithische Prozessor nahezu alle wünschenswerten Eigenschaften anderer Prozessoren (außer Instruktionen für Multiplikation und Division) auf seinem NMOS-Chip mit den Abmessungen 5,6 x 5,6 mm^2. Muster sollen im Frühjahr 1979 erhältlich sein.

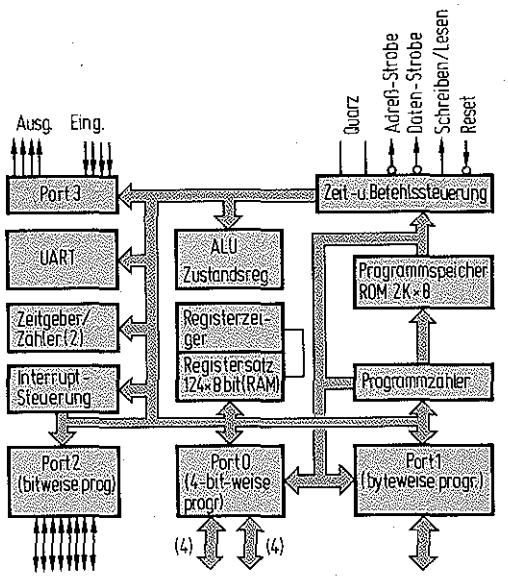


Bild 15. Der monolithische 8-bit-NMOS-Prozessor Z8 von Zilog verfügt über eine sehr flexible Ein/Ausgabe-Architektur, die ihn sowohl für Ein/Ausgabe- als auch speicherintensive Anwendungen geeignet macht. Die Daten in seinen 124 RAM-Speicherplätzen sowie den vier Ein/Ausgabetoren können durch jede der 129 Instruktionen manipuliert werden, wodurch sich große Programm-Dichte und hoher Durchsatz ergeben.

5.11 S 2811

Für Anwendungen, die eine noch höhere Geschwindigkeit erfordern, als mit den bisher beschriebenen monolithischen Prozessoren möglich ist, kann der „Signal-Verarbeitungs-Peripherie-Controller" (SPP) AMI S2811 eingesetzt werden. Er ist schnell genug für Echtzeit-Anwendungen wie schnelle Fourier-Transformation (FFT), digitale Filter oder Sprach-Synthese.

Der in Hochgeschwindigkeits-VMOS-Technologie hergestellte SPP-Baustein kann entweder als peripheres Verarbeitungselement oder als unabhängiger Prozessor eingesetzt werden. Seine einzigartige Architektur umfaßt eine Addier/Subtrahier-Einheit, die entsprechende mathematische Operationen mit 16-bit-Worten in 40 ns ausführt, ein UART, ein RAM mit 256 x 16 bit, ein ROM mit einer Speicherkapazität von 256 x 17 bit, acht Zwischenspeicher mit 16 bit, acht Register mit einer Länge zwischen 5 und 16 bit, einen 5-bit-Schleifenzähler sowie, was am wichtigsten ist, einen Hardware-Multiplizierer der nach dem Booth-Algorithmus arbeitet und eine Laufzeitverzögerung von nur 300 ns aufweist. Durch Verwendung einer Pipeline-Architektur für den Multiplizierer kann der SPP zwei 12-bit-Operanden während eines Taktzyklus laden und das Produkt bereits während des nächsten Taktzyklus ausgeben. Mit Hilfe dieser Schaltung zur Hochgeschwindigkeits-Multiplikation kann der Prozessor einen Algorithmus für ein digitales Filter zweiter Ordnung in nur 2,4 µs ausführen. Das ist um fast zwei Größenordnungen schneller als z. B. bei Verwendung des Prozessors 9940, der für eine Multiplikation 30 µs benötigt.

Zwar enthält der SPP-Baustein über 47 Instruktionen für den Daten-Transfer, für Register-Operationen und bedingte sowie unbedingte Verzweigungen, jedoch ist ihre Leistungsfähigkeit relativ gering, woraus ersichtlich wird, daß es sich beim Typ S2811 mehr um einen Hochgeschwindigkeits-Controller als um einen Prozessor hoher Leistung handelt.

Seine äußerst komplexe Schaltung wurde mit 32 000 aktiven Elementen realisiert, von denen jedes ein Leistungs-Verzögerungszeit-Produkt von nur 0,1 pJ hat. Der Chip mit den Abmessungen 5,5 x 6,6 mm^2 ist in einem 28poligen DIL-Gehäuse untergebracht und weist eine Verlustleistung von nur 500 mW auf. Muster sollen zur Jahresmitte zur Verfügung stehen.

5.12 Monolithischer Analogrechner 2920

Durch Kombination eines Mikrocontrollers mit Sample-and-Hold-Schaltungen, A/D- und D/A-Umsetzern auf einem Chip, auf dem sich auch EPROM, RAM und ALU befinden, entstand bei Intel der „analoge" Mikrorechner (*Bild 16*). Mit ihm ist es möglich, Filter, Multiplizierer, Begrenzer, Gleichrichter und beinahe jedes komplexe analoge Subsystem zu realisieren. Die Zeitbezüge werden durch einen externen Quarz und die Spannungspegel durch die externe Referenz definiert. Die gesamte interne Verarbeitung ist digital. Daraus ergibt sich eine wesentlich höhere Stabilität als ein nur analog aufgebautes System.

Der Analogteil hat vier Eingänge, einen Multiplexer, Sample-and-Hold-Schaltung, D/A-Umsetzer, Komperator, Datenregister, einen Algorithmus für sukzessive Approximation und einen Demultiplexer für acht Ausgänge mit Pufferverstärkern, die jeweils über eine Sample-and-Hold-Schaltung verfügen. Das Datenregister dient als Verbindung zum digitalen Prozessor. Der Digitalteil besteht aus einem Scratchpad-RAM mit

zwei Ports (40 Worte mit 25 bit), einem Binärteiler und einer ALU. Diese kann sowohl das Datenregister als auch jeden Speicherplatz im RAM adressieren. Das im EPROM gespeicherte Programm steuert den Digital- und Analogteil. Es können dort bis zu 192 Befehle mit 24 bit gespeichert werden. Mit einem durchlaufenden Programm werden die analogen Eingangsgrößen abgetastet, digitalisiert, digital verarbeitet nach einer vorgegebenen Funktion und anschließend in die analoge Form zurückgewandelt.

Ein Programm mit der maximalen Länge (192 Befehle) kann mit 13 kHz Wiederholrate ausgeführt werden. Obwohl das Umsetzregister nur eine Länge von 9 bit hat, ist die Auflösung der internen Arithmetik 24 bit. Diese hohe Auflösung ist besonders bei der digitalen Filterung erforderlich.

Die integrierte Schaltung befindet sich in einem 28poligen DIL-Gehäuse mit Fenster für PROM-Löschung.

6 Schlußfolgerungen

Bemerkenswert ist, daß auf dem Mikro-Prozessor/Computer-Markt nicht mit den exotischen, komplexen und hochgezüchteten Bausteinen die großen Umsätze erzielt werden, sondern im Gegenteil mit den kleinen, einfachen und unkomplizierten 4-bit-Einchip-Prozessoren, wie z. B. dem TMS 1000, von dem 1978 nahezu 2 Millionen Stück verkauft wurden. Der Grund dafür dürfte weniger in der Einfachheit dieser Bausteine zu suchen sein, sondern vielmehr in ihrem niedrigen Preis. Daher ist vorauszusehen, daß die Anwender mit großen Stückzahlen zu leistungsfähigeren Bausteinen übergehen werden, sobald diese erst einmal preiswerter geworden sind. Mostek erwartet beispielsweise für den Typ MK3870 im Jahre 1979 einen Absatz von 3 Millionen Stück [19].

Der Trend zielt eindeutig in Richtung auf Prozessoren mit größerer Wortlänge, von vier zu acht und von dort zu 16 bit. Sobald sich die komplexeren Prozessoren zum gleichen Preis herstellen lassen wie momentan die einfachen, wird sich für sie ein großer Markt ergeben. Unzweifelhaft werden mit steigender Komplexität des Chips und der Systeme auch die zugehörige Entwicklungszeit und die anfallenden Kosten steigen. Als Konsequenz muß die Geschwindigkeit abnehmen, mit der diese neuen Produkte entwickelt werden, trotz des größeren Aufwands. Gleichzeitig damit steigen aber auch die Kosten der Fertigungseinrichtungen zur Produktion der Super-Chips in drastischer Weise, von fünfstelligen zu siebenstelligen Beträgen. Als Folge davon werden nur noch sehr finanzstarke Unternehmen in der Lage sein, wirkungsvoll auf dem VLSI-Markt miteinander in Konkurrenz zu treten.

Wirtschaftliche Gesichtspunkte werden die Hersteller monolithischer Prozessoren auch dazu zwingen, Bausteine zu entwickeln, die einen großen Anwendungsbereich abdecken. Nur dadurch lassen sich große Serien realisieren, die eine Amortisation der erforderlichen Investitionen ermöglichen. Das wird automatisch zur Entwicklung universeller Prozessor-Elemente (*UPE*) führen, bei denen es sich um monolithische Prozessoren handelt, die entweder so schnell und leistungsfähig sind, daß sie sich für jede Anwen-

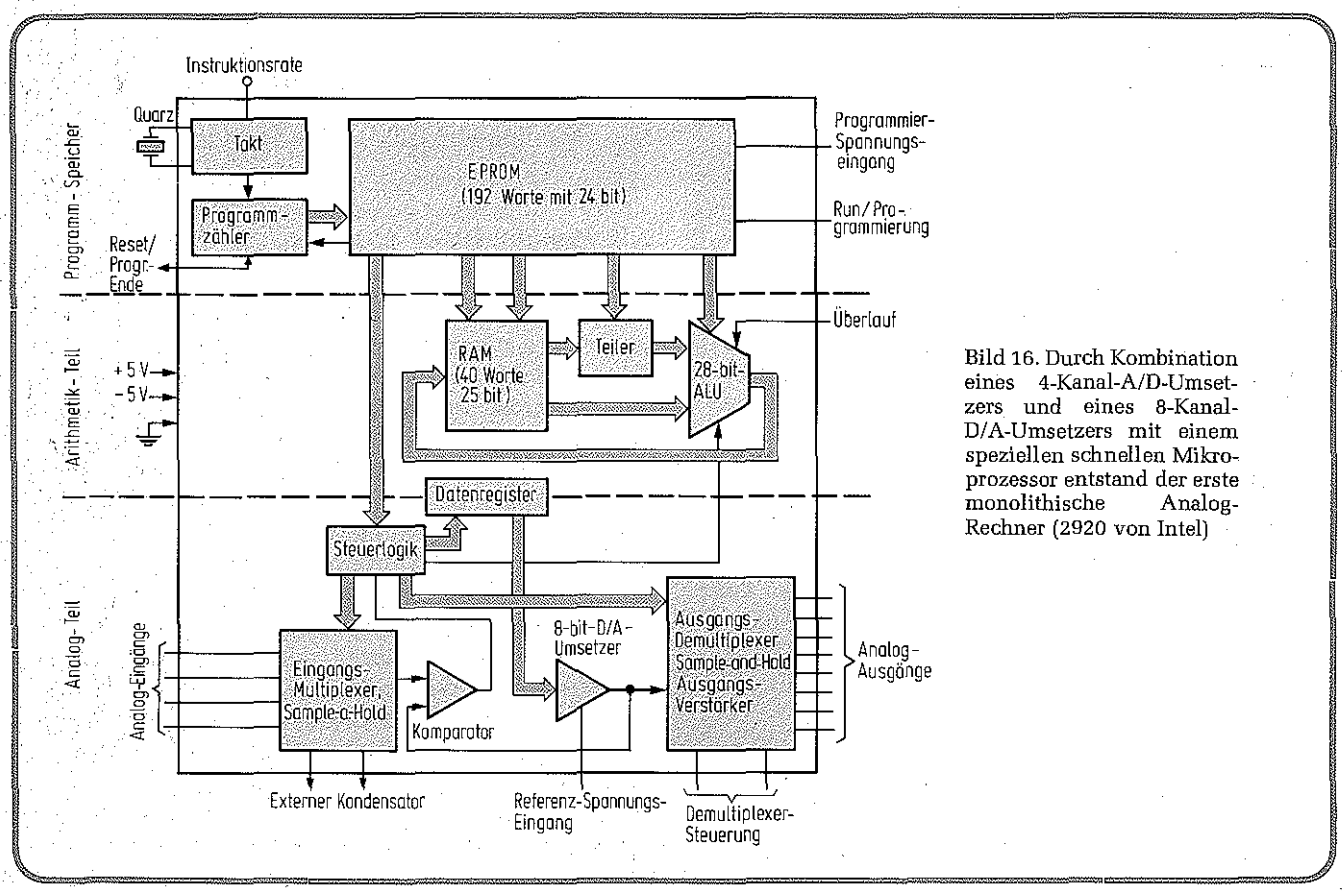


Bild 16. Durch Kombination eines 4-Kanal-A/D-Umsetzers und eines 8-Kanal-D/A-Umsetzers mit einem speziellen schnellen Mikroprozessor entstand der erste monolithische Analog-Rechner (2920 von Intel)

dung eignen, oder aber verschiedenen Applikationen speziell angepaßt werden können, wie es Intel mit der Familie 8048 bereits demonstriert hat. UPEs sollten speziell folgende Eigenschaften haben:

1. Verschiedene Wortlängen zwischen 8 und 32 bit,
2. verschiedene Speichergrößen: 4 K, 16 K, 64 KByte,
3. Mikroprogrammierbarkeit für unterschiedliche Anwendungen,
4. programmierbare Ein/Ausgabe-Architektur (ähnlich wie beim 6801 oder Z8),
5. alle für Multiprozessor-Betrieb erforderlichen Hard- und Software-Eigenschaften,
6. volle Programmierbarkeit durch den Anwender, entweder durch Hardware (über Anschlüsse), Firmware (durch Zerstörung von Sicherungen im Speicher) oder Software (nichtflüchtiger Speicher).

Mit sinkenden Kosten und Abmessungen der Hardware wird auch die Notwendigkeit zur Verringerung des finanziellen und zeitlichen Aufwandes für die Software-Entwicklung immer unumgänglicher. Zwar zeigt sich inzwischen ein deutlicher Trend in Richtung auf höhere Programmiersprachen, aber damit wird nur ein Teil des Problems gelöst. Was wirklich als nächster Schritt erforderlich ist, sind Prozessoren, die eine höhere Programmsprache direkt ausführen können. Western Digital's Lösung, den MCP 1600 mit PASCAL zu mikroprogrammieren, ist bereits ein Schritt in diese Richtung, er geht aber nicht weit genug. Die endgültige Lösung muß mit einem einzigen speziellen Chip erfolgen.

Das Ende der von-Neumann-Architektur ist in Sicht, obwohl noch kein Ersatz dafür vorhanden ist. Stapelspeicher- und Zustands-orientierte Maschinen sowie assoziative Speicher wurden bereits in Erwägung gezogen, jedoch gibt es bisher noch keine Implementationen auf dieser Grundlage.

Beträchtliche Bemühungen werden auch zum Verständnis der Arbeitsweise des menschlichen Gehirns unternommen, bei dem es sich sozusagen um den komplexesten Prozessor handelt, der je gebaut wurde [25]. Es ist deshalb durchaus denkbar, daß man versuchen wird, bei zukünftigen monolithischen Prozessoren die parallele Architektur des Gehirns, seinen Assoziativ-Speicher und seine Redundanz nachzuahmen.

Literatur

[1] *Faggin, F.*: VLSI verändert Computer-Strukturen. ELEKTRONIK 1978, H. 12, S. 57...61.

[2] *Altman, L.*: Advances in Designs and New Processes Yield Surprising Performance. Electronics 1976, H. 7, S. 74...81.

[3] *Chung, T. H. P.*: Electron-Beam Lithography Draws a Finer Line. Electronics 1977, H. 10, S. 89...98.

[4] *Lyman, J.*: X-Ray Lithography Gains Ground. Electronics 1978, H. 15, S. 84...86.

[5] *Cushman, R. H.*: Microprocessor Directory, EDN 1978, 20. November, S. 87...190.

[6] Lofthus, A.: 16-bit Processor Performs Like Minicomputer, Electronics 1976, H. 11, S. 99...105.

[7] *Wilnai, D.*: One-Chip CPU Packs Power of General Purpose Minicomputers, Electronics 1977, H. 13, S. 113...117.

[8] *Forbes, B. E.*: Silicon-on-Sapphire Technology Produces High-speed Single-Chip Processor. Hewlett-Packard Journal, April 1977, S. 2...8.

[9] *Katz, B. J.*: 8086 Microcomputer Bridges the Gap between 8- and 16-bit-Designs. Electronics 1978, H. 4, S. 99...104.

[10] *Wiles, M.*: Compatibility Cures Growing Pains of Microcomputer Family. Electronics 1978, H. 3, S. 95...108.

[11] *Shima, M.*: Two Versions of 16-bit Chip Span Microprocessor Minicomputer Needs. Electronics 1978, H. 26, S. 81...88.

[12] *Le Mair, R.*: Complex System are Simple to Design. Electronic Design 1978, H. 18, S. 100...107.

[13] PASCAL Micro-Engine Preliminary Specifications. Firmenschrift der Western Digital Corporation, 3128 Redhill Ave., Newport Beach, Cal., 92663.

[14] *Chu, Y.*: An LSI Modular Direct-Execution Computer Organization. IEEE Computer Magazine, Juli 1978, S. 69...76.

[15] *Bursky, D.*: Choosing a µP by its Capabilities is a Growing Family Affair. Electronic Design. H. 14, S. 26...38.

[16] *Cushman, R.H.*: Use Forthcoming One-Chip µC's to Achieve Lower Costs. EDN 1978, pp. 75–84. 20. Januar, S. 75...84.

[17] *H. Raphael*: Putting a Microcomputer in a Single Chip. Computer Design 1976, H. 12, S. 59...65.

[18] *Burski, D.*: New Single-Board 16-Bit µCs May Soon Be Challenged by 16-bit Chips. Electronic Design 1978, pp. 52–53. H. 11, S. 52...53.

[19] *Sevin, L.J.*: The Next Generation of Single-Chip Microcomputers. Vortrag beim „General Electric Microprocessor Symposium", Schenectady, N.Y., am 18. Oktober 1978.

[20] *Blume, H.*: Single-chip 8-bit Microcomputer. Electronics 1976, H. 24, S. 99...105.

[21] *Check, W.*: Microcontroller Includes A-D Converter for Lowest Cost Analog Interfacing. Electronics 1978, H. 11, S. 122...127.

[22] *Bryant, J.D.*: 16-bit Microcomputer Seeks Big Byte of Low Cost Controller Tasks. Electronics 1977, H. 13, S. 118...124.

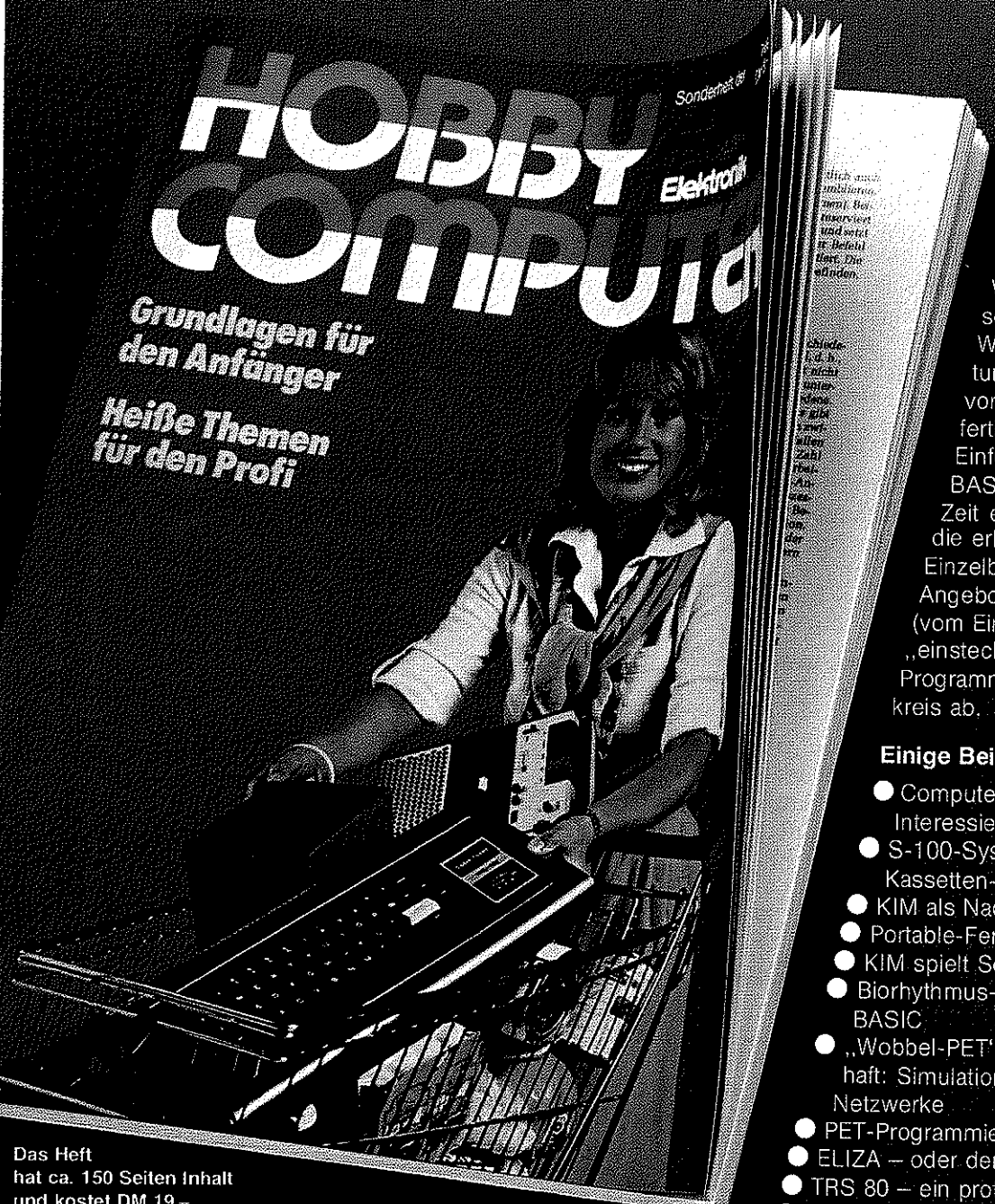
[23] *Peuto, B.L.*: One-Chip Microcomputer Excels in I/O and Memory-Intensive Uses. Electronics 1978, H. 18, S. 128...137.

[24] *Nicholson, W.E.*: the S2811 Signal Processing Peripheral. Beschreibung der Fa. American Microsystems, Inc., 3300 Homestead Avenue, Santa Clara, Cal., 95051. 1978.

[25] *Holtz, K.*: Der selbstlernende und programmierfreie Assoziationscomputer, ELEKTRONIK 1978, H. 14, S. 39...46.

Sonderheft der ELO, FUNKSCHAU und ELEKTRONIK zum aktuellen Thema

HOBBY-COMPUTER



Das Heft hat ca. 150 Seiten Inhalt und kostet DM 19.–

Dieses völlig neue So
bietet dem Anfänger G
lagenbeiträge und sagt
versierten Schaltungste
und Programmierer, welc
lichkeiten das neue Hobb
und was er dafür zu inves
hat. Anhand eines konkret
Beispieles werden Aufbau
Wirkungsweise eines Mikro
sorsystems erklärt.
Wer nicht in die Details der S
tungstechnik eindringen will,
vor allem an das Programmier
fertiger Geräte denkt, der finde
Einführung in die Programmiers
BASIC, die für jeden innerhalb
Zeit erlernbar ist. Ein Querschn
die erhältlichen Systeme – in For
Einzelbeiträgen – zeigt das heutig
Angebot auf dem Hobby-Computer
(vom Einkarten-Computer bis zur
„einsteckfertigen" Anlage). Zahlreic
Programmbeispiele runden den Ther
kreis ab.

Einige Beispiele aus dem Inhalt:

- Computer-Hobby: ein Wegweiser fü
  Interessierte
- S-100-System mit BASIC und schnel
  Kassetten-Interface
- KIM als Nachschlagewerk
- Portable-Fernsehgerät als Monitor
- KIM spielt Schach
- Biorhythmus- und Wochentagsberechnu
  BASIC
- „Wobbel-PET" macht Vierpolrechnung sch
  haft: Simulation des Betriebsverhaltens pa
  Netzwerke
- PET-Programmiertricks
- ELIZA – oder der Computer als Psychoanaly
- TRS 80 – ein professioneller Hobbycomputer
- Lexikon der Fachausdrücke

Wo erhalten Sie dieses Sonderheft:

Bei allen Bahnhofsbuchhandlungen, guten Buchhandlungen, Elektronik-Bauteilehändlern 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 DM 20.50 (DM 19.– plus DM 1.50 Porto) auf unser Postscheckkonto München Nr. 57 58-807 mit Hinweis „Hobby-Computer" zu überweisen. Bitte vergessen Sie nicht, auf dem Zahlungsbeleg in Druckschrift Ihre vollständige Anschrift anzugeben.
Sofort nach Eingang der Zahlung senden wir Ihnen das Heft zu.

Das Heft erhalten Sie in der Schweiz auch
VERLAG THALI AG 6285 Hitzkirch
und in Österreich beim
FACHBUCH-CENTER ERB
Amerlingstraße 1, 1061 Wien

Franzis-Verlag

Karlstr. 37, 8000 München 2, Tel. (0 89)

Philip Hughes

Die Auswahl eines Mikroprozessors zum Aufbau eines speziellen Systems erfordert eine Reihe wichtiger Vorabentscheidungen, die teils die technischen Aspekte, in überwiegendem Maße jedoch finanzielle Gesichtspunkte betreffen.

Mikrocomputer-Projekte richtig geplant

Ein Beispiel soll verdeutlichen, wie sich eine falsche technische Entscheidung auf die Kosten auswirken kann: Firma A plant ein einfaches, relativ langsames Mikrocomputersystem für hohe Stückzahlen, bei dem es erforderlich ist, den Speicherinhalt noch für 2...3 Stunden nach dem Abschalten der Netzspannung zu sichern. Man wählte den Einchip-Mikrocomputer 8048; zwei zusätzliche 256 x 4-bit-CMOS-RAMs wurden für die Datenspeicherung vorgesehen. Da zum Anschluß der RAMs an den 8048 zwei 4-bit-Latch-Speicher erforderlich sind, benötigt das System insgesamt fünf Bausteine.

Im Gegensatz zu dieser Anordnung hätte die Verwendung eines Mikroprozessors ohne Speicher auf dem Chip (z. B. SC/MP) nicht nur die Bausteinzahl auf vier reduziert (SC/MP, ROM, 2 x CMOS-RAM), sondern darüber hinaus auch die Kosten des Netzteiles halbiert.

Eine genauere Betrachtung der verschiedenen Systemaspekte zeigt jedoch, daß beide Lösungen falsch sind, denn aufgrund der niedrigen Arbeitsgeschwindigkeit und des angestrebten hohen Produktionsvolumens wäre ein 4-bit-Prozessor zweckmäßiger gewesen. Die Kosten des Gesamtsystems wären damit um wenigstens 50 % reduziert worden.

Dieses Beispiel verdeutlicht, daß es für den Ingenieur, der Managementfunktionen in Projektierung und Entwicklung wahrnimmt, unumgänglich nötig ist, die Kostenzusammenhänge in einem Mikroprozessorsystem zu durchschauen. Er muß lernen, bei der Planung und Auswahl die richtigen Fragen zu stellen, damit sich schließlich eine optimale Lösung ergibt, wobei nicht unbedingt das neueste Bauteil auf dem Markt für die jeweilige Anwendung das beste sein muß. Einen guten Leitfaden stellt der folgende Fragenkatalog dar:

- Ist der Mikroprozessor schnell genug?
- Was ist die geeignete Wortlänge?
- Mit welchen Startkosten ist zu rechnen?
- Welche Programmiersprache soll gewählt werden?
- Sollen ROMs oder PROMs verwendet werden?
- Ist eine Einchip-Lösung zweckmäßig?
- Ist ein echter Zweitlieferant vorhanden?
- Welche Einflüsse hat der Übergang zu einem anderen Mikroprozessor?

1 Arbeitsgeschwindigkeit

Die Arbeitsgeschwindigkeit des Mikroprozessors ist dann ausreichend, wenn er auf eine Eingabe hin alle erforderlichen Operationen ausführen kann, bevor eine neue Eingabe erfolgt.

2 Wortlänge

Auf dem Markt werden Prozessoren mit 4, 8, 12 und 16 bit Wortlänge angeboten. 4-bit-Prozessoren, wie der TMS 1000 (Texas Instruments) oder die Serie MM 5799 (National Semiconductor), empfehlen sich für Anwendungen in Großserien, wenn Einzelbit-Manipulationen und langsame Dezimal-Arithmetik angewendet werden.

Am populärsten sind Prozessoren mit 8 bit Wortlänge, da sie Einzelbit-Verarbeitung, mathematische Operationen und Byte-Manipulationen ermöglichen. Das Typenspektrum reicht von Ausführungen mit niedriger Arbeitsgeschwindigkeit bis hin zu sehr komplexen Prozessoren. Typische 8-bit-Prozessoren sind:

Billig-Typen:	SC/MP, Signetics 2650
Mittlerer Bereich:	8080A, 6800, F8
Oberer Bereich:	Z 80
Einchip-Ausführungen:	8048, 3870

Der Markt für 16-bit-Prozessoren ist noch jung, drei Typen sind schon einige Zeit auf dem Markt:

GI 1600 – mit 10-bit-Befehlslänge und 16-bit-Datenwort

TI 9900 – der zur Zeit schnellste 16-bit-Prozessor

INS 8900 – ein Billig-Prozessor mit niedriger Arbeitsgeschwindigkeit; er benötigt die wenigsten peripheren Bauelemente zum Aufbau eines Systems.

16-bit-Prozessoren sind besonders in Prozeßsteuerungen und ähnlichen Applikationen, bei denen häufig mathematische Operationen ausgeführt werden müssen, zweckmäßig. In der *Tabelle* sind einige Kriterien aufgeführt, die für die Wahl von 4-bit- oder 16-bit-Typen anstelle eines 8-bit-Prozessors sprechen.

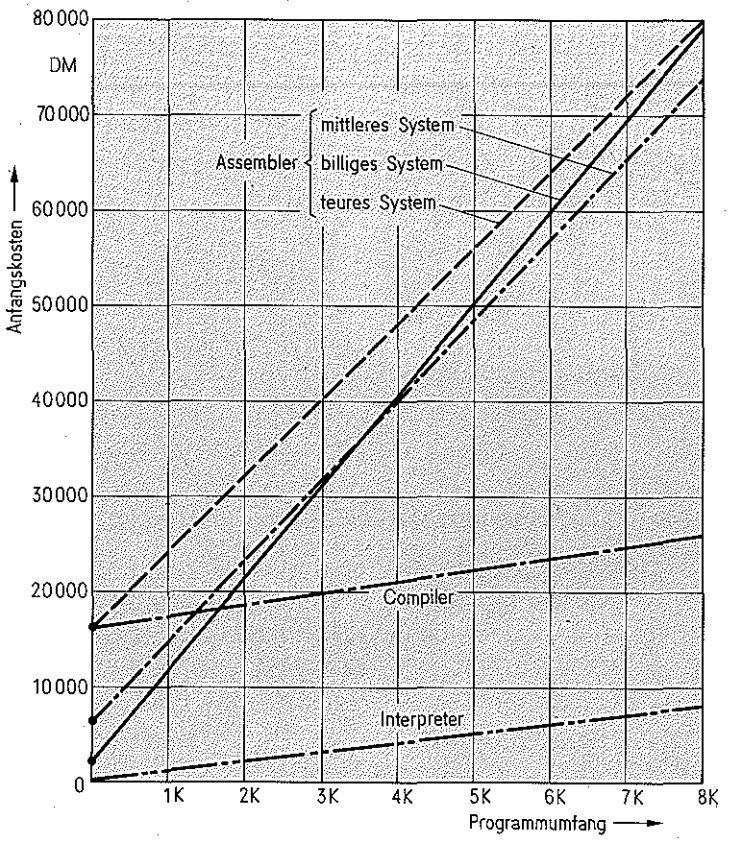


Bild 1. Kostenvergleich für unterschiedlichen Speicherbedarf

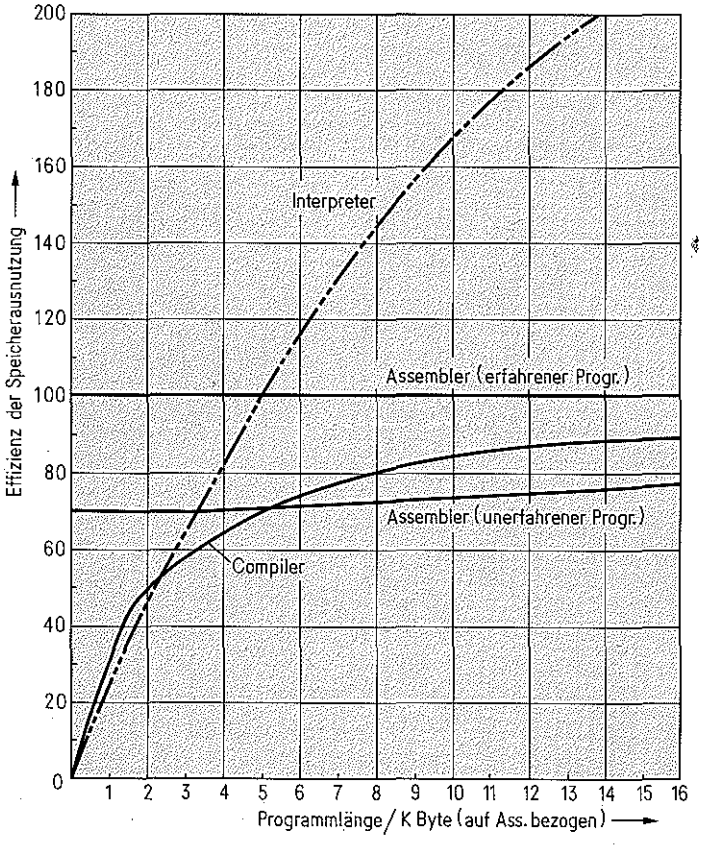


Bild 2. Speicherausnutzung bezogen auf die Programmlänge

3 Startkosten

Je nachdem, ob bereits Erfahrungen mit Mikroprozessoren vorliegen, können die Startkosten unterschiedlich hoch sein. Während gleich zu Anfang eine große Ausgabe für die Anschaffung eines Entwicklungssystems erforderlich wird, sind später, im Laufe der Entwicklung, die Programmierkosten der größte variable Posten, der unter anderem abhängig ist von der Erfahrung des Programmierers. Fängt eine Firma ganz von vorne an, ist für ein Entwicklungssystem mit folgenden Kosten zu rechnen:

Billigsystem mit Assembler, RAM, Teletype : 6000...8000 DM

Entwicklungssystem (mit Teletype) mittlerer Preisklasse: 12 000...18 000 DM

Voll ausgebautes System mit Floppy Disk, Drucker, Streifenleser/Stanzer, Datensichtgerät: 30 000...40 000 DM.

Grundsätzlich ist zu sagen, daß mittlere und größere Systeme eine raschere Programmentwicklung ermöglichen, da sie meist eine Reihe von Hilfsprogrammen enthalten, die Programmtests erlauben und die Fehlersuche erleichtern. Besonders zweckmäßig sind große Entwicklungssysteme, die für verschiedene Mikroprozessoren eingesetzt werden können. Einfache Entwicklungssysteme sollten nur für Programme eingesetzt werden, die bis zu etwa 4 KWorte im Speicher belegen. Darüber hinaus sind große Systeme empfehlenswert (*Bild 1*).

4 Programmiersprachen

Höherentwickelte Programmiersprachen vereinfachen die Programmierung und verkürzen die Programmentwicklungszeit. Eine Befehlszeile in einer interpretativen Programmsprache (z. B. TINY BASIC) ergibt nach der Übersetzung im Durchschnitt etwa 40 Zeilen in Assemblersprache. Während die Programmierung in einer interpretativen Sprache kein großes Entwicklungssystem erfordert, wird bei Verwendung einer Compilersprache häufig schon ein RAM-Bereich von 64 K benötigt. Als Faustregel für die Programmierkosten kann man ansetzen: *Eine komplette getestete und dokumentierte Programmzeile in Compilersprache kostet ca. 5mal, eine Zeile in interpretativer Sprache doppelt soviel wie eine Zeile im Assemblercode.*

Die Wahl der Programmiersprache beeinflußt nicht nur die Startkosten, sondern auch die Entwicklungsdauer

Tabelle der Kriterien, die gegen 8-bit-Prozessoren sprechen

	4 bit	16 bit
hohe Stückzahl	●	
niedrige Geschwindigkeit	●	
hohe Geschwindigkeit		●
zahlreiche mathematische Operationen		●

und hat darüber hinaus Einfluß auf den Speicherumfang.

4.1 Assembler

Eine Zeile im Assemblercode entspricht einer Zeile Maschinensprache. Der Assembler erspart dem Programmierer die Adressenberechnung und stellt die einfachste Sprache dar. Sie ermöglicht 100 %ige Speicherausnutzung, wenn der Programmierer über genügend Erfahrung verfügt.

4.2 Makro-Assembler

Makro-Assembler bieten dem Programmierer die Möglichkeit, Instruktionsketten zu Makrobefehlen zusammenzufassen, die durch ein Symbol aufgerufen werden. Dadurch wird die Codierarbeit reduziert und die Fehlerquote gesenkt. Erfahrene Programmierer können auf diese Weise die Programmierkosten um 5...20 % verringern.

4.3 Compiler

Höher entwickelte Programmiersprachen verwenden Elemente der englischen Sprache sowie mathematische Ausdrücke, die für den Anfänger leichter erlernbar sind als die Symbole der Assemblersprache. Die einzelnen Zeilen des vom Programmierer geschriebenen Quell-Programms werden in Maschinencode übersetzt und mit speziellen Unterprogrammen, z. B. für mathematische Funktionen, zu einem Programmpaket zusammengesetzt. Die Speicherausnutzung erreicht nicht 100 % wie bei der Assemblersprache, da besonders bei kleinen Programmen die speziellen Unterprogramme zu wenig genutzt werden. *Bild 2* zeigt den Zusammenhang.

4.4 Interpretative Sprache

Diese Programmiertechnik verwendet ein im Assemblercode geschriebenes Interpreter-Programm, welches das Quell-Programm liest und direkt ausführt. Die besondere Art der Übersetzung verlangsamt die Programmausführung um den Zeitfaktor 5, was jedoch bei vielen Anwendungen keine Rolle spielt. Abgesehen von diesem Nachteil sind interpretative Sprachen jedoch für kurze Programme in bezug auf die Speicherausnutzung so günstig wie Assemblerprogramme, da 1 Byte in interpretativer Sprache etwa 5 Byte in Assemblersprache entspricht. Interpreter für 8-bit-Mikroprozessoren belegen etwa 4...6 KByte Speicher, so daß ein 1-KByte-Programm in interpretativer Sprache den gleichen Speicherbedarf (einschließlich Interpreter) hat, wie ein äquivalentes 5-K-Assemblerprogramm. Interpreterprogramme sind inzwischen in maskenprogrammierten ROMs erhältlich. Vermutlich werden in einigen Jahren Prozessoren mit Interpretern auf dem Chip üblich sein. Wenn es daher die Geschwindigkeitsanforderungen erlauben, sollten zur Reduzierung der Startkosten für alle Programme mit mehr als 5 KByte Interpreter verwendet werden, ansonsten Compilersprachen.

5 ROM oder PROM?

Liegen Programmlänge und Produktionsstückzahl fest, sind geeignete Speicher-Bausteine auszuwählen (*Bild 3*). ROMs mit Maskenprogrammierung sind bereits ab Stückzahlen um 100 Einheiten preiswerter als PROMs mit gleicher Speichergröße, wobei bemerkt werden muß, daß die Kostenrelation Pfg./bit um so günstiger ist, je höher die Speicherkapazität eines einzelnen ROMs ist (*Bild 4*).

Ein Nachteil der Maskenprogrammierung ist jedoch darin zu sehen, daß sie nur vom Hersteller ausgeführt werden kann und somit für den Anwender, nach Erstellung des Programms, eine Wartezeit von 8...10 Wochen vergeht, bevor er die programmierten ROMs erhält. Wer es sehr eilig hat, sollte daher PROMs verwenden, außerdem besteht dann noch die Möglichkeit, innerhalb einer Serie eine eventuelle Programmkorrektur vorzunehmen.

6 Einchip-Mikrocomputer oder nicht?

Für Anwendungen in großen Stückzahlen (50 000...100 000/Jahr) bieten Einchip-Mikrocomputer eine sehr preiswerte Alternative. Die Kosten eines mit diskreten Komponenten aufgebauten Systems können bis zu 300 % darüber liegen.

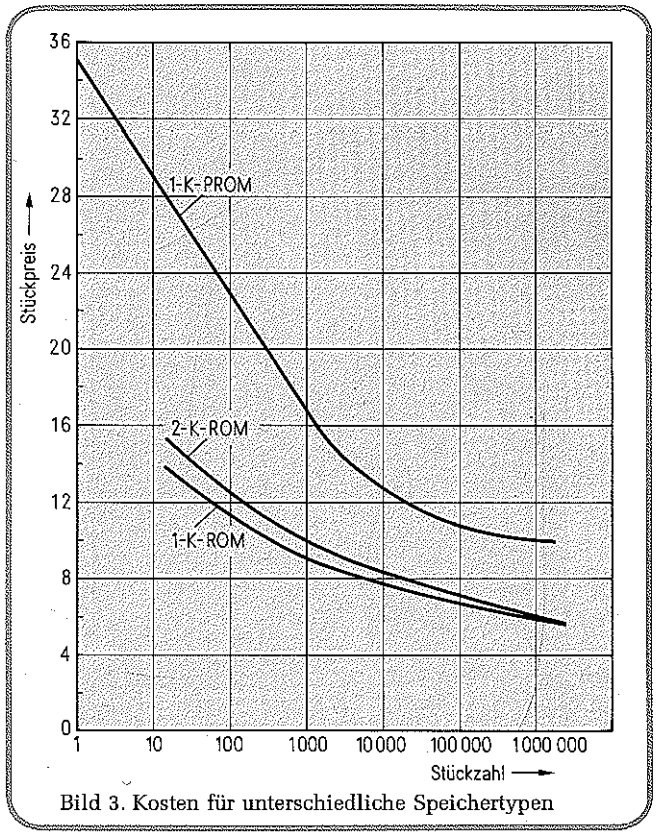


Bild 3. Kosten für unterschiedliche Speichertypen

Eine Einchip-Lösung ist allerdings nur dann interessant, wenn der Mikrocomputer direkt in der vom Hersteller gelieferten Form verwendet werden kann, d. h. im wesentlichen ohne zusätzliche Komponenten. Ist z. B. Speichererweiterung erforderlich, werden neben dem Speicher noch zusätzliche Bausteine benötigt; außerdem werden E/A-Leitungen belegt, die dann nicht mehr anderweitig benutzt werden können. Sollen außerdem Relais, Anzeigeelemente usw. angeschlossen werden, sind mit Sicherheit zusätzliche Treiber notwendig, welche die Einchip-Lösung finanziell eventuell unattraktiv machen. Ebenso ist es schwierig, ein Einchip-System auf höhere Computerleistung auszubauen, während dieses Vorhaben in einem busorientierten System keine großen Probleme bereitet.

Um rasch mit mittleren Stückzahlen auf den Markt zu kommen, ist ein Mikrocomputersystem mit einem „diskreten" Mikroprozessor günstiger (*Bild 5*).

7 Zweitlieferanten

Bei der Auswahl eines Mikroprozessors spielt das Vorhandensein von Zweitlieferanten eine wichtige Rolle. Entscheidend ist für den Anwender jedoch, daß der Zweitlieferant mit dem Ersthersteller auch wirklich einen „Second-Source-Vertrag" hat, der ihm die entsprechende Unterstützung und Hilfestellung bietet. Neben echten Zweitlieferanten gibt es auch eine Reihe von Herstellern, die eine eigene Version des Produktes baut. Es versteht sich von selbst, daß dieses Produkt mindestens so gut sein muß wie das Vorbild, um konkurrenzfähig zu sein.

Die Frage, ob es sich lohnt, einen neuen, besseren Mikroprozessor anstelle eines bereits in großen Stückzahlen eingesetzten zu verwenden, muß von Fall zu Fall sehr sorgfältig geprüft werden. Die Kosten, die durch das erforderliche Bauteilelager, Einarbeitung des technischen Personals und neue Dokumentation verursacht werden, können den erhofften Gewinn bei weitem übertreffen.

8 Gesamtkosten

Auf Grund der vorausgegangenen Überlegungen läßt sich eine einfache Formel zur Ermittlung der auf eine Einheit bezogenen Gesamtkosten aufstellen:

$$K_S = \frac{Stk}{V} + \frac{Pr}{A} \cdot R_k + S_k$$

Die einzelnen Größen haben folgende Bedeutung:

K_s = Kosten eines Systems
Stk = Startkosten (Entwicklungssystem usw.)
V = Produktionsstückzahl
Pr = Programmlänge in Bytes
A = Speicherausnutzung durch das Programm
R_k = Kosten für Festwertspeicher
S_k = Kosten des restlichen Systems

Literatur

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

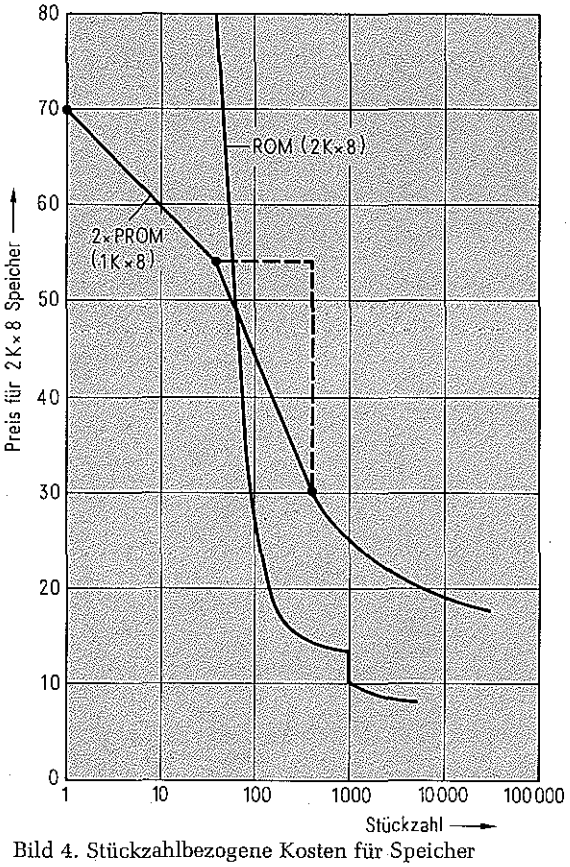


Bild 4. Stückzahlbezogene Kosten für Speicher

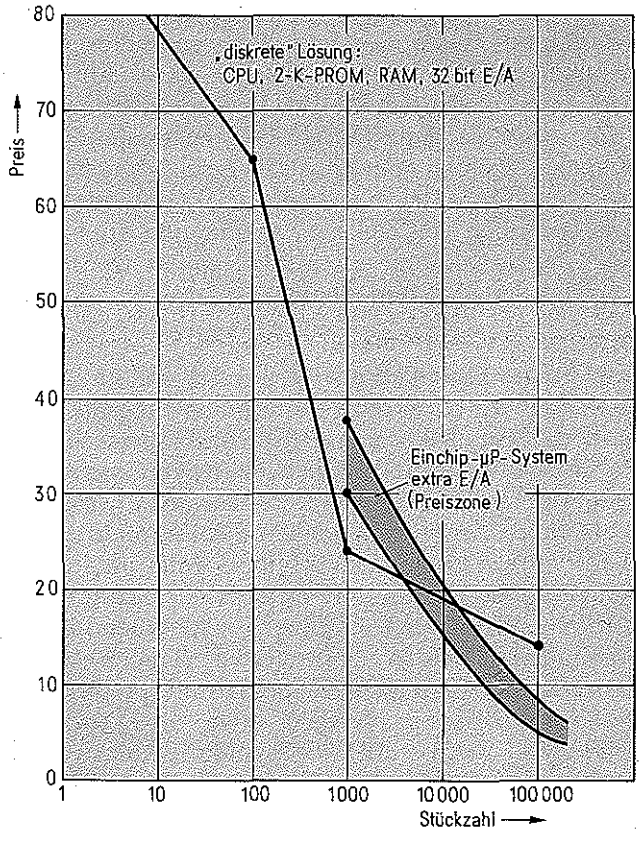


Bild 5. Vergleich: Einchip-Mikrocomputer und „diskrete" Lösung mit CPU, Speicher und E/A

Dipl.-Ing.
Ludwig Dorn

Der HMOS-Prozeß *(High Performance MOS)* ist durch Verkleinerung der Transistor-Strukturen aus dem NMOS-Prozeß entstanden [2]. Er wurde erstmals bei RAM-Bausteinen angewendet, die seit längerer Zeit gefertigt werden. Seit kurzem wird in dieser Technik der 16-bit-Mikroprozessor 8086 hergestellt, der Taktfrequenzen bis zu 8 MHz erreicht. Er wird im folgenden Beitrag vorgestellt.

HMOS-Prozeß führt zu leistungsfähigem 16-bit-Mikroprozessor

Dieser Prozessor hat zwar eine gänzlich neue Struktur, ist aber trotzdem voll abwärtskompatibel zu der schon länger bestehenden 8080/85-Familie, dem am weitesten verbreiteten Mikroprozessorsystem. Die Kompatibilität gilt sowohl für die Software als auch für den Bus. Alle existierenden Peripherie-Bausteine können z. B. verwendet werden – auch zum Aufbau eines Einkarten-Computers, der zu den anderen Multibuskarten der 8080/85-Familie kompatibel ist.

Der Prozessor wurde mit einer kompletten Familie an Bausteinen auf den Markt gebracht, z. B. Taktgenerator mit Treiber, Bus-Controller usw. Wie sich aus *Bild 1* entnehmen läßt, ist es möglich, mit nur vier Chips dieser neuen Familie sowie Standard-Speichern und -Peripherie-Bauelementen sehr einfach ein Minimal-System aufzubauen.

1 Zentraleinheit

Der 8086 benötigt nur eine Versorgungsspannung von 5 V. Das kommt erst richtig zum Tragen, seit genügend RAM-, ROM-, PROM- und EPROM-Speicherbauelemente zur Verfügung stehen, die ebenfalls nur eine Versorgungsspannung benötigen. Die Unterbrin-

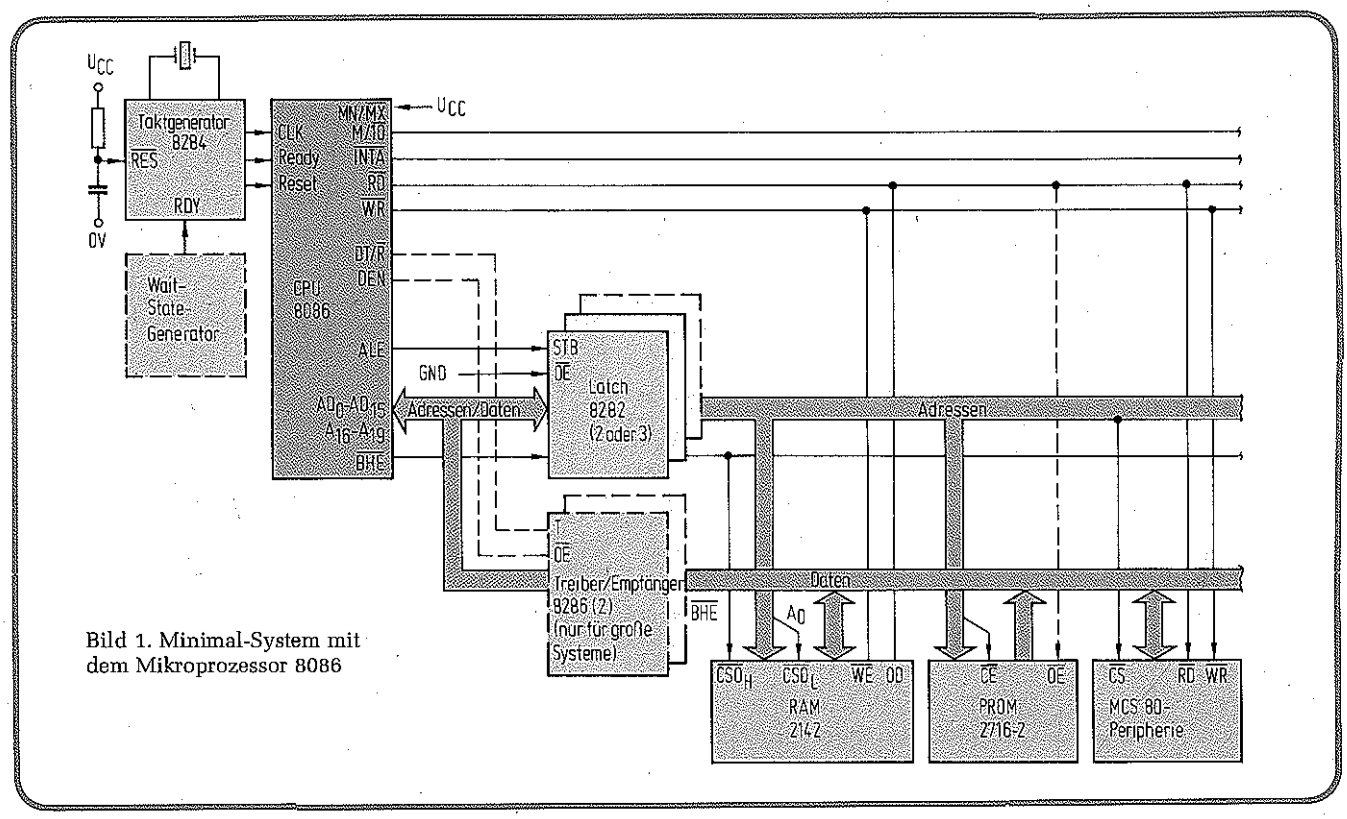


Bild 1. Minimal-System mit dem Mikroprozessor 8086

Tabelle 1. Datenbussteuerung mit den Signalen Bus High Enable und Adreßbit 0

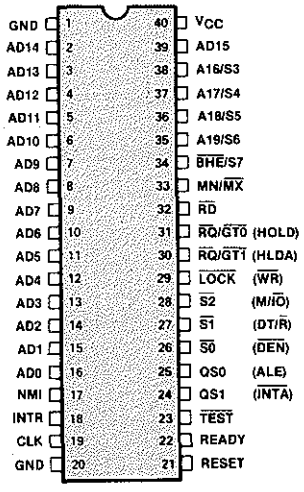
A0 (Adreßbit 0)	BHE (Bus High Enable)	
0	0	Wortübertragung
0	1	Byteübertragung – gerade Adresse (unteres Byte)
1	0	Byteübertragung – ungerade Adresse (oberes Byte)
1	1	keine Übertragung

gung in einem kostensparenden 40poligen Gehäuse wurde durch die Belegung der meisten Anschlüsse mit mehreren Funktionen erreicht (*Bild 2*). So werden z. B. über den Datenbus auch die Adressen übertragen, während die Funktion einiger Steuer- bzw. Statusleitungen über einen einzigen Eingang umgeschaltet werden können (*Minimum/Maximum-Mode*). Im Minimum-Mode erzeugt der Prozessor die Bussteuersignale selbst (Aufbau eines einfachen Systems); im Maximum-Mode dagegen erzeugt sie (und einige andere Steuersignale) der Bus-Controller 8288. Die hierbei freigewordenen Anschlüsse der Zentraleinheit unterstützen in diesem Fall Multiprozessoranwendungen. Dies sind z. B. zwei unabhängige bidirektionale „Request/Grant"-Leitungen, über die der Prozessor in einen Wartezustand gebracht werden kann, um direkten Speicherzugriff zu ermöglichen bzw. einen anderen Prozessor auf denselben Bus zugreifen zu lassen. Die entsprechenden Quittungssignale erzeugt der 8086 auf den gleichen Leitungen.

Ein anderer Ausgang, das Lock-Signal, wird durch einen Befehls-Vorsatz von 1 Byte aktiviert. Dies gibt die Möglichkeit, während der Datenübertragungsoperationen des 8086 den Zugriff anderer Prozessoren auf den gleichen Speicher zu verhindern. Die Anwendung dieses Signales wird durch folgendes Beispiel verdeutlicht: Zwei Prozessoren arbeiten mit dem gleichen Pufferspeicher. Ein Flag (PFLAG) am Anfang des Speichers gibt Auskunft, ob der Puffer gerade benutzt wird. Die Befehlsfolge in *Bild 3* gestattet gleichzeitig Abfrage und Setzen dieses Flags, insbesondere mit Hilfe des Austausch-Befehls (*Exchange*). Der Austausch-Befehl bewirkt zunächst eine Datenübertragung aus dem Speicher in Richtung Zentraleinheit – Auslesen des Flag – und anschließend eine zweite Datenübertragung von der Zentraleinheit zu dem Speicher – Setzen des Flags. War der Puffer belegt, so wurde durch diese Operation das Flag nicht verändert, andernfalls wurde gleichzeitig mit dem Auslesen die Belegt-Information in das Flag übertragen. Ein Konflikt tritt dann auf, wenn ein zweiter Prozessor zwischen den beiden Datenübertragungen des Austausch-Befehls den gleichen Speicherplatz auslesen kann und ebenfalls die Nicht-Belegt-Information erhält. Das kann leicht durch Nutzung des Lock-Signals beim 8086 verhindert werden.

Um den Speicher auch byteweise adressieren und ansprechen zu können, wurde zu den 20 Adreßbits ein weiteres Bussteuersignal (*Bus High Enable, BHE*) hinzugefügt. Aus *Tabelle 1* ist das Zusammenwirken dieses Signales mit dem Adreßbit Null zu entnehmen. Interessant ist die Tatsache, daß bei diesem Prozessor ein Wort im Speicher nicht unbedingt an einer geraden Adresse beginnen muß. Ist der Anfang eines Wortes an einer ungeraden Adresse, so führt der Prozessor automatisch zwei Byte-Zugriffe auf den Speicher durch. Dies bietet bei zeitunkritischen Programmteilen die Möglichkeit, den Speicher optimal auszunutzen.

In *Bild 4* ist die interne Struktur des Prozessors dargestellt. Es fällt sofort die Aufteilung in die Ausführungseinheit (*Execution Unit*) und die Bus-Interface-Einheit (*Bus Interface Unit*) auf. Diese beiden Einheiten tauschen zwar Daten direkt miteinander aus, arbeiten aber aufgrund ihrer unterschiedlichen Aufgaben zum größten Teil relativ unabhängig. Zur Ausfüh-



◀ Bild 2. Anschlußbelegung der CPU: Die Stifte 24...31 haben im Minimum- bzw. Maximum-Mode unterschiedliche Bedeutung (Originalbild Intel)

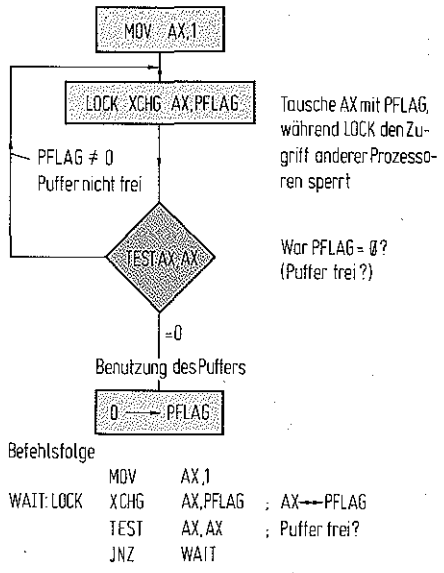


Bild 3. ▶ Abfrage eines Flags im Speicher mit dem Lock-Befehl

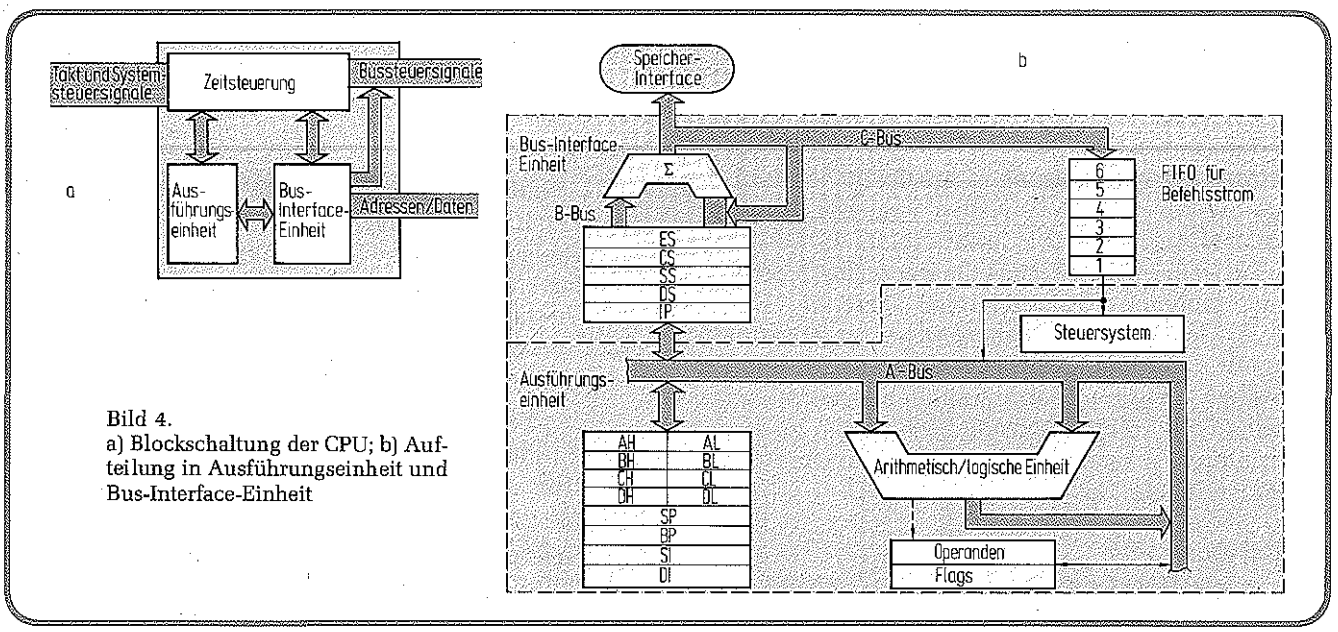


Bild 4.
a) Blockschaltung der CPU; b) Aufteilung in Ausführungseinheit und Bus-Interface-Einheit

rungseinheit gehören der allgemeine Registersatz, die arithmetisch/logische Einheit, ein Operandenregister, die Flags und die Steuereinheit. Dort werden die von der Bus-Interface-Einheit abgeholten Befehle interpretiert, die Daten, ebenfalls von der Bus-Interface-Einheit kommend, verarbeitet und dieser wieder zur Verfügung gestellt. Die Bus-Interface-Einheit umfaßt einen speziellen Adreßregistersatz, das Adreßaddierwerk und ein 6 Byte langes FIFO-Register für die Befehle. Als besondere Aufgabe dieser Funktionseinheit ist die Optimierung der Prozessorgeschwindigkeit anzusehen. Dies geschieht zum ersten, indem die Bus-Interface-Einheit die Befehle im voraus aus dem Speicher abholt und im FIFO ablegt. Die Ausführungseinheit kann die Befehle abholen, ohne den Speicherzugriff abzuwarten. Zum zweiten führt sie die Operandenübertragung zwischen dem Speicher bzw. der Peripherie und der Ausführungseinheit durch, generiert hierfür die notwendigen physikalischen Adressen und erzeugt optimale Bussteuersignale, so daß trotz hoher Prozessorgeschwindigkeiten relativ langsame Speicher-Bauelemente eingesetzt werden können.

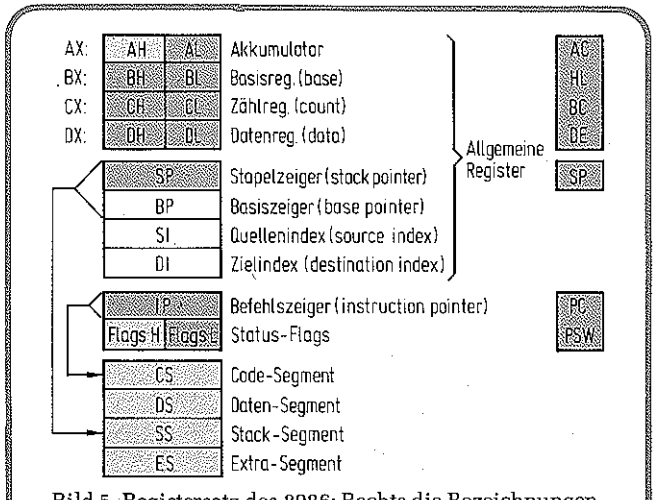


Bild 5. Registersatz des 8086: Rechts die Bezeichnungen der entsprechenden 8080-Register

Der allgemeine Registersatz (*Bild 5*) besteht aus acht 16-bit-Registern. Vier dieser Register können auch byteweise adressiert werden. Damit lassen sich die meisten Datenoperationen symmetrisch durchführen. Weiterhin wird der Registersatz durch den Befehlszeiger, die Status-Flags und vier Segmentregister vervollständigt. Durch die Segmentregister kann der 8086 einen Speicherbereich von 1 MByte direkt adressieren. Die hierfür benötigten 20 Adreßbits werden durch Addition der Offset-Adresse zu dem Inhalt eines der vier Segmentregister, der vorher um 4 bit nach links verschoben wurde, gebildet (*Bild 6a*). Der Offset kann aus einem Registerinhalt (z. B. Befehlszeiger), der Summe verschiedener Registerinhalte oder einer dem Opcode eines Befehls folgenden Konstanten bestehen. Die vier Segmentregister definieren zu einem Zeitpunkt somit vier unabhängige Segmente zu jeweils 64 KByte Länge innerhalb des 1-MByte-Speicherraumes. Die Segmente können in Inkrementen von 16 Byte an jeder beliebigen Stelle beginnen – Überlappen ist erlaubt (*Bild 6b*). Das Code-Segment ist für den Programmteil vorgesehen, das Stack-Segment für den Stack. Die beiden restlichen, das Daten-Segment und das Extra-Segment, erlauben einen Datenverkehr innerhalb zweier unabhängiger Speicherbereiche und damit gleichzeitig innerhalb des gesamten Speicherbereiches.

Der 8086 bietet ein Interrupt-System mit 256 Ebenen. Interrupts können sowohl hardwaremäßig als auch softwaremäßig eingeleitet werden. Neben den maskierbaren Interrupts steht auch ein nicht maskierbarer zur Verfügung. Intern sind einige Interrupt-Ebenen fest vorgegeben, z. B. Division durch Null, Einzelschritt usw. Letzterer gestattet auf sehr einfache Art und ohne zusätzliche Hardware, die Software in Einzelschritten zu testen, auch wenn die Programme in einem ROM abgespeichert sind.

2 Zusatz-Elemente

2.1 Taktgenerator

Wie aus Bild 1 ersichtlich, beinhaltet der Taktgenerator 8284 neben dem Taktoszillator für den 8086 noch Synchronisationsschaltungen für das Reset- und das Ready-Signal, das für langsame Speicher oder Peripherie benötigt wird. Die Taktfrequenz für den Teiler kann entweder vom internen Quarzoszillator oder von einem externen Taktgenerator erzeugt werden (*Bild 7*). Dies gestattet insbesondere im Zusammenhang mit dem Synchronisationseingang für den Prozessortakt, mehrere Zentraleinheiten innerhalb eines Systems vollkommen taktsynchron laufen zu lassen. Ein weiterer Ausgang bietet den Takt für die Peripherie-Bausteine an.

2.2 Bus-Controller

In der Maximal-Konfiguration (*Bild 8*) erzeugt der Bus-Controller 8288 die Bussteuersignale. Diese Ausgänge sind Multibus-kompatibel und haben eine hohe Treiberleistung (32 mA bei 300 pF). Die für manche Speicher- oder Peripherie-Bausteine benötigten vorzeitigen Schreibimpulse (aktiv bevor die Daten stabil sind) erzeugt diese Einheit ebenfalls. Während die Zentraleinheit in der Minimal-Konfiguration die Steuerung der Adreß-Latches und der Bus-Treiber/Empfänger selbst übernimmt, wird hier die Steuerung vom 8288 übernommen. Dadurch werden, wie schon erwähnt, einige Anschlüsse der Zentraleinheit für weitere Statusinformationen und verbesserte Multiprozessoranwendungen frei.

2.3 Weitere Peripherie-Bausteine

Die 8fach-Tristate-Latches 8282/8283 und die 8fach-Treiber/Empfänger 8286/8287 sind in einem platzsparenden 20poligen Gehäuse untergebracht und haben dieselbe Stiftbelegung. Ihre Treiberleistung beträgt wie beim Bus-Controller 32 mA bei 300 pF im gesamten Temperaturbereich. Sowohl invertierende als auch nicht invertierende Ausführungen stehen zur Verfügung.

Der Interrupt-Controller 8259A, Nachfolger des 8259 für das MCS80/85-Konzept, verarbeitet acht prioritätsgesteuerte Interrupt-Eingänge, ist bis zu 64 Interrupt-Ebenen kaskadierbar und läßt sich softwaremäßig programmieren. Während des „Interrupt-Acknowledge"-Zyklus der Zentraleinheit erzeugt er einen vorher frei programmierbaren 8-bit-Vektor, der von dem 8086 als Eingang in eine der 256 Interrupt-Ebenen interpretiert wird.

In *Tabelle 2* ist die Liste der derzeit verfügbaren und kompatiblen Peripherie-Bausteine von Intel zusammengestellt. Beim Hardwareentwurf mit diesen Bausteinen ist zu beachten, daß die Adreßleitung A0 dieser Bausteine nicht mit dem Adreßbit 0 des 8086-Bus verbunden werden darf, da A0 beim 8086 zwischen

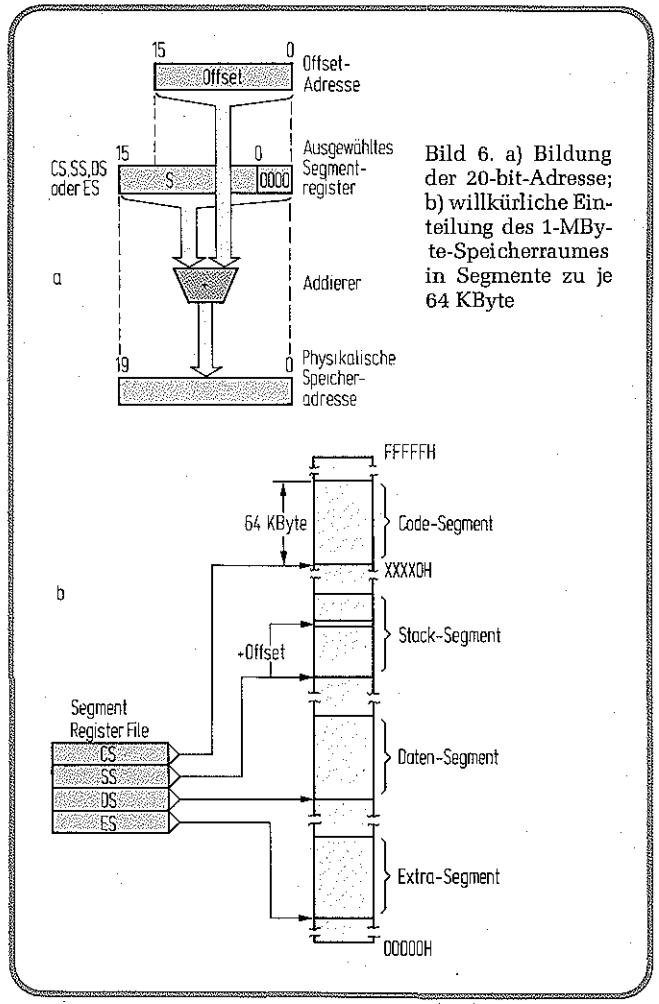


Bild 6. a) Bildung der 20-bit-Adresse; b) willkürliche Einteilung des 1-MByte-Speicherraumes in Segmente zu je 64 KByte

Tabelle 2. Kompatible Peripherie-Bausteine

8202	Programmierb. Dyn.-RAM-Controller
8251A	Programmierb. Kommunikations-Interface (UART)
8253-5	Programmierb. Intervall-Zeitgeber
8255A-5	Programmierb. Peripherie-Interface
8257-5	Programmierb. DMA-Controller
8259A	Programmierb. Interrupt-Controller
8271	Programmierb. Floppy-Disk-Controller
8273	Programmierb. HDLC/SDLC-Controller
8275	Programmierb. CRT-Controller
8278	Programmierb. Tastatur/Display-Interface
8279	Programmierb. Tastatur/Display-Interface
8291	GPIB-Talker/Listener (IEEE488-Bus-Interface)
8292	GPIB-Controller (IEEE488-Bus-Interface)
8294	Datenverschlüsselungs-Einheit
8295	Matrix-Drucker-Controller
UPI-41	Universelles Peripherie-Interface

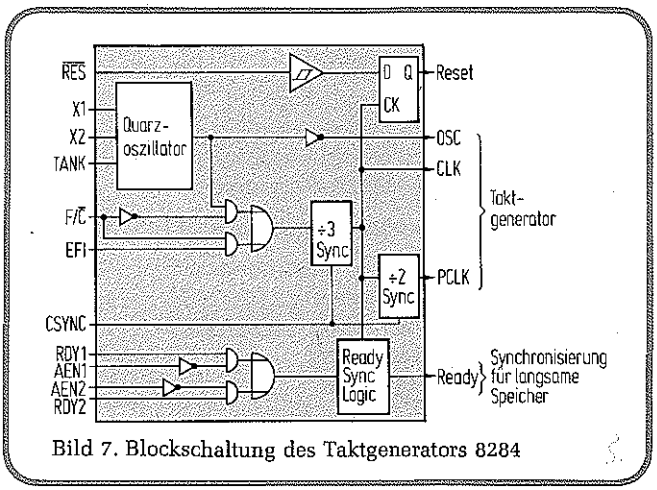


Bild 7. Blockschaltung des Taktgenerators 8284

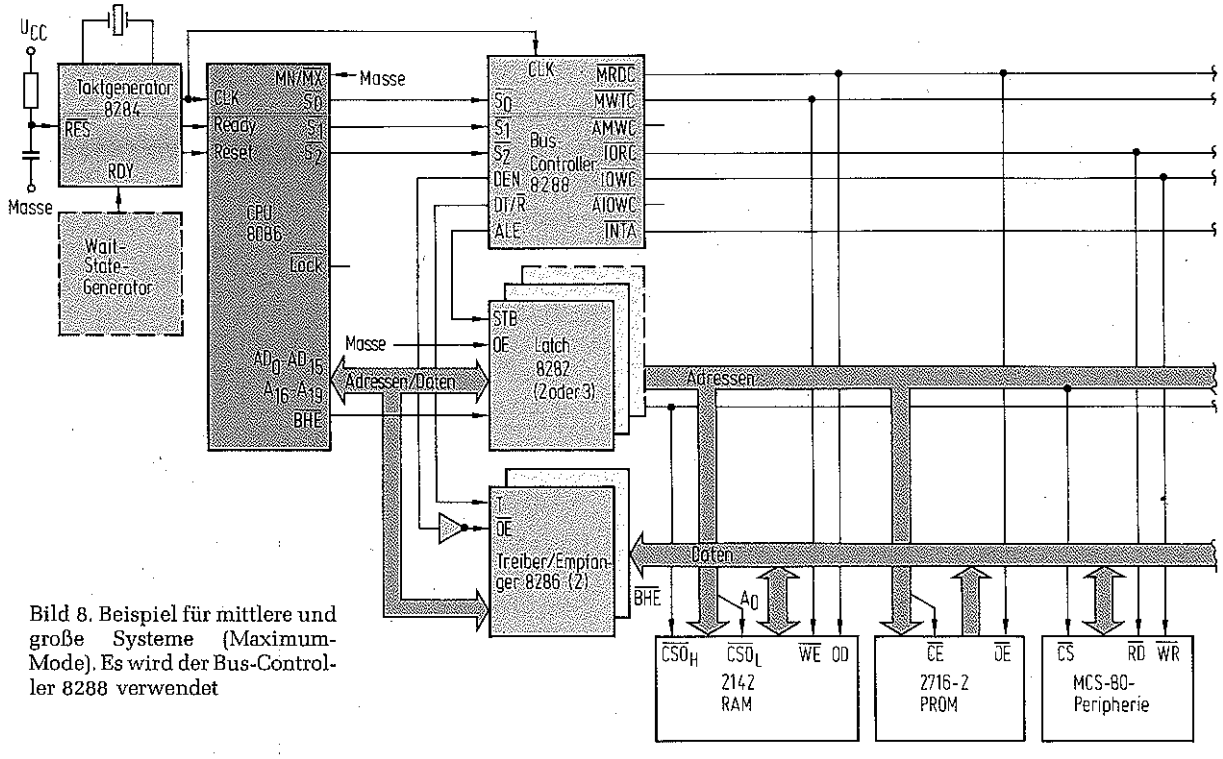


Bild 8. Beispiel für mittlere und große Systeme (Maximum-Mode). Es wird der Bus-Controller 8288 verwendet

unterem und oberem Byte des 16 bit breiten Datenbus unterscheidet, während alle Peripherie-Bausteine nur einen 8 bit breiten Datenbus haben. Durch Verschiebung der Adreßleitungen (Verbindung zwischen A0 des Peripherie-Bausteines mit A1 des 8086-Bus usw.) kann dieses Problem umgangen werden.

3 Programmierung

Durch die erwähnten Segmentregister ergeben sich folgende Eigenschaften: Da die Adressierung innerhalb eines Segmentes immer relativ zu dessen Anfang ist, lassen sich leicht dynamisch verschiebbare Programme erzeugen. Zum anderen unterstützen diese Segmentregister „Reentrant-Programme". Dieser Begriff läßt sich am besten mit „wiedereintrittsfähige Programme" übersetzen. Um ihn zu erläutern, nehme man folgendes Beispiel an (*Bild 9*): Der Programmablauf wird während eines Unterprogrammes durch einen Interrupt unterbrochen. Die Interrupt-Service-Routine beinhaltet wiederum einen Aufruf zum gleichen Unterprogramm. Die Speicherplätze, die dieses Unterprogramm während des ersten Durchlaufs benutzt hat, würden während des zweiten Durchlaufs überschrieben werden, so daß der Ablauf dieses Unterprogrammes nach Beendigung der Interrupt-Service-Routine mit falschen Daten fortgesetzt wird. Dies kann beim 8086 einfach durch Verschieben des Datensegmentes d.h. Neuladen des Segmentregisters für das Unterprogramm verhindert werden.

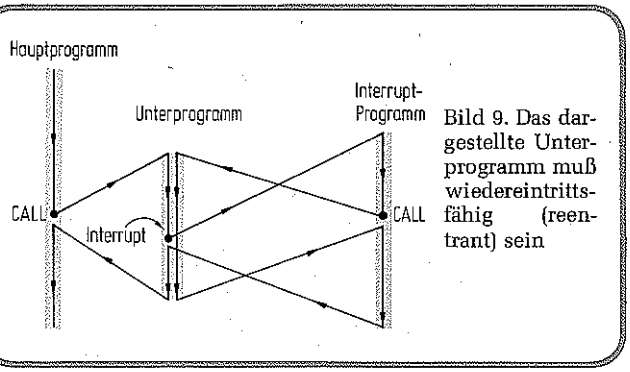
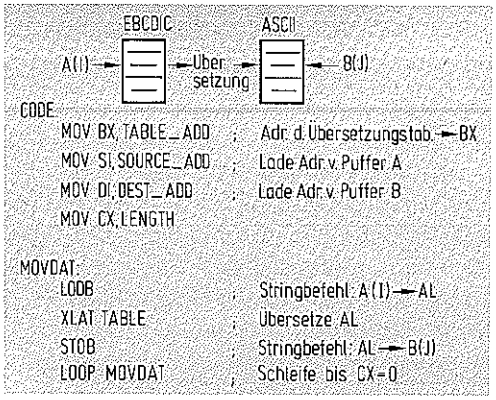


Bild 9. Das dargestellte Unterprogramm muß wiedereintrittsfähig (reentrant) sein

Der umfangreiche Befehlssatz (97 Befehlsarten) umfaßt Datentransfer-Operationen, arithmetische und logische Befehle, String-Manipulationen, die Steuer-Transfer-Gruppe (Sprünge, Unterprogrammaufrufe usw.) sowie Prozessor-Steuer-Befehle. Unter den arithmetischen Operationen sind besonders Multiplikation und Division, sowohl mit als auch ohne Vorzeichen, zu erwähnen. Mit den entsprechenden Adjust-Befehlen können alle arithmetischen Operationen auch für die dezimale Zahlendarstellung angewendet werden.

Besonders angenehm für den Programmierer zeigen sich die String-Manipulations-Befehle durch ihren modularen Charakter. So können z. B. durch Voranstellen des Wiederholungs-Präfix sehr einfach Blöcke beliebiger Länge byteweise und wortweise verschoben, bestimmte Daten innerhalb eines Blockes gesucht oder ganze Blöcke verglichen werden. Andererseits kann man durch einfache Zusammensetzung dieser String-Manipulationen mit anderen Operationen und dem Loop-Befehl sehr komplexe Manipulationen mit ganzen Datenblöcken durchführen oder z. B. Blöcke an einen Port ausgeben. Ein spezieller Ein-Byte-Befehl aus dieser Gruppe erlaubt auf einfache Art die Übersetzung eines Codes in einen anderen (*Bild 10*).



◀ Bild 10. Programmbeispiel: Übersetzung eines in EBCDIC codierten Puffers in einen ASCII-codierten Puffer

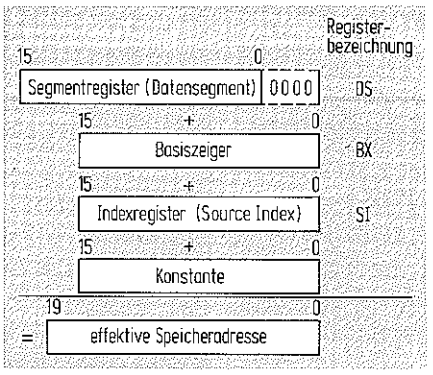


Bild 11. ▶ Mögliche Adressierungsart des 8086 am Beispiel MOV AL, (BX). KONSTANTE (SI)

Unter die Steuer-Transfer-Befehle fällt eine große Anzahl bedingter Sprünge mit einer 8-bit-Offset-Adresse und damit nur 2 Byte Befehlslänge. Generell gibt es hier drei verschiedene Adressierungsarten: Der 8-bit-Offset erlaubt einen Sprung innerhalb des Bereiches von ±128 Byte relativ zum Sprungbefehl selbst. Die verschiedenen Sprung- und Call-Befehle können mit einem 16-bit-Offset beliebig die Steuerung innerhalb eines ganzen Segmentes übertragen, während ein 32-bit-Offset die Adressierung innerhalb des gesamten Speicherraumes von 1 MByte erlaubt. Hierbei werden je 16 bit für das Segmentregister und den Befehlszeiger verwendet. Sowohl bei der Adressierung innerhalb eines Segmentes (64 KByte) als auch innerhalb des gesamten Speicherraumes steht neben der direkten auch die indirekte Adressierungsart zur Verfügung. Verschiedene Software-Interrupts ergänzen das Spektrum der Steuer-Transfer-Gruppe.

Die Ein/Ausgabe-Befehle können bis zu 64 K – auch indirekt über ein Register – adressieren. Auch hier lassen sich die Daten byteweise und wortweise übertragen.

Ganz aus dem Rahmen der eben beschriebenen Gruppen fällt der Escape-Befehl. Mit seiner Hilfe können Daten oder Befehle aus dem Speicher abgerufen bzw. einem anderen, dem 8086 parallel geschalteten Prozessor zur Verfügung gestellt werden. Der 8086 führt in diesem Fall keine Operation mit den Daten aus, während seine komplexen Adressierungsarten voll genutzt werden können.

Der Prozessor bietet 24 verschiedene Adressierungsarten. Aus *Bild 11* kann man am Beispiel eines MOV-Befehles die Komplexität der Adressierungsarten ersehen. Hier werden vier verschiedene Quellen, das Segmentregister, der Basiszeiger, das Ziel-Indexregister und eine dem Befehl folgende Konstante addiert, um die physikalische Speicheradresse zu bilden. Dieses Adressierungsbeispiel läßt sich auch auf Sprungbefehle und Unterprogrammaufrufe übertragen.

Die Länge der Befehle variiert zwischen 1 und 6 Byte in Inkrementen zu einem Byte, womit eine optimale Speicherausnutzung gewährleistet ist. Insbesondere gilt dies auch für die Konstanten, die einem 2-Operanden-Befehl folgen. Lassen sich diese Konstanten mit 8 bit darstellen, so wird trotz Wortoperation dem Befehl nur ein Byte angehängt.

4 Entwicklungshilfsmittel

4.1 Hardware

Als Entwicklungssystem zur Software-Entwicklung steht das MDS 800 (*Mikrocomputer Development System*) oder das Modell 231 der neuen Serie II zur Verfügung. Mit diesem System können Entwicklungen für die Mikrocomputerfamilien MCS 48, MCS 80/85 und MCS 86 durchgeführt werden. Die Minimalausrüstung hierfür sind 64-KByte-Speicher und zwei Floppy-Disk-Laufwerke. Die Entwicklungs-Software für den 8086 wie auch für die anderen Mikrocomputerfamilien läuft unter dem äußerst komfortablen Betriebssystem ISIS II mit dem Editor und vielen anderen Hilfsprogrammen. Die MCS86-Software besteht zur Zeit aus dem Assembler 86, dem PLM86-Compiler, dem Übersetzer für 8080/85-Programme CONV 86, aus Binder, Lader und weiteren Hilfsprogrammen.

Zum Software- und Hardware-Test bieten sich zwei verschiedene Hilfsmittel an. Der Echtzeittestadapter ICE 86 (*In Circuit Emulator*) verbindet das Entwicklungssystem direkt mit dem Anwendersystem und gestattet den Software- und Hardware-Test direkt in der Anwenderschaltung. Dieser In-Circuit-Emulator zeichnet sich, wie auch die anderen Emulatoren für die verschiedenen Prozessorfamilien der Firma Intel, besonders durch die vielfältigen Möglichkeiten des „symbolischen Debug" aus. Dies bedeutet, daß Adressen nicht in Form ihres absoluten Wertes (z. B. bei Unterbrechungspunkten), sondern einfach durch Benutzung des Namens, der bei der Software-Erstellung vereinbart wurde, eingegeben werden. Symbolisches Debug ist die einzige Möglichkeit, große Programme oder solche, die in einer höheren Programmiersprache geschrieben wurden, sinnvoll auszutesten. Weiterhin gestattet ein Trace-Speicher das Austesten von Programmen, die in Echtzeit ablaufen müssen.

Als weitere Testhilfe bietet sich der Einkarten-Computer ISBC 86/12 mit einem Monitor (Betriebssystem), einem Verbindungskabel zu dem Entwicklungssystem und der entsprechenden Treibersoftware

für das Entwicklungssystem an. 32-KByte-RAM-Speicher, flexible Ein- und Ausgänge, das Multibus-Interface und damit die Anschlußmöglichkeit vieler anderer Karten zeichnen diese Testkonfiguration aus. Die im Entwicklungssystem erstellten Programme werden über das Verbindungskabel in den RAM-Bereich des Einkarten-Computers übertragen und können dort in Einzelschritten oder bis zu einem vorher definierten Stopp ausgeführt werden. Anzeige und Veränderung von Speicher- oder Registerinhalten wie auch die Gesamtkontrolle des Tests sind bei dieser Konfiguration von der Bedienungskonsole des Entwicklungssystemes aus möglich. Für den Anfang mit kleinen Programmen und wenigen E/A-Leitungen läßt sich statt des erwähnten Einkarten-Computers auch das einfachere Modell SDK 86 (*System Development Kit 86*) mit ähnlichen Eigenschaften für den Test einsetzen.

4.2 *Software*

In *Bild 12* sind die Software-Entwicklungshilfsmittel und deren Zusammenspiel dargestellt: Das Übersetzungsprogramm (CONV-86) konvertiert den Quell-Code für den 8080 bzw. 8085 in den des 8086. Dies bietet die Möglichkeit, fertige und ausgetestete Routinen sehr leicht auf den neuen Prozessor zu übertragen. Gewisse Einschränkungen existieren insofern, daß selbstmodifizierende Programme, Interrupt-Folgen und zeitabhängige Programmstücke wie Zeitschleifen nicht direkt übersetzt werden können.

Der Assembler 86 übersetzt den Quell-Code in den verschiebbaren Objekt-Code. Die Strukturierung für Programm und Daten bietet bei der Programmierung in Assembler Möglichkeiten, wie sie bisher nur bei Compilern für höhere Programmiersprachen zu finden waren. So haben z. B. Variable folgende Attribute: Segment, in dem sich die Variable befindet, Offset zwischen Segmentanfang und der Variablen, Anzahl der Bytes in der Variablen (*Array*) und Typ (Doppelwort, Wort oder Byte). Diese Attribute können bei der Programmierung symbolisch benutzt werden, also ohne daß man deren absoluten Wert kennt. Datenstrukturen sind die Zusammenfassung mehrerer verschiedener Variablen. Auch auf der Bit-Ebene sind symbolische Vereinbarungen sowohl für einzelne als auch für Bit-Gruppen möglich. Hierdurch wird das Testen bzw. Maskieren dieser Bits wesentlich erleichtert und die Fehlerhäufigkeit während der Programmierung wesentlich vermindert. Das letzte gilt ebenso für die Attribute der Variablen. Weitere Details, z. B. die Optimierung der Befehlslänge (automatische Auswahl, ob 8- oder 16-bit-Offset bzw. -Daten bei Sprungbefehlen usw.), Symbollänge bis zu 31 Zeichen und formatfreie Eingabe kennzeichnen diesen Assembler.

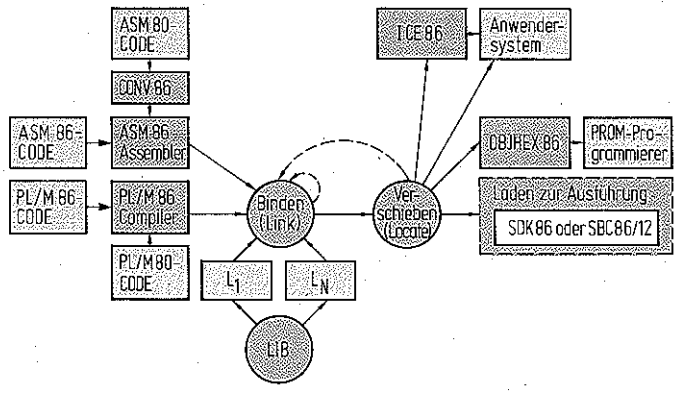


Bild 12. Zusammenspiel der Entwicklungs-Software

Je preisgünstiger die Speicher-Bauelemente werden, desto größer werden die Programme. Inzwischen liegen die Software-Kosten in den meisten Fällen weit über den Hardware-Kosten. Um sie zu verringern oder auch um die Software-Entwicklungszeit niedrig zu halten, rechtfertigt sich der Einsatz einer höheren Programmiersprache. Mit PL/M 86 steht eine Programmiersprache zur Verfügung, die sich unter anderem für Steuerungen und mathematische Anwendungen eignet. Sie zeichnet sich besonders durch Dualarithmetik (mit und ohne Vorzeichen), 32-bit-Gleitkommaarithmetik, eine große Anzahl von fertigen Anwender-Routinen und die Unterstützung von Interrupt-Programmen und „Reentrant"-Routinen aus. Interessant ist auch die Möglichkeit, die korrespondierenden Assembler-Befehle auszudrucken, so daß zeitkritische Programmteile in Assembler-Sprache optimiert werden können. Ein Optimierungslauf des PL/M 86-Compilers ergibt einen geringen Mehraufwand im Programmspeicher gegenüber der Assembler-Programmierung. Da die bei 8080 und 8085 schon länger eingesetzte Programmiersprache PL/M 80 eine Teilmenge von PL/M 86 darstellt, können solche Programme direkt für den 8086 übernommen werden.

Binder und „Locater" (Verschiebeprogramm) oder auch die kombinierte und schnelle Version für die Testphase QRL 86 verbinden mehrere verschiebbare Objekt-Moduln, die von verschiedenen Quellen kommen können (Assembler, PL/M 86-Compiler und Bibliothek), zu einem einzigen Modul und weisen absolute Adressen zu. Weitere Hilfsprogramme ergänzen die MCS86-Entwicklungssoftware zu einem vollständigen Paket.

Dipl.-Ing. Ludwig Dorn, geboren in Langenselbold bei Hanau, studierte Nachrichtentechnik an der Technischen Hochschule in Darmstadt. Schon während des Studiums leitete er die Entwicklung in einem Ingenieurbüro. Neben allgemeinen digitalen Entwicklungen führte er hier, wie auch später in einer großen Maschinenbaufirma, hauptsächlich Hardware- und Software-Entwicklungen mit Mikroprozessoren durch. Seit Frühjahr 1978 ist er bei der Firma Intel als Applikations-Ingenieur tätig.
Hobbys: Segelfliegen, Amateurfunk
Telefon: 0 61 21/7 48 55
ELEKTRONIK-Leser seit 1972

Literatur

[1] MCS-86 Users Manual. July 1978, Druckschrift der Firma Intel.
[2] HMOS Reliability. 1978, Zuverlässigkeitsbericht der Firma Intel; kurze Beschreibung der HMOS-Technologie.

Gary Daniels, BSEE
Ing. (grad.)
Max-Eberhard Lösel

Fortschritte in der Halbleitertechnologie haben es ermöglicht, daß auf einem einzigen Siliziumplättchen ein Mikroprozessor realisiert werden konnte, der im Vergleich zu seinen Vorgängern eine um eine Größenordnung höhere Packungsdichte und Leistungsfähigkeit besitzt. Der in HMOS-Technik gefertigte 16-bit-Mikroprozessor MC68000 vereinigt auf einem einzigen Siliziumkristall mehr als 68000 aktive Elemente. Diese hohe Packungsdichte und die damit verbundene Verzehnfachung des Leistungspotentials gegenüber dem bisherigen MC6800 sind das direkte Ergebnis neuer Schaltungstechniken und Fertigungsverfahren.

Ein 16-bit-Mikroprozessor in HMOS-Technik

1 Technologie und Konzept

Plasmaätzverfahren, Projektionsdruck und HMOS-Schaltungstechniken (*High density short channel MOS*) waren das technologische Fundament, auf dem die neue Mikroprozessorkonzeption realisiert wurde. Das Ziel war dabei die Entwicklung eines Mikrocomputers, der sich bei gleichzeitiger Steigerung der Leistungsfähigkeit durch einfache Handhabung und hohe Flexibilität bei der Programmierung auszeichnet.

Der MC68000 bietet dem Anwender folgende Leistungsfunktionen:

- 32-bit-Daten- und Adreßregister,
- 16 Megabyte direkt adressierbarer Speicherbereich,
- ein Befehlssatz mit 61 leistungsstarken Instruktionen,
- Operationen mit sechs verschiedenen Datentypen,
- Ein/Ausgabe mit Memory-Mapping,
- 14 verschiedene Adressierungsarten.

Bei der Konzipierung des Systems konzentrierte man sich im Hinblick auf Register, Befehle (einschließlich aller Adressierungsarten) und Datenarten auf eine klare und benutzerfreundliche Auslegung der Systemarchitektur. Für den Anwender hat ein solches Systemkonzept verschiedene Vorteile: ein besseres Verständnis der Funktionsabläufe im Prozessor, Zeiteinsparungen bei der Programmentwicklung und eine bessere Nutzung der Speicherkapazitäten durch kompaktere Programme.

Selbst bei der Verwendung von Standard-Speichern mit relativ langen Zugriffzeiten läßt sich ein hoher Systemdurchsatz erreichen (bis zu einer Gesamtleistung von zwei Millionen Befehlen und Datenwortübertragungen pro Sekunde). Die flexible Konzeption des Datenbusses erlaubt den gleichzeitigen Anschluß von langsamen wie auch schnellen Peripherieeinheiten und Speicherelementen; die unterschiedlichen Zugriffsraten werden dabei zur maximalen Nutzung der Systemleistung automatisch optimiert.

Die Hardware-Konzeption des MC68000 wurde in starkem Maße von den neuesten Entwicklungen im Bereich der Software geprägt. Die 16-bit- und 32-bit-Mikroprozessoren der dritten Generation müssen hardwareseitig alle notwendigen Voraussetzungen für einen effizienten Ablauf für Compiler Höherer Programmiersprachen und damit erzeugte Objektmoduln aufweisen. Entsprechend dieser Forderung wurde dieser Mikroprozessor mit einer großen Anzahl von Registern und Stacks, einem umfangreichen Adressierungsbereich und leistungsstarken Programmiersprachen orientierten Befehlen, wie z. B. *LINK, UNLINK, CHK* ausgestattet. Die für die Ablauforganisation der Software verantwortlichen Betriebssysteme unterstützen den Mikroprozessor durch eine Reihe leistungsfähiger Funktionen. Hierzu gehören neben privilegierten Befehlen und Speicherverwaltung eine vektorgesteuerte, leistungsstarke Interruptstruktur mit mehreren Unterbrechungsebenen, Trap-Funktionen (Fallen-Funktionen) und spezielle Befehle wie z. B. *EXG, LDM, STM* oder *TRAP*.

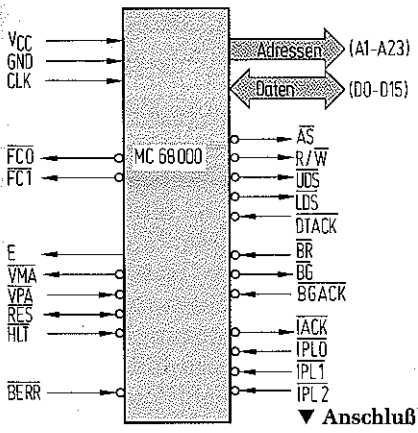
Für die Konfiguration von anwendungsspezifischen Multiprozessorsystemen ist die Mikroprozessoreinheit mit den notwendigen Software- und Hardware-Schnittstellen ausgestattet. Auf dem CPU-Chip ist die Hardware für die gemeinsame Benutzung von Systembus und Speichereinheiten (zusammen mit anderen CPU- und DMA-Bausteinen) integriert. Außerdem werden Multiprozessor-Systeme durch Befehle wie *TEST* und *SET*, TEST und *RESET* unterstützt. Diese Systemeigenschaften des MC68000 sichern dem Anwender ein Höchstmaß an Flexibilität bei der Konzeption von Multiprozessorsystemen.

2 Aufbau

2.1 CPU-Architektur

Ein leistungsfähiger Mikroprozessor muß heute komplexe, umfangreiche Aufgabenstellungen genauso effizient lösen können wie kleine, einfache Anwendungsprobleme. Unter diesem Gesichtspunkt einer vielseitigen Einsetzbarkeit wurde die Zentraleinheit konzipiert.

Zusätzlich zu dem 24-bit-Programmzähler und 16-bit-Statusregister besitzt der MC68000 siebzehn 32-



◀ Bild 1. Anschlußbelegung des Prozessorbausteins

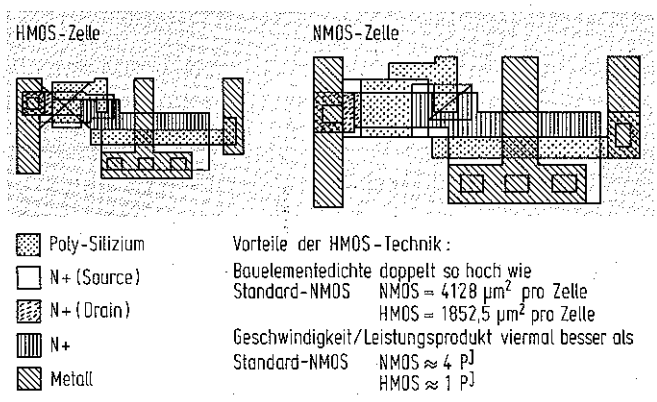


Bild 2. ▶ Vergleich zwischen HMOS- und NMOS-Technik

▼ Anschlußbezeichnungen

A1-A23	Adreßleitungen	23-bit-Adreßbus, in Verbindung mit UDS und LDS sind bis zu 16 777 216 Byte adressierbar.
D0-D15	Datenleitungen	16-bit-Datenbus, überträgt 8- oder 16-bit-Informationen.
AS	Adreß-Strobe	Zeigt gültige Adresse an und zeigt gleichzeitig nicht zulässige Busoperationen an (Bussperre).
R/W	Read/Write	Definiert Busoperationen als Schreiben oder Lesen und steuert externen Buspuffer.
UDS, LDS	Data Strobe	Identifiziert mit Hilfe von R/W und AS die benötigten Datenbytes.
DTACK	Data Transfer Acknowledge	Erlaubt einen synchronen Buszyklus mit langsamen Einheiten oder Speichern.
BR	Bus Request	Signal an den Prozessor von einer den Bus anfordernden Einheit.
BG	Bus Grant	Signal vom Prozessor zur Bewilligung der Anforderung für die Busbenutzung.
BGACK	Bus Grant Acknowledge	Quittungssignal an den Prozessor für den Empfang des BG-Signals mit der gültigen Selektion des gegenwärtigen Prozesses.
IAC	Interrupt Acknowledge	Zeigt an, daß sich der Bus in einem Interrupt-Service-Zyklus befindet.
IPL0, IPL1, IPL2	Interrupt-Priority Level	Gibt die Prioritätsebene der Interruptfunktion an den Prozessor.
FC0, FC1	Function Code	Informiert die externen Einheiten über den augenblicklichen Buszyklus.
CLK	Clock	Master-TTL-Takteingang.
RES	Reset	Übergibt Rücksetzsignal (Initialisierung) an Prozessor und Peripherieeinheiten.
HLT		Versetzt Prozessor in Haltezustand und ermöglicht eine schrittweise Programmausführung.
BERR	Bus Error	Beendet einen Buszyklus, wenn keine Rückmeldung oder eine ungültige Rückmeldung empfangen wird.
E	Enable	Enable-Clock für M6800-Systeme.
VPA	Valid Peripheral Address	Identifiziert adressierten Bereich als kompatibel mit M6800-Systemen.
VMA	Valid Memory Address	Zeigt für die Einheiten der M6800-Familie an, daß eine gültige Adresse am Bus anliegt.
V_{CC}	+5 V	–
GND	Masse (2 Anschlüsse)	–

bit-Register (*Bild 2*). Die ersten acht Register (D0...D7) sind als Datenregister für Byte-(8-bit), Wort-(16-bit-) und Doppel-Wort (32-bit-)Operationen vorgesehen. Die neun Register des zweiten Satzes lassen sich als Stapelzeiger (*Stack-Pointer*) oder Basisadreßregister verwenden. Alle 17 Register sind als Index-Register benutzbar.

Der 24-bit-Programmschrittzähler (*programm counter*) erlaubt die Adressierung eines Speicherbereichs von insgesamt 16 MByte (oder genau 16 777 216 Byte). Mit diesem großen Adressierbereich kann der Anwender ohne Paging-Methoden oder andere zeitraubende Techniken umfangreiche modular aufgebaute Programmsysteme entwickeln. Das Statusregister der Zentraleinheit (*Bild 4*) enthält die Interrupt-Maske (acht Unterbrechungsebenen) und den Condition-Code sowie die Zustandsinformationen *Overflow* (V), *Zero* (Z), *Negative* (N), *Carry* (C) und *Extend* (X). Weitere Zustandsbits geben an, ob sich der Prozessor im *TRACE*-Modus (T) oder im *SUPERVISORY*-Status (S) befindet. Zusätzlich zu diesen Feldern ist im Statusregister ausreichend Platz für künftige Erweiterungen der Prozessor-Familie reserviert.

Sechs verschiedene Arten von Daten werden verarbeitet:

- Bits,
- BCD-Ziffern (4 bit),
- ASCII-Zeichen (7 bit),
- Bytes (8 bit),
- Worte (16 bit),
- Doppel-Worte (32 bit).

Außerdem sind im Befehlssatz Instruktionen für Operationen mit anderen Datenarten, wie z. B. Speicheradressen oder Status-Wort enthalten.

Die in *Tabelle 1* aufgeführten Adressierungsarten umfassen 14 Grundtypen:

- Registeradressierung Direkt,
- Registeradressierung Indirekt,
- Absolute Adressierung,
- Unmittelbare Adressierung,
- Programmschrittzähler relative Adressierung.

Sämtliche Adressierungsarten enthalten die Möglichkeit zur Nachinkrementierung, Vordekrementierung, Offsetting und Indizierung.

2.2 Befehlssatz

Die CPU besitzt einen leistungsfähigen und vielseitigen Befehlssatz, wie aus den in *Tabelle 2* aufgeführten 61 Instruktionen ersichtlich ist. Bei der Entwicklung dieses Prozessors lag das Schwergewicht vor allem auf einer benutzerfreundlichen Systemausstattung zur Unterstützung von strukturierten Höheren Programmiersprachen. Jeder einzelne Befehl des MC68000 ist mit Einbyte-, Einwort- oder Zweiwortregistern verwendbar. Bei nahezu allen Befehlen läßt sich jede beliebige Adressierung aus den 14 verfügbaren Adressierungsarten verwenden. Unter Berücksichtigung aller vorhandenen Instruktionstypen, Daten und Adressierungsarten ergeben sich insgesamt mehr als 1000 leistungsfähige Befehle. Dazu gehören: Multiplikation und Division mit vorzeichenlosen bzw. vorzeichenbehafteten Operanden, „schnelle" arithmetische Operationen, BCD-Arithmetik und erweiterte Operationen (mit Trap-Funktion). Mit diesen Befehlen besitzt der MC68000 einen der vielseitigsten und umfangreichsten Instruktionssätze. Mikrocodierung und Befehlsaufbau sorgen für eine flexible Anpassung der Mikroprozessoreinheit an aktuelle wie künftige Aufgabenstellungen.

Fortschritte in der VLSI-Halbleitertechnik haben in den letzten Jahren bei der Computerhardware zu erheblichen Preissenkungen geführt. Der MC68000 z. B. bietet heute zum gleichen Preis auf einem einzigen Baustein die drei- bis vierfache Leistung eines vor zehn Jahren hergestellten Prozessors. Im gleichen Zeitraum sind jedoch die Aufwendungen für die Software im Verhältnis zu den Gesamtkosten überproportional stark angestiegen. Die Kostensteigerung im Bereich der Software ist in erster Linie auf den personalintensiven Charakter der Programmierung und die inflationäre Preisentwicklung der letzten Jahre zurückzuführen. Nur durch einschneidende Änderungen in der Computerarchitektur kann diesem Trend entgegengewirkt werden. Deshalb wurden durch die Konzeption der CPU die notwendigen Voraussetzungen für eine Reduzierung der Softwarekosten geschaffen. Der MC68000 wurde mit einer Reihe von innovativen Funktionen ausgestattet, die die Programmierung einfacher, schneller und zuverlässiger machen.

Durch den transparenten architektonischen Aufbau der CPU wird das Schreiben von Programmen sowohl in maschinenorientierter Assemblersprache, wie auch in Höheren Programmiersprachen entscheidend vereinfacht. Register- und speicherbezogene Operationen mit ganzzahligen Daten lassen sich unabhängig von den Daten durchführen. Eigene spezielle Befehle für 8-bit-, 16-bit- und 32-bit-Worte sind nicht mehr erforderlich. Der Programmierer merkt sich nur den mnemonischen Operations-Code für jeden auszu-

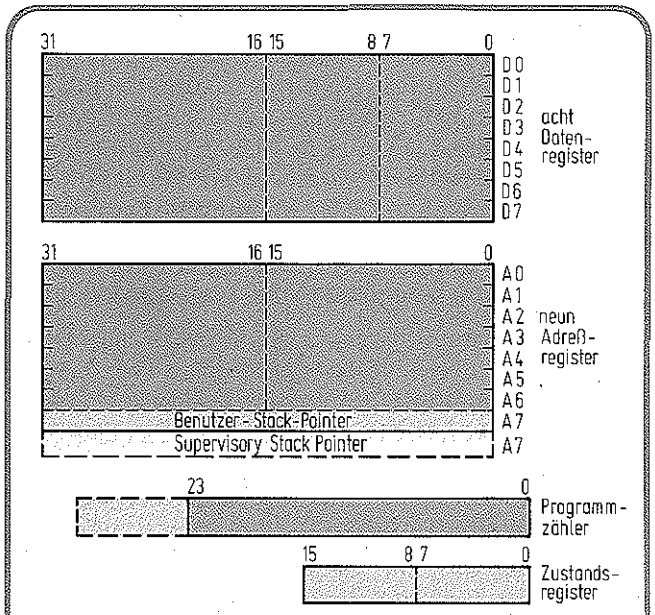


Bild 3. Registersatz

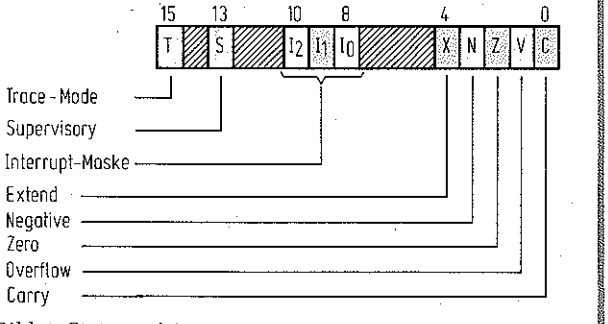


Bild 4. Statusregister

Tabelle 1. Adressierungsarten

Direkte Adressierung der Register	
Datenregister direkt	EA = Dx
Adreßregister direkt	EA = Ax
Statusregister direkt	EA = SR
Absolute Datenadressierung	
A Absolut kurz	EA = (nächstes Wort)
B Absolut lang	EA = (nächsten zwei Worte)
Programmschrittzähler relative Adressierung	
Relativ mit Offset	EA = (PC) + D16
Relativ mit Indizierung & Offset	EA = (PC) + (Rx) + D8
Indirekte Adressierung über Register	
Register indirekt	EA = (Ax)
Register indirekt mit anschließender Inkrementierung	EA = (Ax), Ax ← Ax + N
Register indirekt vorhergehender Dekrementierung	Ax ← Ax – N, EA = (Ax)
Register indirekt mit Offset	EA = (Ax) + D16
Indiziert Register-indirekt mit Offset	EA = (Ax) + (Rx) + D8
unmittelbare Datenadressierung	
Direkt	DATA = nächstes Wort bzw. Worte
Schnell direkt	INHERENT DATA

Definitionen:

EA	= Effektive Adresse
Ax	= Adreß-Register
Dx	= Datenregister
Rx	= Adreß- oder Datenregister benutzt als Indexregister
SR	= Statusregister
PC	= Program Counter (Programmschrittzähler)
D8	= 8 bit Offset
D16	= 16 bit Offset
N	= 1 für Byte, 2 für Wort und 4 für Doppelwort
( )	= Inhalt von
←	= wird ersetzt durch

führenden Befehl und gibt dann Datengröße und Adressierungsart für den Quellen- bzw. Zieloperanden an. Auf diese Weise konnten die symbolischen Befehlsanweisungen auf einen leicht zu merkenden und doch umfassenden Satz von 61 Instruktionen reduziert werden – 11 weniger als beim Mikroprozessor MC6800. Die Implementierung von Mehrfachfunktionsbefehlen hat zu einer wesentlichen Steigerung der Flexibilität und Leistungsfähigkeit geführt. Hinzu kommt, daß alle Datenregister und Speicherzellen für nahezu alle Operationen mit ganzzahligen Daten sowohl als Ziel wie auch als Quelle verwendet werden können.

Die 14 Adressierungsarten konnten ohne Verlust an Leistungsfähigkeit äußerst einfach gehalten werden. Sie sind genau aufeinander abgestimmt und unabhängig von den jeweiligen Befehlsoperationen. Außerdem lassen sich alle Adreßregister für die direkten, Register-direkten und indizierten Adressierungsarten verwenden (Unmittelbare, Programmschrittzähler, Relative und Absolute Adressierung benutzen keine Adreßregister). Eine Steigerung der Flexibilität wird außerdem dadurch erreicht, daß sich alle Datenregister und Adreßregister als Index-Register verwenden lassen. Bei Stackoperationen kann jedes der acht Adreßregister als Stapelzeiger für die Unterprogrammbearbeitung mit den Adressierungsarten Register-Indirekt, Nachinkrementierung/Vordekrementierung benutzt werden. Register A7 ist ein spezielles Register. Dieses läßt sich zusätzlich zu seiner normalen Adressierungsfunktion als System-Stapelzeiger verwenden, wenn für Unterprogrammaufruf Trap- und Interruptfunktionen, Programmzähler und Statusregister im Stack abgespeichert werden sollen.

2.3 Strukturierte modulare Programmierung

Für das Programmieren von Mikroprozessoren wurden in den letzten Jahren zahlreiche neue Techniken und Verfahren entwickelt, die zu erheblichen Leistungssteigerungen und Qualitätsverbesserungen in der Programmentwicklung geführt haben. Den meisten dieser Methoden liegt das Prinzip der Modularität zugrunde, die Aufteilung einer bestimmten Aufgabe oder Funktion in kleine, leichter überschaubare Bausteine und Unterroutinen, die sich einfacher programmieren und austesten lassen. Eine entscheidende Rolle spielt dabei die Verwendung von leistungsfähigen Makroassemblern und blockstrukturierten Höheren Programmiersprachen, wie z. B. PASCAL. Solche Konzepte sind jedoch nur dann sinnvoll, wenn die einfache Übergabe von Parametern innerhalb und zwischen den *„reentrant"* geschriebenen, wiedereintrittsfähigen Softwaremoduln gewährleistet ist. (Ein wiedereintrittsfähiges Programm muß sowohl von interruptgesteuerten wie auch von nicht-interruptgesteuerten Programmen ohne Datenverlust aufgerufen werden können.)

Die *„LINK"* und *„UNLINK"*-Befehle erlauben durch die Möglichkeit der Manipulierung verbundener Datenbereichslisten im Stack eine Vereinfachung bei Unterprogrammaufrufen mit Hilfe von zwei komplementären Befehlen. Den gleichen Vorteil einer Vereinfachung bei der Programmierung von Unterprogrammaufrufen bieten die Befehle *„STM"* (*Store Multiple Registers*) und *„LDM"* (*Load Multiple Registers*). Diese Befehle gestatten über eine effektive Adresse das Laden und Speichern mehrerer vom Programmierer spezifizierter Register. Mit dem *„TRAP"*-Befehl stehen

Tabelle 2. Befehlssatz

Mnemonisch	Beschreibung
ABCD	Add Decimal with Extend
ADD	Add
ADDX	Add with Extend
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Arithmetic Compare
DCNT	Decrement and Branch Non-Zero
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LDM	Load Multiple Registers
LDQ	Load Register Quick
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Two's Complement
NEGX	Two's Complement with Extend
NOP	No Operation
NOT	One's Complement
OR	Logical Or
PACK	Pack ASCII to BCD
PEA	Push Effective Address
RESET	Reset External Devices
ROTL	Rotate Left without Extend
ROTR	Rotate Right without Extend
ROTXL	Rotate Left with Extend
ROTXR	Rotate Right with Extend
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
SCC	Set Conditional
STM	Store Multiple Registers
STOP	Stop
SUB	Subtract
SUBX	Subtract with Extend
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink Stack
UNPK	Unpack BCD to ASCII

für das Arbeiten mit Unterprogrammen und für benutzergenerierte Makroroutinen 16 Software-Trap-Vektoren zur Verfügung. Weitere Befehle für die strukturierte Programmierung sind: *PEA (Push Effective Address)*, LEA (*Load Effective Address*), *RTR (Return to Restore)* sowie normale *JSR-*, *BSR-* und *RTS*-Instruktionen. Eine einfache Generierung von wiedereintrittsfähigen modularen Ein/Ausgabe-Routinen wird durch die leistungsfähige vektorgesteuerte Prioritäts-Interruptstruktur des Mikroprozessors erreicht. Acht maskierbare Prioritätsebenen mit 192 Vektoradressen garantieren ein Höchstmaß an Flexibilität bei der Ein/Ausgabe-Steuerung. (Insgesamt gibt es 256 Vektoradressen für Unterbrechungen, Hardware- und Software-Traps.)

2.4 Erweiterte Funktionen für das Austesten von Programmen

Einen großen Teil seiner Zeit verwendet der Programmierer von Mikroprozessorsystemen auch heute noch für die Fehlersuche und Fehlerkorrektur (Debugging). In fast allen Fällen erfordert das Austesten mehr Zeit als das Schreiben der Programme. In der Praxis gilt die Faustregel: „Die letzten 20 % der Arbeit verbrauchen 80 % der gesamten Arbeitsleistung". Der Mikroprozessor MC68000 besitzt eine ganze Reihe von Eigenschaften, durch die die Fehlermöglichkeit bei der Programmierung entscheidend reduziert wird. Hierzu gehören z. B. der benutzerorientierte Aufbau der Rechnerarchitektur und die Möglichkeit zur strukturierten modularen Programmierung.

Von besonderer Bedeutung für den Programmierer sind Funktionen, die ihm die Suche nach Programmfehlern entscheidend erleichtern. Hierzu gehören Hardware-Trap-Funktionen zur Anzeige anormaler interner Zustände des Mikroprozessors. Folgende Fehlerzustände werden festgestellt:

- Wortzugriff mit einer ungeraden Adresse,
- Unzulässige Befehle,
- Nichtimplementierte Befehle,
- Unzulässige Adressierungsart,
- Unzulässiger Speicherzugriff (Bus-Fehler),
- Überlauf bei Division (Division durch Null),
- Überlauf-Condition-Code (eigener Befehl TRAPV).

Außerdem stehen dem Programmierer die sechzehn Software-TRAP-Instruktionen für anwendungsorientierte Fehlersuch- und Fehlerkorrektur-Routinen zur Verfügung.

Ein weiterer Befehl zur Fehlersuche ist die *CHK* (*Check Register Against Bounds*-)Instruktion. Sie dient zur Datenfeldkontrolle durch Prüfung von $0 \leqq (REG) < LIMIT$. Ist der Inhalt des Registers negativ oder größer als die Feldgrenze, wird ein Trap (vektorgesteuerter Interrupt) erzeugt.

Schließlich gibt es eine Funktion, mit der sich der Programmablauf Befehl für Befehl verfolgen läßt. In diesem sogenannten *TRACE MODE* erfolgt nach jeder Befehlsausführung ein CPU-generierter Sprung auf eine Programmablaufroutine (*tracing routine*). Der *TRACE MODE* steht dem Programmierer in beiden Mikroprozessorzuständen – *SUPERVISORY-* und *USER*-Status – zur Verfügung; er kann jedoch nur eingegeben werden, wenn sich der Prozessor im *SUPERVISORY*-Status befindet. Mit dem *SUPERVISORY/USER*-Status besitzt der Mikroprozessor eine zusätzliche Fehlerschutzfunktion: Bei Verwendung einer externen Memory-Management-Einheit lassen sich ausgewählte Speicherbereiche als geschützte Bereiche definieren.

3 Besonderheiten des Software-Konzepts

3.1 Flexibilität für zukünftige Entwicklungen

Die Neuentwicklungen der letzten Zeit im Bereich der VLSI-Schaltungstechnologie haben das Innovationstempo auf dem Gebiet der Mikroprozessor-Technik entscheidend beeinflußt. So wurde zum Beispiel seit der Ankündigung des Mikroprozessors MC6800 eine ganze Reihe von neuen hochentwickelten Produkten konzipiert. Diese Entwicklung setzte zwei Schwerpunkte: Erhöhte Funktionalität beim MC6802 und MC6801, mehr Leistung bei den Mikroprozessoren MC68A00, MC68B00 und MC6809 (*Bild 5*). Bei der Entwicklung dieser Mikroprozessoren wurden vor allem die Vorteile der benutzerorientierten Prozessorarchitektur des MC6800 und die durch neue Schaltkreistechniken erzielten Verbesserungen in der Verarbeitungsgeschwindigkeit genutzt. Bei allen Produkten war die volle Kompatibilität zum MC6800 ein wichtiger Bestandteil der Prozessorkonzeption.

Eine weitere Zielsetzung bei der Entwicklung des MC68000 war eine hohe Anpassungsfähigkeit des Mikroprozessors an künftige Aufgabenstellungen. Zu diesem Zweck wurde dieser Mikroprozessor mit einer Vielzahl von Funktionen ausgestattet, die hohe Flexibilität bei künftigen Systemerweiterungen gewährleisten. So bietet der Mikroprozessor z. B. die Möglichkeit, 8-bit-, 16-bit- und 32-bit-Operationen durchzuführen, ohne daß dabei Registerkettung oder eine Multiplexverarbeitung des internen Datenbusses erforderlich wäre.

Auch in der Konzeption des Befehlssatzes wurde von Anfang an die aus der Weiterentwicklung der Halbleitertechnik resultierenden Möglichkeiten zur Systemerweiterung berücksichtigt. So ist z. B. ein Achtel der Op-Code-Liste für die Implementierung

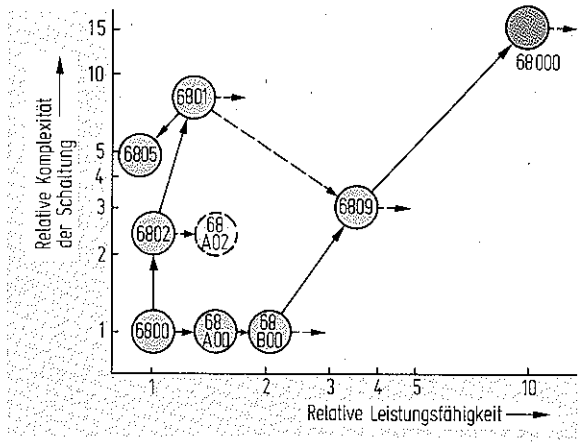
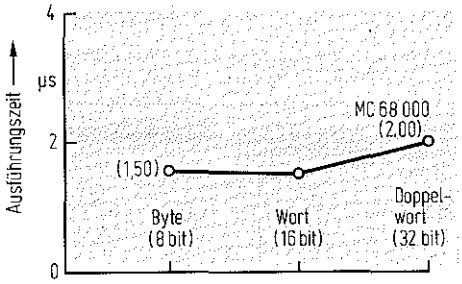
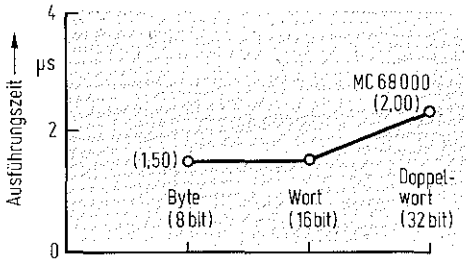


Bild 5. Entwicklungstendenz der Motorola-Mikroprozessoren



◀ Bild 6. Ausführungszeit für Addition eines Datenelementes, adressiert mit kurzer absoluter Adresse, zu einem Register

Bild 7. ▶ Ausführungszeit für Transport eines Datenelementes, adressiert mit kurzer absoluter Adresse, aus dem Speicher in die Register



von neuen Instruktionen reserviert. In der Zwischenzeit kann der Benutzer mit den verfügbaren Trap-Funktionen eigene spezielle Befehle definieren.

3.2 *Speicherverwaltung des großen Adreßbereichs*

Sinkende Kosten für Speicherelemente und die Verwendung von Höheren Programmiersprachen und leistungsfähigen Betriebssystemen – dies sind die Hauptgründe für die vielseitigen Einsatzmöglichkeiten der neuen Mikroprozessorgeneration in komplexen, speicherintensiven Anwendungsbereichen. Um den damit verbundenen hohen Anforderungen gerecht zu werden, wurde der Mikroprozessor MC68000 mit Funktionen ausgestattet, die eine direkte Adressierung von 16-MByte-Speichern erlauben. Die Verwaltung und der direkte Zugriff auf diesen umfangreichen Adreßbereich erfolgt auf Wort- oder Byte-Basis, da die Größe des Operanden bei der Befehlsangabe definiert werden kann. Durch die beiden vorhandenen Signale „*Upper Data Strobe (UDS)*" und „*Lower Data Strobe (LDS)*" wird der Zugriff auf niederwertige und höherwertige Bytes oder ganze Worte erheblich vereinfacht.

Weitere leistungsfähige Funktionen stehen zur Verfügung, die dem Programmierer eine effiziente Verwaltung des Speichers erlauben. Hierzu gehören Adressierungsarten wie: Register-indirekt, indizierte Adressierung sowie *Short* und *Long Absolute* und Programmschrittzähler-relative Adressierung. Diese gestatten auf einfache Weise Adreßrechnungen (*Register-indirekt* und *indiziert*), direkten Zugriff auf Speicheradressen (*Short* und *Long Absolute*) sowie positionsunabhängige und speicherverschiebbare (*relocatable*) Programmcodierung (*Programmschrittzähler-relativ*). Eine weitere Adressierungsart, die eine effiziente Speicherverwaltung ermöglicht, ist die indirekte Adressierung mit Vordekrementierung und Nachinkrementierung. Sie erlaubt dem Programmierer die Generierung von bis zu acht simultanen Stacks. Für das Speichermanagement steht dem Programmierer außerdem ein spezieller Befehl zur Verfügung: Der Befehl *CHK (Check Register Against Bounds)* ist für die Implementierung einer Speicherschutz/Managementstruktur vorgesehen.

Eine besonders nützliche Eigenschaft des MC68000 ist die Unterscheidung zwischen einem *USER*- und einem *SUPERVISOR*-Modus. Im *SUPERVISOR*-Modus lassen sich bestimmte geschützte Operationen innerhalb des Prozessorsystems durchführen. Im *USER*-Modus bietet der MC68000 die Möglichkeit, einen externen Memory-Management-Controller zu verwenden, der dem Programmierer die Verwaltung großer Adreßbereiche erleichtert. Die Speicherverwaltungsoperationen des Controllers können vom Programmierer überwacht werden, wenn sich der Prozessor im *USER*-Modus befindet. Eine Änderung bzw. Aktualisierung ist jedoch nur im *SUPERVISOR*-Modus möglich. Der Memory-Management-Controller erlaubt sowohl die Verwaltung von verschiedenen Speichersegmenten unterschiedlicher Größe (*Memory Segmentation*) wie auch die dynamische Verwaltung von Multi-Task-Verschiebungen (*relocation*) und geschützten Speicherbereichen. Außerdem regelt der Memory-Management-Controller den Zugriff auf Speichersegmente, die Nur-Lesedaten, Lese/Schreibdaten, Programme oder geschützte Daten sind.

3.3 *Erhöhung der Durchsatzgeschwindigkeit*

Die Entwicklung preiswerter hochintegrierter RAM- und ROM-Speicherelemente in VLSI-Technologie ändert nichts daran, daß die Anzahl der für die Programmausführung erforderlichen Byte-Codes auch weiterhin zu den wichtigsten Leistungsmerkmalen eines Mikroprozessors gehört. Die Arbeitsgeschwindigkeit eines Mikroprozessors ist im hohen Maße davon abhängig, wieviel Befehlsworte pro Zeiteinheit verarbeitet werden können. Bereits zu Beginn der Entwicklungsarbeiten für den MC68000 wurden deshalb umfangreiche Untersuchungen über die Benutzung von Befehlen und deren Ablaufstruktur in verschiedenen Mikroprozessoranwendungen durchgeführt. Diese Untersuchungen befassen sich sowohl mit dynamischen wie auch mit der statischen Häufigkeit von Befehlen (die dynamische gibt an, wie oft ein bestimmter Befehl ausgeführt wurde, die statische, wie oft ein Befehl in einem Programmausdruck erscheint, bzw. wie oft er von einem Assemblerprogramm übersetzt worden ist). Das benutzerfreundliche, transparente Architekturkonzept des MC68000 ist größtenteils das Ergebnis dieser umfangreichen Untersuchungen. Rechnerarchitektur, Befehlssatz und Adressierungsarten wurden mit der gleichen Zielsetzung entwickelt: Reduzierung der für eine bestimmte Aufgabe erforderlichen Befehle. Eine weitere

Leistungssteigerung bei der Programmierung wird durch spezielle Befehle erreicht, die eine Verbesserung der Codierdichte und kürzere Befehlsausführungszeiten ermöglichen. So erlauben z. B. Einzelwort-Additions- und Subtraktionsbefehle mit schneller direkter Adressierung (*Quick Immediate Addressing*) arithmetische Operationen mit kleinen Werten direkt im Datenregister. Ein „*Load Quick Immediate*" (LDQ-)Befehl ermöglicht das Laden eines kleinen (8 bit langen) Wortes in ein beliebiges Register mit einer Einzelwort-Operation. Zur Beschleunigung von Schleifenoperation steht eine Einzelwortinstruktion für Dekrementierung um eins und Sprung, wenn ungleich Null (*DCNT*), zur Verfügung. Die Befehle *TRAP*, Store Multiple Register (*STM*), Load Multiple Register (*LDM*), Link Stack (*LINK*), Unlink Stack (*UNLK*) und Check Limit (*CHK*) führen zu einer beträchtlichen Reduzierung des Codieraufwandes für Unterprogramme, Betriebssystemaufrufe und Stack-Operationen.

Die Befehle „*Signed*" und „*Unsigned Multiply*" (*MULS* und *MULU*), „Signed" und „Unsigned Divide" (*DIVS* und *DIVU*) führen neben BCD-arithmetischen Instruktionen (*ABCD, SBCD, PACK* und *UNPD*) und standardmäßigen binären Operationen von ganzen Zahlen zu einer Reduzierung des Codieraufwands und zu einer erhöhten Leistung bei arithmetischen Operationen. Für den Transport von Daten besitzt der Mikroprozessor einen leistungsfähigen *MOVE*-Befehl, mit dem sich sowohl Bytes, Worte und Doppelworte übertragen bzw. transportieren lassen. Es sind folgende Daten-Transporte möglich: Register zu Register, Register zu Speicher, Speicher zu Register und Speicher zu Speicher. Zusätzlich zu diesen leistungsfähigen Befehlen, die zu einer erheblichen Steigerung des Systemdurchsatzes beitragen, ist der MC68000 mit einer Reihe von Funktionen ausgestattet, die eine Reduzierung der Ausführungszeiten aller Befehle erlauben. Hierzu gehören vor allem der separate Daten- und Adreßbus (ohne Multiplex-Technik), Instruction „*Prefetch Pipeline*" sowie interne 32-bit-Register.

3.4 *Software-Hilfsmittel und Kompatibilität zum MC6800*

Systementwicklern und Programmierern, die den MC68000 in einer Applikation verwenden wollen, steht ein vollständiges kompatibles Hard- und Softwaresystem zur Verfügung. Die Software-Hilfsmittel umfassen:

- Disk Operating System,
- Debug-Programme,
- Assembler,
- Höhere Programmsprachen,
- Translator zur Übersetzung bereits vorhandener Programme für den MC6800, so daß sie auf dem MC68000 ablaufen können. Dabei sind nur minimale Eingriffe des Programmierers erforderlich.

Der Instruktionssatz des MC68000 wurde so entworfen, daß er ein Superset der Instruktionen des MC6800 darstellt. Dadurch wird eine Übersetzung des Programms des MC6800 möglich, die nach erfolgter Umsetzung für den Ablauf auf dem MC68000 weiter optimiert werden können.

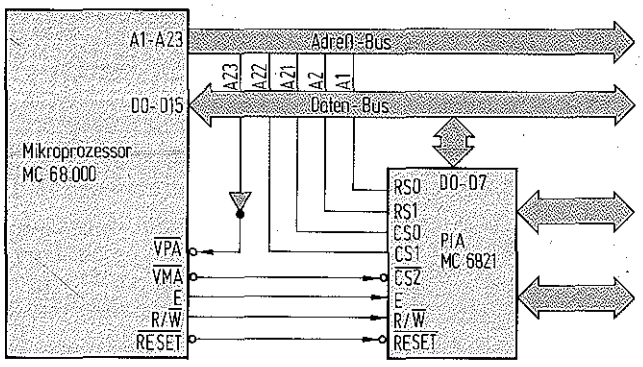


Bild 8. Anschluß des PIA-Bausteins MC 6821 an den Prozessor MC 68000

Die Software-Kompatibilität wird auch ausgedehnt auf die Anschlußmöglichkeit von Peripherie-Controller. Motorolas umfangreiche Liste von intelligenten Peripherie-Elementen, wie z. B. der *ADLC* MC6854 und der *GPIA* MC68488, können direkt ohne Schwierigkeit an den MC68000 angeschlossen werden.

Auf der Hardwareseite besteht durch die drei Signalleitungen *Enable (E), Valid Memory Address (VMA)* und *Valid Peripheral Address (VPA)* die Möglichkeit zur Integration zahlreicher Bausteine der Familie 6800 in ein System auf Basis des MC68000. *Bild 8* zeigt beispielsweise, wie der Interface-Adapter MC6821 (*PIA*) angeschlossen wird.

Der Anschluß des neuen Einchip-Mikrocomputers MC6801E ist ebenso möglich und gestattet dem Anwender die Entwicklung spezieller Eingabe/Ausgabe-Funktionen.

Zum MC68000 gibt es universelle Peripherie-Controller, die der neuen Mikroprozessor-Architektur angepaßt sind, wie z. B. DMA-Controller und Memory-Managementeinheit.

Literatur

[1] *Schmid, H.*: Monolithische Mikrocomputer-Bausteine. ELEKTRONIK 1979, H. 8, S. 59...67 und H. 9, S. 57...64 (Gesamtübersicht).

[2] *Huse, H.*: Das TMS 9900/TI 990-Konzept. ELEKTRONIK 1976, H. 11, S. 95...99.

[3] *Shima, M.*: 16-bit-Chip genügt Mikro- und Minicomputeranwendungen. ELEKTRONIK 1979, H. 8, S. 83...89; (Z 8000).

[4] *Dorn, L.*: HMOS-Prozeß führt zu leistungsfähigem 16-bit-Mikroprozessor. ELEKTRONIK 1979, H. 5, S. 57...63; (8086).

Gary Daniels, BSEE an der University of New Mexico 1965. Freier Mitarbeiter an der Arizona State University. Bei Motorola/Austin Texas seit 1966, wurde hier 1975 MOS-Mikrocomputer-Design-Manager, Verfasser zahlreicher Veröffentlichungen, Inhaber von 6 US-Patenten, einige weitere sind angemeldet. Mitglied des IEEE. (Foto liegt nicht vor.)

Ing. (grad.) Max-Eberhard Lösel, geboren in Dessau, studierte in Karlsruhe, anschließend bei GfK in Karlsruhe, ging dann zu Texas Instruments und Fairchild. Seit 1978 bei Motorola als Leiter des Technischen Service und des MPU-System-Verkaufs. Hobbys: Tauchen, Filmen und Amateurfunk. Privat-Tel. (0 61 21) 46 38 61
ELEKTRONIK-Leser seit 1964

Masatoshi Shima

Obwohl der erste 16-bit-Mikroprozessor seit 1976 auf dem Markt ist, spricht man erst jetzt, da weitere Typen hinzugekommen sind, vom Anbrechen einer neuen Ära. Ein typischer Vertreter dieser Prozessor-Generation ist der Z8000, der in zwei verschiedenen Ausführungen angeboten wird.

16-bit-Chip genügt Mikro- und Minicomputeranforderungen

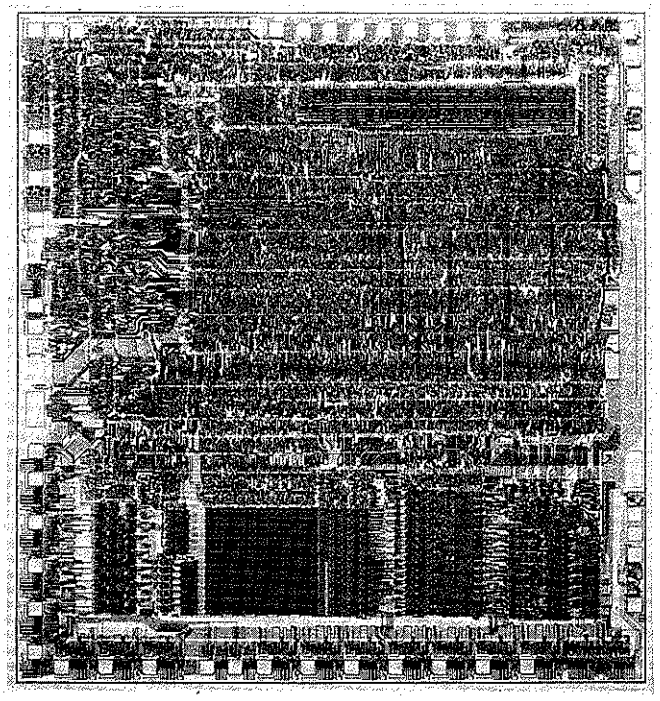
Ein Mikrocomputer trifft zur Zeit dann die Anforderungen des Marktes, wenn er die Anwendungen der gegenwärtigen 8- und 16-bit-Typen abdeckt, gleichzeitig aber eine Architektur aufweist, die ihm eine lange Produktlebensdauer garantiert. Der Z8000 erfüllt die erste Forderung mit Leichtigkeit, er übertrifft sie sogar mit einem Mehrfachen des Durchsatzes existierender Mikroprozessoren. Um die zweite Forderung zu erfüllen, gingen seine Entwickler von der traditionellen byteorientierten Struktur ab und entwarfen eine CPU mit minicomputerähnlicher (regelmäßiger) Architektur. Damit (und mit einigen Pluspunkten gegenüber Minis) ausgestattet, eignet sich dieser Prozessor für Minicomputer und µP-Anwendungen.

Einige seiner hervorstechenden Merkmale sind: Er handhabt sieben Datentypen, von Bits bis Zeichenketten (*String*), und bietet acht Adressierungsarten zur Auswahl. Mit seinen 81 Befehlstypen, die, kombiniert mit den verschiedenen Adressierungsarten und Datentypen, 414 Befehle ergeben, übertrifft er in dieser Hinsicht die meisten Minicomputer. Der Befehlssatz ist außerdem sehr übersichtlich gestaltet: Mehr als 90 % der Befehle können in jeder der fünf Haupt-Adressierungsarten verwendet werden, mit 8-bit-, 16-bit- oder 32-bit-Daten.

Was die Architektur betrifft, so hat er eine große Zahl von Registern auf dem Chip: an 16-bit-Registern insgesamt 24. Sie vermindern die speicherbezogenen Befehle in den Programmen erheblich. 16 davon sind Allzweckregister, und alle außer einem können ohne Einschränkungen als Indexregister verwendet werden. Ebenfalls auf Minicomputeranwendungen zielt der 8-MByte-Speicherbereich ab, der direkt adressiert werden kann. Anstatt ihn linear adressierbar zu machen, hat man ihn in 128 Segmente zu 64 KByte eingeteilt. Diese Einteilung trifft eher die Art, wie Programmierer normalerweise den Speicherraum benutzen: Jede Prozedur und jeder Datenblock, ob lokal oder global, sind in einem eigenen Segment abgelegt. Um die Handhabung des Speicherraumes weiter zu erleichtern, kann ein Speicherverwaltungs-Baustein dynamische Verschiebeprozeduren und den Speicherschutz in großen Systemen übernehmen.

1 Konzept und Technologie

Da der Prozessor auch die bestehenden µP-Anforderungen abdecken soll, werden zwei Versionen angeboten: Neben der 48poligen Speicher-Segment-Ausführung mit 23-Adreßleitungen gibt es einen 40poligen Chip, der 64 KByte (ein Segment) adressiert. Der Übergang von der 40- zur 48poligen Version ist problemlos, da jedes „Nicht-Segment"-Programm in einem beliebigen Segment (mit Hilfe des Befehls „Lade Programmstatus") läuft.
Schließlich kennt der Z8000 zwei Betriebsarten, „System" und „Normal", die Betriebssystem und An-



Der Z8000-Chip enthält 17 500 Transistoren

wenderprogramm auseinanderhalten (wie in Computeranlagen). Jede Betriebsart (*Mode*) hat ihren eigenen Stapelspeicher (*Stack*), und globale Eigenschaften, wie vorrangige Befehle, beeinflussen den normalen Programmiervorgang nicht.

Der Chip enthält auf einer Fläche von 6 x 6,5 mm^2 17 500 Transistoren. Das wurde durch einen NMOS-Prozeß (Depletion-Load-Si-Gate-Technik) möglich, mit dem noch kleinere Strukturen als bisher erzeugt werden (ähnlich der HMOS-Technik). Auf einem Quadratmillimeter finden 148 Gatter Platz. Der Baustein wird mit einer 5-V-Spannung versorgt und benötigt einen 1-Phasen-4-MHz-Takt (250 ns). Da die CPU mindestens drei Taktzyklen für einen Speicherzyklus braucht, sind Speicher mit einer Zykluszeit von 750 ns und einer Zugriffszeit von 430 ns erforderlich.

2 Architektur

2.1 Steuersignale

Wie *Bild* 1 zeigt, hat der Prozessor zusätzlich zu Versorgungseingängen, Takteingang, Adreß- und Datenbus sechs Gruppen von Steuerleitungen: für Bus-Zeitsteuerung, Status-, CPU-Status- und Bus-Steuerung sowie für Mehrprozessorkopplungen und Interrupts. Die drei Bus-Zeitsteuer-Ausgänge koordinieren den Datenfluß über die Adreß- und Datenleitungen. Ein Adreß-Strobe-Signal zeigt an, daß die Adressen gültig sind, und ein Daten-Strobe-Signal erscheint während der Zeit, in der gültige Daten von oder zu der CPU gehen. Die Speicher-Anforderungs-Leitung (*Memory Request*) führt ein Signal, das die Anpassung von dynamischen RAMs erleichtert.

Die nächste Gruppe liefert Informationen über den CPU-Status. Eine Lese/Schreib-Leitung (*Read/Write*) gibt frühzeitig Auskunft über den kommenden Zyklus, während das Normal-System-Signal die gegenwärtige Betriebsart anzeigt. Der Zustand der Wort/Byte-Leitung hängt davon ab, ob die CPU gerade auf 16- oder 8-bit-Daten zugreift.

Die vier Status-Steuerleitungen ergeben ein 4-bit-Wort, das Aufschluß über verschiedene Bus-Zustände gibt, einschließlich Speicher-Anforderung, Stack-Operationen, Befehlshol-Phase, Interrupt-Quittierung, interne Operationen und andere.

Eine weitere Gruppe bilden die drei CPU-Status-Eingänge: „Reset" initialisiert die CPU, „Wait" signalisiert, daß der CPU-Datentransfer noch nicht ausgeführt ist, und „Stop" hält die CPU intern an (dynamische Speicher werden aber weiterhin aufgefrischt). Die Zentraleinheit kann immer dann angehalten werden, wenn das erste Wort eines Befehls geholt ist.

Alle Adreß-, Daten- und Steuerleitungen werden mit log. „0" am Eingang „Bus Request" in den hochohmigen Zustand versetzt. Damit können sie von anderen Einheiten benutzt werden. „Bus Acknowledge" bestätigt dieses Signal.

Zusammen mit bestimmten Befehlen koordinieren zwei weitere Leitungen den Verkehr mit anderen Prozessoren: Die Leitung „Multimicro output" gibt die Anforderung aus, während der entsprechende Eingang Anforderungen entgegennimmt. Damit kann

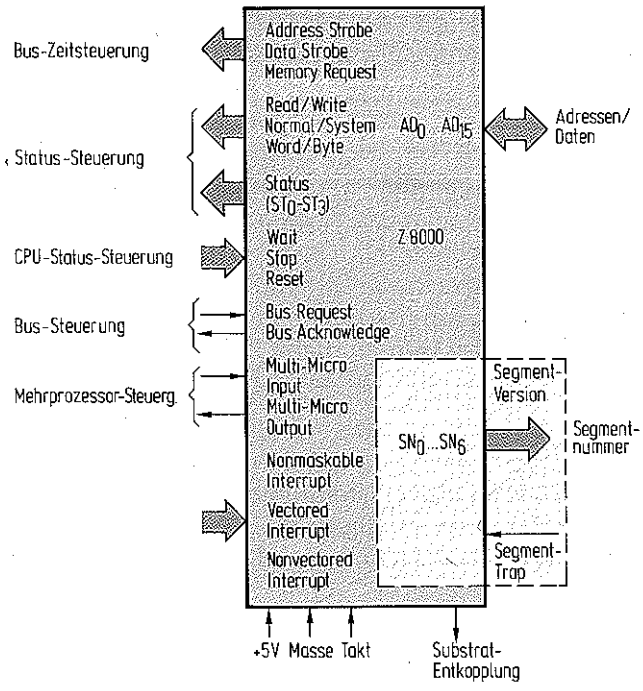


Bild 1. Die Anschlüsse des Z8000 (Zilog): Die Nicht-Segment-Version kann 64 KByte adressieren, während die minicomputerähnliche Segment-Version insgesamt einen Speicherbereich von 8 MByte ansprechen kann

Die Baustein-Familie

Selbst ein Minimalsystem mit dem Z8000 führt Befehle schnell aus und ist leicht zu programmieren. Bald wird eine Familie von Zusatz-Bausteinen erhältlich sein, mit denen diese Fortschritte auch auf komplexe Computersysteme und -netzwerke ausgedehnt werden können. Gemeint sind folgende Einheiten:

- der Speicherverwaltungs-Baustein Z-MMU (*Memory-Management Unit*), der Speicher-Segmentierung und -Schutz übernimmt und die physikalische Adresse berechnet;
- der universelle Peripherie-Controller Z-UPC (*Universal Peripheral Controller*), ein programmierbarer Peripherie-Baustein auf der Basis des Einchip-Mikrocomputers Z8;
- der Zähler und Ein/Ausgabe-Baustein Z-CIO (*Counter and Parallel I/O*) mit drei programmierbaren 16-bit-Zählern, zwei bidirektionalen 8-bit-Ports und einem 4-bit-Port;
- der serielle E/A-Baustein Z-SIO (*Serial-I/O*), der zwei Vollduplex-Kanäle hat und sowohl asynchrone als auch bisynchrone Protokolle (bis 880 KByte/s) handhaben kann;
- die Puffereinheit Z-MBU (*Microprocessor Buffer Unit*), ein 128 x 8-bit-FIFO, das bis zu beliebiger Tiefe kaskadiert werden kann und für Multiprozessor-Anwendungen geeignet ist;
- das Z-FIFO, ebenfalls ein 128 x 8-bit-FIFO, mit dem Z-MBUs erweitert oder Schnittstellen zwischen E/A-Ports und Benutzer-Peripherie aufgebaut werden;
- zwei RAMs (Z-Bus RAMs), ein statisches mit einer Kapazität von 2 K x 8 bit und ein selbstauffrischendes pseudo-statisches mit einer Kapazität von 4 K x 8 bit.

jede CPU in einem Mehrprozessorsystem jede andere, asynchron laufende CPU daran hindern, auf eine kritische Einheit zuzugreifen.

Schließlich hat der Z8000 noch drei Interrupt-Eingänge und, in der Segment-Version, einen sogenannten Trap-Eingang. Interrupts werden asynchron ausgelöst, im Normalfall von Peripherieeinheiten, die bedient werden wollen. Als Traps (zu deutsch Falle) bezeichnet man synchrone Unterbrechungen. Sie treten jedesmal auf, wenn bestimmte Befehle mit denselben Daten ausgeführt werden. Beim Z8000 werden beide Unterbrechungstypen auf ähnliche Weise behandelt.

2.2 Registerstruktur

Der Z8000 ist ein registerorientierter Prozessor, bei dem der Benutzer im allgemeinen frei wählen kann, welches Register er wofür benutzt. In der Tat ist keines (mit Ausnahme des Stapelzeigers) an irgendeinen Befehl gebunden oder unterliegt irgendwelchen Beschränkungen. Engpässe früherer Mikroprozessoren wie Spezialregister, die allein bestimmte Funktionen ausführen können, wurden somit vermieden. Das Programmieren wird effektiv und übersichtlich. Alle 16 der 16-bit-Register R 0...R 15 können als Akkumulator verwendet werden. Alle außer R 0 eignen sich als Indexregister, als Basisregister und als Zeiger für die indirekte Adressierung.

Wie *Bild 2* zeigt, erreicht man eine ungeheuer große Flexibilität durch die einzigartige Anordnung von Registerpaaren, die sich gegenseitig überlappen. Die sechzehn 8-bit-Register (RH 0...RH 7 und RL 0 bis RL 7), die alle als Akkumulatoren zu verwenden sind, überlappen sich mit den ersten acht 16-bit-Registern (R 0...R 7). Die acht 32-bit-Register (RR 0...RR 14) sind Doppelregister, und die vier 64-bit-Register

Stammbaum des neuen Prozessors

Die Entwicklung der Mikroprozessor-Architektur vom ersten 8-bit-Typ zur heutigen 4. Generation verlief stürmisch und dramatisch. Sie war einerseits bestimmt von den technologischen Grenzen, andererseits von den Wünschen der Anwender im Hinblick auf Hardware und Software.

Die Unzulänglichkeiten des ersten 8-bit-µP (8008), der 1971 entwickelt wurde, waren technologischer Natur. Der MOS-Prozeß war relativ neu und gestattete nur eine begrenzte Komplexität. Mikroprozessoren waren außerdem Abfallprodukte von Rechnerentwicklungen, die von Halbleiter-, nicht von Computer-Herstellern stammten. Ihre Eigenschaften ließen deshalb einiges zu wünschen übrig.

Fortschritte im Herstellungsprozeß gaben den Entwicklern ein leistungsfähiges Werkzeug in die Hand, um die nächste Generation in Angriff zu nehmen: Die N-Kanal-Silicon-Gate-MOS-Technik ermöglichte beispielsweise eine um den Faktor vier höhere Geschwindigkeit gegenüber der PMOS-Technik. Damit wurde die 2. Generation (8080) im Jahr 1974 geboren.

Als die 3. Generation im Entwurfsstadium war, waren die Benutzer bereits auf einem höheren Niveau angelangt und befaßten sich auch schon mit höheren Programmiersprachen. Datenverarbeitungsanwendungen wurden immer populärer, und Disketten-Betriebssysteme waren eingeführt. Auf dem Gebiet der Software lagen jetzt die Schwächen, die es auszuräumen galt. Der Z80 bot als Antwort softwareorientierte Eigenschaften. Er kam mit einer Reihe zusätzlicher Befehle und einem zweiten Registersatz, mit zwei Indexregistern und einer verbesserten Interrupt-Behandlung. Trotzdem enthielt er – wegen seiner Quellcode-Kompatibilität mit dem 8080 – immer noch viele Schwachstellen.

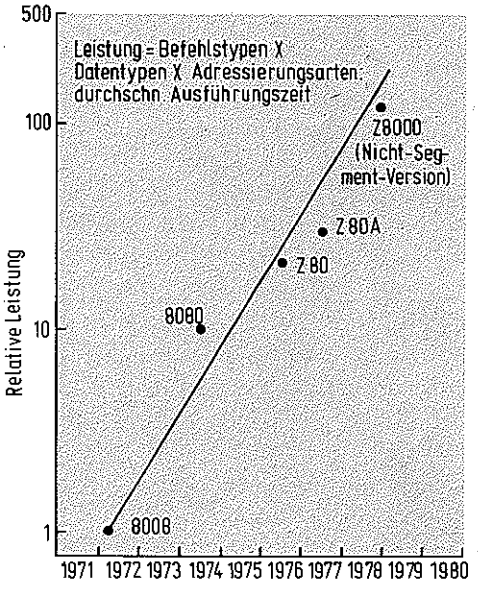
Der Z80 stellt die letzte Stufe der ursprünglichen Mikroprozessor-Architektur und des ursprünglichen Befehlsformats dar. Jeder Versuch, auf dieser Grundlage eine weitere Leistungssteigerung zu erzielen, würde zwei oder drei Befehlshol-Phasen voraussetzen und eine enorm schlechte Ausnutzung von Speicherplatz und -geschwindigkeit zur Folge haben. Überdies ließ die steigende Popularität höherer Programmiersprachen zusammen mit dem Wunsch nach mehr Adreßraum, der von den fallenden Speicherkosten unterstützt wurde, die Möglichkeiten von 8-bit-Prozessoren überholt erscheinen. Die Tendenzen in Richtung größere Programme, komplexere Systeme verteilter Intelligenz und fortschrittliche Speicherverwaltung wiesen alle auf die 16-bit-Architektur. Aber bei Zilog kam man zu dem Schluß, daß ein Chip mit Minicomputereigenschaften ohne 32-bit-Operationen und Speichersegmentierung keine 10 Jahre überdauern könnte. Deshalb entschloß man sich zur jetzigen Z8000-Struktur.

Die *Tabelle* vergleicht Mikroprozessoren verschiedener Generationen. In der Grafik wird die Steigerung der relativen Leistungsfähigkeit dargestellt. Obwohl sie keine absoluten Werte berücksichtigen kann, gibt sie doch einen guten Vergleich, da sie Befehle, Adressierung, Datentypen und Geschwindigkeit – also Software und Hardware – miteinbezieht.

Vergleich charakteristischer Mikroprozessordaten

	8080	Z 80	Z 80 A	Z 8000
Erstmals hergestellt	1974	1976	1977	1978
Verlustleistung (W)	1,2	1,0		1,5
Zahl der Transistoren	4 800	8 200		17 500
Zahl der Gatter	1 600	2 733		5 833
Chip-Größe (mm^2)	22,3	27,1	22,4	39,3
Gatter/mm^2	72	101	122	148
Befehlstypen*	34	52		81
Befehlstypen kombiniert mit Datentypen*	39	60		149
Befehlstypen komb. mit Datentypen u. Adressierungsarten	65	128		414

* Nach einer konservativen Zählmethode. Der Benutzer hat wesentlich mehr Operationscodes.



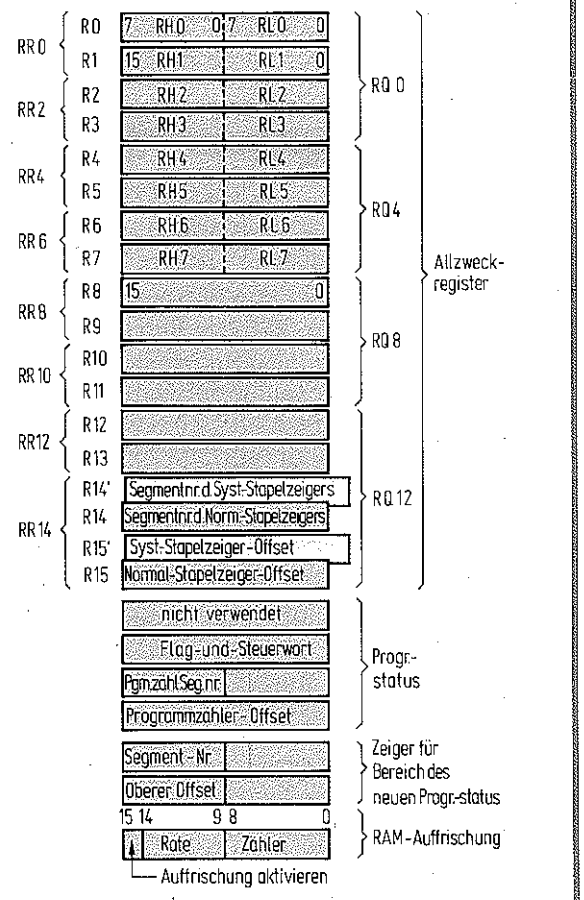


Bild 2. Registerstruktur: Sechzehn 16-bit-Register sind in höher- und niederwertigen Bytes (RH und RL) angeordnet und können als 32-bit- (RR) und 64-bit-Register (RQ) benutzt werden

(RQ 0...RQ 12) sind Vierfachregister, die von einigen Befehlen wie Multiplikation, Division und erweiterten Vorzeichen-Operationen benutzt werden.

In der Nicht-Segment-Version fungiert R 15 als Stapelzeiger, während in der Segment-Version R 14 und R 15 bzw. RR 14 den Stapelzeiger darstellen: R 14 enthält die Segmentnummer, R 15 den Offset. Die einzigen Befehle, die ausschließlich den Stapelzeiger verwenden, sind „Call", „Call relative", „Return" und „Return from Interrupt". Die Befehle „Push" und „Pop" können jedes Register als Stapelzeiger benutzen. Trotzdem ist es allen Befehlen möglich, den Stapelzeiger zu beeinflussen, da er zur Gruppe der Allzweckregister gehört.

Die zwei Betriebsarten des Prozessors – „Normal" und „System" – haben beide einen eigenen Stapelzeiger, wie in Bild 2 durch R 14' und R 15' angedeutet. Obwohl beide Stapelspeicher getrennt sind, kann auf den „Normal"-Stapelzeiger im System-Modus zugegriffen werden (mit dem Befehl „Load Control Word"). Zwei Stapelzeiger erleichtern die Programmumschaltung, wenn Interrupts oder Traps auftreten. Der Stapelspeicher des Normal-Betriebs enthält niemals Information aus dem System-Betrieb, da Daten, die beim Auftreten von Interrupts oder Traps zu retten sind, immer im System-Stapelspeicher abgelegt werden, bevor der neue Programmstatus geladen wird.

Zusätzlich zu den Allzweckregistern sind Programmstatusregister vorhanden, die Zustandsbits (*Flag*), Steuerbits und den Programmzählerstand enthalten. In der 40poligen Version sind das zwei 16-bit-Register, von denen eines als Programmzähler fungiert. Bei der Segment-Version sind im Programmstatus-Bereich vier 16-bit-Register vorgesehen: Flag- und Steuerwort, 2-Wort-Programmzähler und ein Register, das für zukünftige Aufgaben reserviert ist. Ein weiteres Register enthält den Zeiger für den Bereich, in dem der neue Programmstatus operieren wird. Es enthält zwei Worte in der Segment-Version und eines in der Nicht-Segment-Version. Schließlich steht in einem Auffrischregister noch ein 9-bit-Zähler zum automatischen Auffrischen von dynamischen RAMs zur Verfügung.

3 Befehls- und Unterbrechungsstruktur

3.1 Zeitsteuerung

Der Prozessor führt Befehle schrittweise aus, indem er folgende Maschinenzyklen durchläuft: Speicher schreiben oder lesen, E/A-Einheit schreiben oder lesen, Daten intern bearbeiten. Da ein Speicherzyklus drei Taktzyklen dauert (holen oder abspeichern), ist der kürzeste Maschinenbefehl ebenfalls drei Taktzyklen lang. Für komplexe Befehle können allerdings bis zu acht Taktzyklen vergehen, bis sie ausgeführt sind.

Den größtmöglichen Durchsatz würde man erreichen, wenn jeder Befehl in der Zeit eines Speicherzyklus auszuführen wäre. Kein Taktzyklus sollte für andere Phasen verschwendet werden. Zahlreiche Vergleichsprogramme (*Benchmark*) haben gezeigt, daß beim Z8000 die effektive Speicherzykluszeit (auch Busbenutzungszeit oder Buseffektivität genannt) 80...85 % der gesamten Befehlsausführungszeit beträgt. Sie steigt bis zu 90 %, wenn Sprungbefehle ausgeschlossen werden. Dies bedeutet eine erhebliche Verbesserung gegenüber den 65...70 % des Z80.

Ein Grund für den hohen Wirkungsgrad des Z8000 ist sein „Look-ahead"-Befehlsdecodierer, der die Ausführung wesentlich beschleunigt (*Bild 3*). Da er an den internen Bus angeschlossen ist und da der Befehlssatz sehr regelmäßig aufgebaut ist, kann mit der Ausführung eines Befehls bereits begonnen werden, während er noch im Befehlsregister gespeichert ist. Die „Look-ahead"-Logik sorgt für einen erheblich höheren Durchsatz z. B. bei der direkten und indizierten Adressierung (nach der Register-Adressierung am meisten verwendet). Hier benötigt der Prozessor keine zusätzlichen Taktzyklen, um zu erkennen, ob es sich um einen kurzen oder langen Offset handelt. Die Register-Register-Ladebefehle wurden so optimiert, daß sie nur die drei Taktzyklen ihres Speicherzugriffs brauchen. In den meisten Befehlen deckt sich die Datenbearbeitungszeit vollständig mit der Hol-Phase für das erste Wort des nächsten Befehls.

3.2 Einwort-Codierung

In der gesamten Konzeption wurde peinlich genau darauf geachtet, daß die Befehle danach optimiert und beschleunigt wurden, wie wichtig sie statistisch gese-

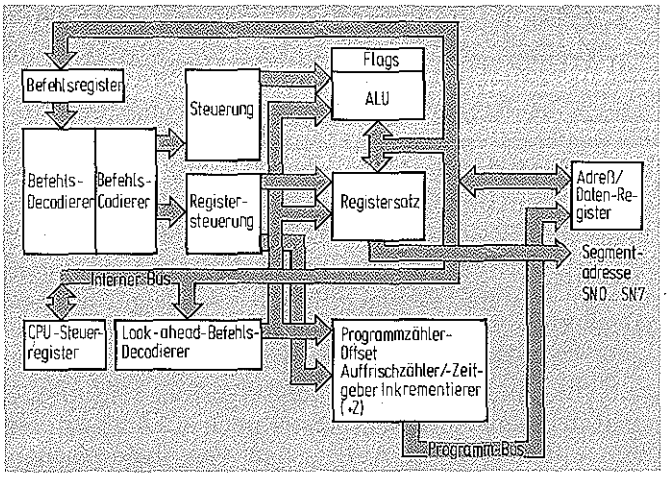


Bild 3. Interner Aufbau

hen sind. Manche wurden in einem einzigen Wort so ausgerichtet, daß sie die Ausführung beschleunigen, die Logik vereinfachen und einen großen Bereich für die relative Adressierung ermöglichen. Um die Ausführung noch schneller zu machen und noch weniger Programmspeicher zu belegen, wurden die am häufigsten benutzten Befehle in einem Wort codiert. Darunter sind: relative Sprünge, dekrementieren und springen falls ungleich null, Byte unmittelbar (*immediate*) laden, Wort unmittelbar laden und relative Unterprogrammsprünge. Bemerkenswert sind die festprogrammierten Block- und String-Befehle, die durch Interrupts unterbrechbar sind und Speicher-zu-Speicher-Transfer mit einer Geschwindigkeit von 888 000 Byte/s ausführen.

3.3 Sonderbefehle

Eine Anzahl von leistungsfähigen Befehlen des Z8000 findet man in früheren Mikroprozessormodellen nicht. Das sind z. B. 32-bit-Multiplikation und -Division und andere Instruktionen, die Mehrfachworte laden und abspeichern. Und es sind Befehle vorhanden, die den Inhalt beliebiger Register oder Speicherplätze um eine Zahl von 1...16 vermindern oder erhöhen. Schließlich erhöhen mehrere Adressierungsarten für Push- und Pop-, Lade- und Speicherbefehle die Leistungsfähigkeit.

3.4 Format

Ein wichtiger Faktor beim Mikroprozessor ist sein Befehlsformat. Die Komplexität der Logik und damit die Chip-Größe hängen davon ab. Als ideal muß man es bezeichnen, wenn eine „totale Regelmäßigkeit" erreicht wird, d. h. wenn jeder Befehl alle Datentypen und alle Adressierungsarten benutzen kann. Das ist beim Z8000 zum großen Teil gelungen.

Von den acht wählbaren Adressierungsarten können die fünf ersten (*Tabelle*) für nahezu alle Befehle verwendet werden; Ausnahmen bilden einige Rotier- und Schiebebefehle. Die drei restlichen lassen sich auf alle Lade- und Speicherbefehle anwenden. Um Speicherplatz zu sparen, ist die relative Adressierung zusätzlich bei Sprüngen, Unterprogrammaufrufen sowie dekrementieren/springen falls ungleich null (*decrement and jump on non-zero*) möglich. Einige Befehle führen automatisch Inkrementierung und Dekrementierung durch. Alle acht Adressierungsarten kennt der Lade-Adreß-Befehl, der selbst den kompliziertesten Operanden-Adressier-Ablauf unterstützt.

Das Befehlsformat zeigt *Bild 4*. Die zwei höchstwertigen Bits zeigen an, ob das gepackte (*tight*) oder das allgemeine (*general*) Format benutzt wird. Erstere benutzen 1-Wort-Befehle, sie belegen weniger Speicher und werden schneller ausgeführt. Wenn die höchstwertigen Bits nicht beide „1" sind, handelt es sich um das ungepackte Format. Sie bestimmen – zusammen mit dem Quellregister-Feld des Befehls –, welche der fünf Haupt-Adressierungsarten ausgeführt wird. Wie Bild 4b zeigt, unterscheidet der Quellwert Null „unmittelbar" (*immediate*) und „direkt" von „indirekt" und „indiziert", die beide ein Quellregister benötigen. Die Quell- und Zielregister-Felder sind 4 bit lang, um die 16 Allzweckregister adressieren zu können.

Der Z8000 hat keine Befehle für Speicher-zu-Speicher-Arithmetik. Allerdings führt er Speicher-zu-Speicher-Transfers mit Hilfe komplexer, vorprogrammierter Block-Transfer- und String-Bearbeitungs-Befehle durch, und er bietet die Befehle „Speichern", „Push" und „Vergleichen" in der unmittelbaren Adressierungsart. Dies läßt ein kompakteres Befehlsformat mit mehr Operationscodes für zusätzliche

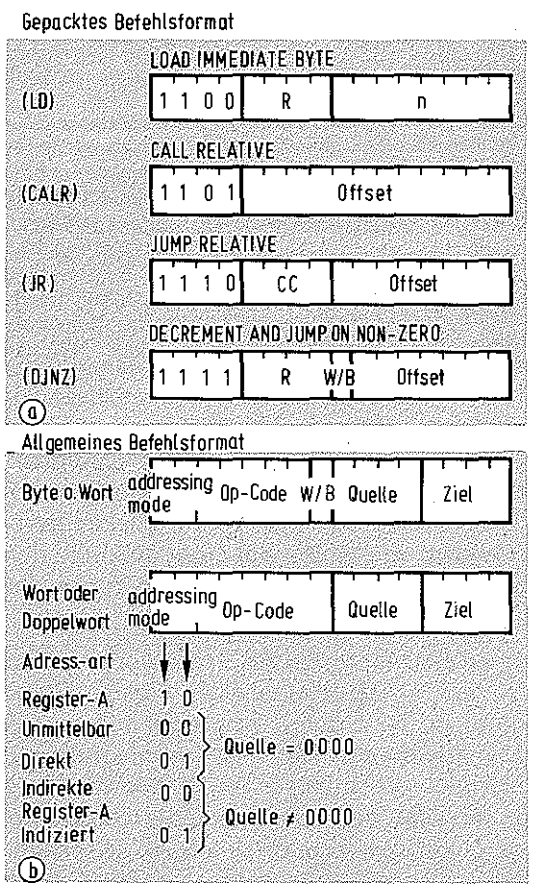


Bild 4. Das gepackte Format (a) benutzt nur 16 bit, wird schnell ausgeführt und spart Speicherplatz. Das allgemeine Format, das für Bytes, Worte und Doppelworte verwendet wird, gibt die Adressierungsart und das Quellregister an

Instruktionen zu als eine allgemeine Speicher-zu-Speicher-Adressierung, wie sie etwa im Modell PDP-11 (Digital Equipment) angewendet wird. Diese Maschine hat zwei Sätze von Adressierungsarten und Registerfeldern.

3.5 *Unterbrechungen*

Die sieben Unterbrechungsmöglichkeiten (interne wie externe) sind nach Prioritäten geordnet. Die drei Interrupts, der nicht-maskierbare, der vektorierte und der nicht-vektorierte werden extern ausgelöst (die letzten beiden sind maskierbar). Von den vier Traps ist der einzig externe der Segmenteingang, der nur in der Segment-Version vorhanden ist. Die restlichen drei werden dann ausgelöst, wenn bestimmte Befehle, die auf den System-Modus beschränkt sind, im Normal-Modus verwendet werden, wenn der System-Modus aufgerufen wird oder wenn ein ungültiger Befehl im Programm erscheint. Nach fallender Priorität geordnet, ergibt sich die Reihenfolge: interne Traps, nicht-maskierbarer Interrupt, Segment-Trap, vektorierter und nicht-vektorierter Interrupt.

Wenn ein Interrupt oder Trap auftritt, wird der Programmstatus, der in zwei (Nicht-Segment-Version) oder drei 16-bit-Worten (Segment-Version) enthalten ist, im System-Stack abgelegt – gefolgt von einem zusätzlichen Wort (*Reason Word*). Das im Normalfall über den Unterbrechungsgrund Auskunft gibt. Im Falle eines internen Trap ist dieses zusätzliche Wort das erste Wort des unterbrochenen Befehls. Beim

Tabelle der Adressierungsarten

Adressiergs-art	Diagramm	Operand
Register *	(Register) Befehl → Operand	Registerinhalt
Indirekte Register-A *	(Register) Befehl → Adresse → Operand	Inhalt der Speicherstelle, d. Adr. im Register steht
Direkt * (direct address)	Befehl; Adresse → Operand	Inhalt der Speicherstelle, deren Adr. im Befehl steht
Unmittelbar * (immediate)	Befehl; Operand	Teil des Befehls
Indiziert *	(Register) Befehl → Versatz; Adresse + → Operand	Inhalt der Speicherstelle, deren Adr. sich aus einem Teil des Befehls und dem Offset im Register ergibt
Relativ (relative address)	(Programmzähler) Befehl → Adresse; Versatz + → Operand	Inhalt der Speicherstelle, deren Adr. sich aus Progr.-zählerstand und dem Versatz im Befehl ergibt
Basis-A. (base address)	(Register) Befehl → Adresse; Versatz + → Operand	Inhalt der Speicherstelle, deren Adr. sich aus Registerinhalt und Versatz im Befehl ergibt
Basis-indiziert (base index)	(Register) Befehl → Adresse; (Register) Versatz + → Operand	Inhalt der Speicherstelle, deren Adr. sich aus der Adresse in einem Register und dem Versatz in einem anderen Register ergibt

*Haupt-Adressierungsarten

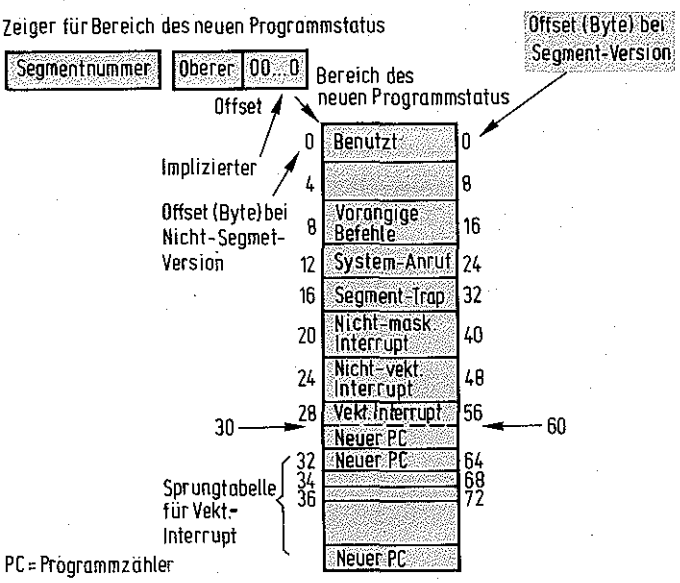


Bild 5. Der Zeiger für den Bereich des neuen Programmstatus ist zwei Worte lang; die sieben höherwertigen Bits im zweiten Wort geben den Beginn eines Bereiches an, von dem der neue Programmstatus geholt wird, nachdem ein Interrupt oder Trap aufgetreten ist

Segment-Trap und allen Interrupts ist es der Vektor auf dem Datenbus, der von der CPU während des Bestätigungs-Maschinenzyklus gelesen wird.

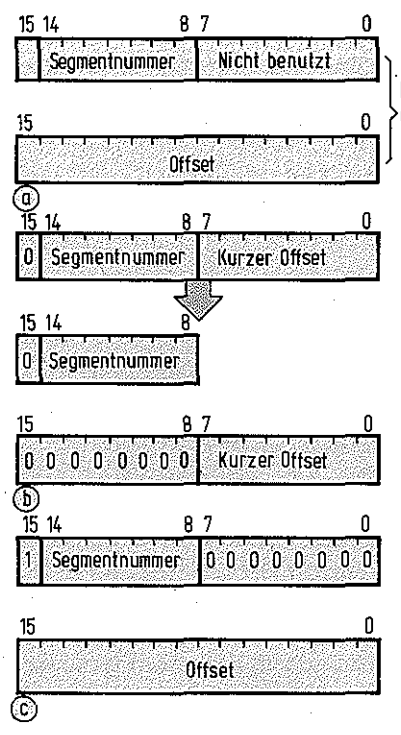
Da somit der vorhergehende Programmstatus im System-Stack abgelegt ist, muß ein neuer von der Tabelle „Neuer Programmstatus" (*Bild 5*) geholt werden. Die Lage dieser Tabelle wird von einem Zeiger (*New-Program-Status Area Pointer*) in der CPU (Bild 2) bestimmt, und zwar vom höherwertigen Byte des entsprechenden Registers bei der Nicht-Segment-Version. Bei der Segment-Version geben sieben Bits zusätzlich die Segmentnummer an. Nachdem Interrupt oder Trap beendet sind, tritt der Prozessor in eine Reset-Sequenz ein. Ein neuer Programmstatus wird dann von einer festen Speicherstelle im Segment 0 geholt.

3.6 *Vorrangige Befehle*

Um eine klare Trennung von Betriebssystem und Anwenderprogramm zu erreichen, wurden die beiden Betriebsarten „System" und „Normal" geschaffen. Die Unterscheidung ergibt sich aus den vorrangigen Befehlen, die nur im System-Modus ausgeführt werden, während sie einen Trap auslösen, wenn sie im „normalen" Programmfluß erscheinen. Folgende Befehle gehören zu dieser Gruppe: E/A-Befehle, Halt, Interrupt unterdrücken/ermöglichen (*disable/enable*), Lade Steuerwort, Lade neuen Programmstatus, Rückkehr vom Interrupt und alle Mehr-Prozessor-Befehle.

4 Adressierung

Höhere Programmiersprachen, ausgefeilte Betriebssysteme, große Programme und Datenmengen sowie fallende Speicherpreise beschleunigen den Trend, Mikrocomputersysteme mit immer größeren Speichern zu bauen. Immer wichtiger werden dabei die Fragen: Welche Methode des Speicherzugriffs ist die beste und wie soll der Benutzer den Speicher verwalten? Die Antwort des Z8000 ist die Segment-Adressie-



◀ Bild 6. Segmentierte Adressen erscheinen als Doppelworte (32 bit), falls sie in Registern oder im Speicher stehen (a). In Befehlen können sie Worte (b) oder Doppelworte (c) sein

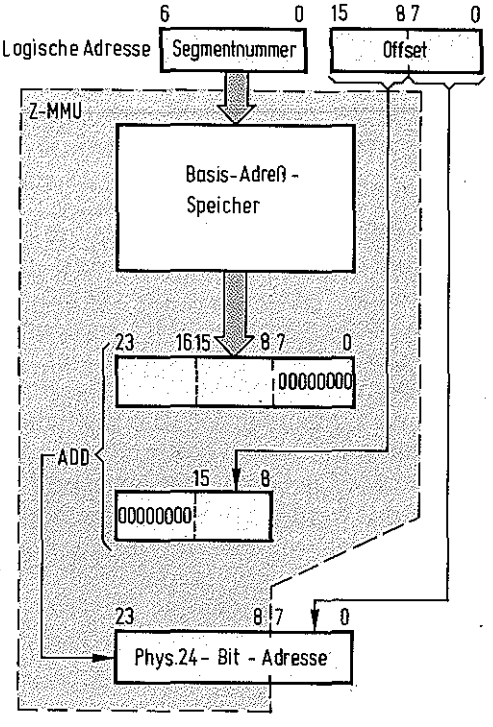


Bild 7. ▶ Der Speicherverwaltungs-Baustein wandelt die logische in die physikalische Adresse um, indem er zur 24-bit-Basis, die zu jedem Segment gehört, den 16-bit-Offset addiert

rung. Die Segmentnummer ist eine 7-bit-Ganzzahl ohne Vorzeichen (0...127); der Offset ist eine 16-bit-Ganzzahl zwischen 0 und 65 535.

In einem Register belegt eine segmentierte Adresse immer ein Registerpaar (*Bild 6*) oder Doppelwort (*Long Word*, 32 bit). Die beiden Worte können getrennt oder gemeinsam durch jede Wort- oder Doppelwortregister-Operation beeinflußt werden. Auch im Speicher stehen alle segmentierten Adressen als Doppelworte. In Befehlen jedoch haben segmentierte Adressen zwei verschiedene Formate: mit kurzem Offset (Bild 6b) oder mit langem Offset (Bild 6c). Ein kurzer Offset kann immer dann benutzt werden, wenn die Adresse innerhalb der ersten 256 Speicherstellen eines Segments liegt, da die acht höherwertigen Bits des Offset null sind. Dieses Format erlaubt eine sehr kompakte Adreß-Codierung und eignet sich nicht nur für die indizierte Adressierung, sondern auch für die direkte Adressierung, wenn kurze Datenbereiche benutzt werden oder wenn Unterprogramme am Anfang eines Segments beginnen.

4.1 Speicherverwaltungs-Baustein

Solche Adressen, die vom Programmierer manipuliert, von Befehlen benutzt und an die Ausgänge des Prozessors gegeben werden, nennt man logische Adressen. Diese logischen Adressen, die ja aus der Verknüpfung von Segmentnummer und Offset bestehen, in physikalische 24-bit-Adressen zu verwandeln, ist die Aufgabe des Speicherverwaltungs-Bausteins Z-MMU (*Memory-Management-Unit*, siehe Kasten S. 84). *Bild 7* zeigt das Prinzip seiner Arbeitsweise (*Relocation*): Ein 24-bit-Ursprung (Basis) steht im logischen Zusammenhang mit jedem Segment. Um die physikalische 24-bit-Adresse zu bilden, addiert der Z-MMU den 16-bit-Offset zur Basis des gewünschten Segments (in der Praxis sendet der Z8000 die Segmentnummer eine halbe Taktperiode vor dem 16-bit-Offset aus). Damit kann der Prozessor 8 MByte direkt adressieren. Zusätzlich übernimmt der Baustein Aufgaben der Speicherverwaltung und schützt bestimmte Bereiche vor unbeabsichtigtem Überschreiben. Jeder Z-MMU speichert 64 Segment-Eintrittsdaten, die aus der Basis-Adresse sowie Status und Segmentgröße bestehen. Segmente können in ihrer Größe zwischen 256 Byte und 64 KByte in Schritten von 256 Byte variiert werden. Zwei Z-MMUs decken den gesamten Bereich der vorhandenen 128 Segmente ab. Trotzdem können mehrere in einem System benutzt werden, um verschiedene Übersetzungstabellen aufzubauen. Allerdings sind zur selben Zeit maximal zwei aktiv.

Literatur

Die Redaktion empfiehlt zum Nachschlagen:

[1] *Dorn, L.*: HMOS-Prozeß führt zu leistungsfähigem 16-bit-Mikroprozessor. ELEKTRONIK 1979, H. 5, S. 57...63.

Masatoshi Shima wurde in Shizuoka, Japan, geboren. Während er noch bei der Busicom Corp. in seinem Heimatland beschäftigt war, war er bereits an der Entwicklung des ersten Mikroprozessors 4004 (Intel) beteiligt. Später wechselte er dann zur Firma Intel über, bei der er am Entwurf und an der Entwicklung des 8080 beteiligt war. Bei seinem jetzigen Arbeitgeber Zilog war er für die Entwicklung der Z80-Familie und des Z8000 verantwortlich.

Klaus Piotter

I^3L ist eine Weiterentwicklung des I^2L-Prozesses [1]. In dieser bipolaren Technik wird einer der neuen 16-bit-Mikroprozessoren hergestellt. Seine Besonderheit: Er arbeitet mit dem Befehlssatz eines bekannten Minicomputers.

16-bit-I^3L-Prozessor mit Minicomputer-Befehlssatz

Es handelt sich beim Mikroprozessor 9440 in der Tat um eine Minicomputer-Zentraleinheit. Daten und Befehle sind in einem externen Speicher abgelegt. Über einen 16-bit-Bus werden Daten und Adressen zwischen der Zentraleinheit und den anderen Einheiten des Computers ausgetauscht. Obwohl der Prozessor 16 bit parallel verarbeitet, werden für die Adressierung des Speichers nur 15 bit verwendet. Dies ergibt einen direkt adressierbaren Speicherbereich von 32 768 Worten zu 16 bit.

An die Ein/Ausgabe-Ports können bis zu 63 periphere Einheiten angeschlossen werden, wobei die Ein/Ausgabe-Operation über das Programm, durch Programmunterbrechung (*Interrupt*) oder DMA (*Direct Memory Access*) ausgelöst werden kann. Bei der programmgesteuerten Ein/Ausgabe kann der Prozessor ein 16-bit-Datenwort zu oder von einer peripheren Einheit übertragen, gleichzeitig überwacht er die Ausführung dieser Operation.

Das Interrupt-System erlaubt es jedem peripheren Gerät, bei entsprechender Priorität den normalen Programmablauf zu unterbrechen. Erfolgt ein Interrupt, so wird der momentane Programmzählerstand gesichert und ein Sprung in die Interrupt-Service-Routine ausgeführt.

Schnelle periphere Einheiten, wie Magnetband oder -platte, können über einen Datenkanal direkt auf den Hauptspeicher zugreifen, ohne daß zusätzliche Instruktionen notwendig sind. Die Übertragungsrate bei direktem Speicherzugriff (DMA) wird nur durch die Geschwindigkeit des Speichers begrenzt.

Obwohl in der Struktur unterschiedlich zu der Reihe der NOVA-Minicomputer (Data General), bietet der 9440 vergleichbare Leistung und den gleichen Befehlssatz. Um ihn voll nutzen zu können, wurde das Softwarepaket FIRE (*Fairchild Integrated Real-Time Executive*) geschaffen. Es besteht aus allen für die Programmentwicklung notwendigen Hilfen wie Editor, Assembler, Ladeprogrammen, Debugger (Programm zur Erkennung und Beseitigung von Fehlern), einem kompletten Satz von Diagnose-Programmen und Compilern für BASIC und FORTRAN.

1 Register und Datenwege

Der Mikroprozessor enthält zwei Funktionsblöcke (*Bild 1*): Der eine enthält die Register, der andere steuert den Datenfluß zwischen den Registern. Insgesamt sind vier allgemeine Register (die Akkumulatoren AC0...AC3) vorhanden, zwei Multiplexer, eine arithmetisch/logische Einheit (ALU) und vier spezielle Register wie Scratch-Register (Zwischenspeicherregister), Busregister, Befehlsregister und Programmzähler. Der interne Datenfluß läuft über einen 4-bit-Datenweg. Die Akkumulatoren speichern die Operanden für alle arithmetisch/logischen Operationen. AC2 und AC3 werden auch als Indexregister verwendet. AC3

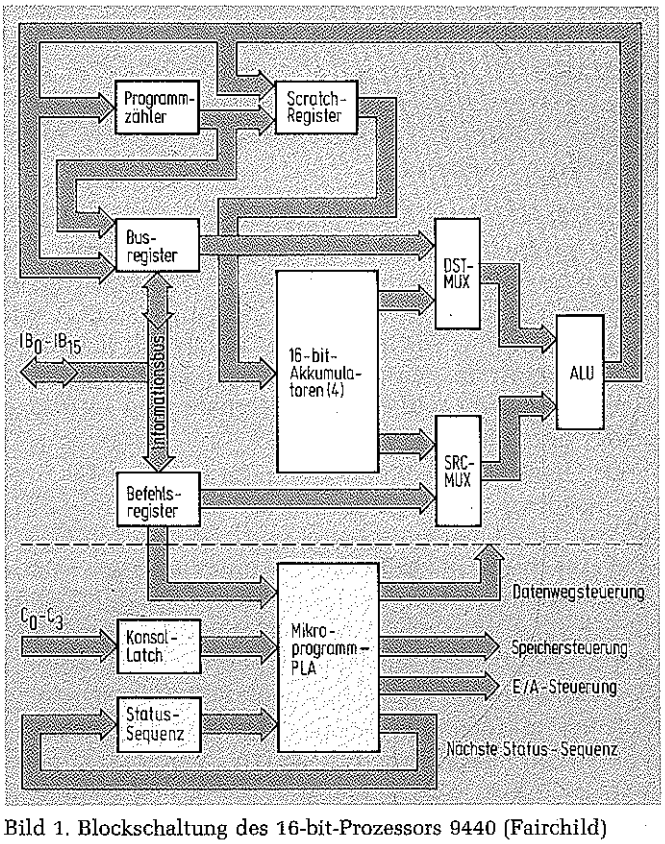


Bild 1. Blockschaltung des 16-bit-Prozessors 9440 (Fairchild)

dient zusätzlich als Auffangregister für die Rückkehradresse beim Aufruf von Unterprogrammen. Alle Ein- oder Ausgabe-Operationen werden über die Akkumulatoren durchgeführt; der Inhalt einer Speicherstelle kann jedoch ohne Operation mit einem Akkumulator entweder inkrementiert oder dekrementiert werden. Zwischen dem Speicher und den Akkumulatoren können die Daten in jeder Richtung bewegt werden.

Die Datenquellen(*Source*)- und Datensenken(*Destination*)-Multiplexer sind mit allen vier Akkumulatoren verbunden und wählen die Quellen- und Senkenregister für jede einzelne Operation aus. Diese Multiplexer erhalten auch Signale vom Bus- und Befehlsregister (*Instruction Register*), dadurch kann die ALU (*Arithmetic Logic Unit*) zur Berechnung der effektiven Adresse und weiteren Aufgaben herangezogen werden.

Die 4-bit-ALU arbeitet zwei 16-bit-Worte in vier aufeinanderfolgenden Schritten ab, wobei pro Schritt ein 4-bit-Abschnitt des Wortes verarbeitet wird. Durch Hinzufügen des zugehörigen Übertrages (*Carry Bit*) zu den 16 Bits des Ergebnisses der ALU entsteht ein 17-bit-Wort, das nach links oder rechts verschoben werden kann.

Daten, die von der ALU zum Bestimmungsakkumulator übertragen werden, werden für einen Zyklus im Scratch-Register abgespeichert. Das Busregister ist an den bidirektionalen Informationsbus angeschlossen und kann 16 bit parallel ausgeben oder einlesen. Das Befehlsregister wird während eines Befehlsabrufs (*Instruction-Fetch Operation*) direkt vom Informationsbus mit 16 bit parallel geladen. Der 15-bit-Programmzähler bestimmt die Reihenfolge, in der die Befehle ausgeführt werden. Er wird zur Anwahl der nächstfolgenden Speicherstelle fortlaufend erhöht. Die Befehlsfolge kann jedoch jederzeit durch Änderung des Programmzählerinhaltes (Sprungbefehl) oder durch zweimaliges Inkrementieren (Skip-Befehl) verändert werden.

2 Interne Steuerung

Die Signale von der Steuereinheit werden von einem maskenprogrammierten PLA ausgegeben. Für jede einzelne der 72 unterschiedlichen Operationen auf den Datenwegen steht am Ausgang des PLA ein 24-bit-Wort zur Verfügung, das durch die jeweilige Kombination an den 19 PLA-Eingängen ausgewählt wird. Eingangs- und Ausgangssignale sind in *Bild 2* näher bezeichnet.

Der Baustein arbeitet mit seinem internen Oszillator, wenn an die Anschlüsse CP (*Clock Pulse*) und XTL (*Cristal*) ein Quarz geschaltet wird. Ebenso kann über den Anschluß CP ein Taktsignal eines externen Oszillators eingespeist werden. Das interne Taktsignal, ein Synchronisationssignal und verschiedene Signale für die Steuerung des zeitlichen Ablaufs werden von einer Taktlogik auf dem Chip generiert.

Jeder Makrobefehl wird als Folge verschiedener Mikrobefehle ausgeführt. Ein Mikro-Zyklus wiederum setzt sich aus verschiedenen Phasen, den Nano-Zyklen, zusammen, von denen vier die Datenwege steuern. Während jeder dieser Phasen werden 4 bit des 16-bit-Operanden verarbeitet. Die einzelnen Register werden mit der abfallenden Flanke des Taktsignals getaktet. Eine weitere Phase sorgt für eine Verzögerung, um die Zeit vom Anlegen neuer Eingangssignale an das PLA bis zum stabilen Zustand der PLA-Ausgänge sicherzustellen.

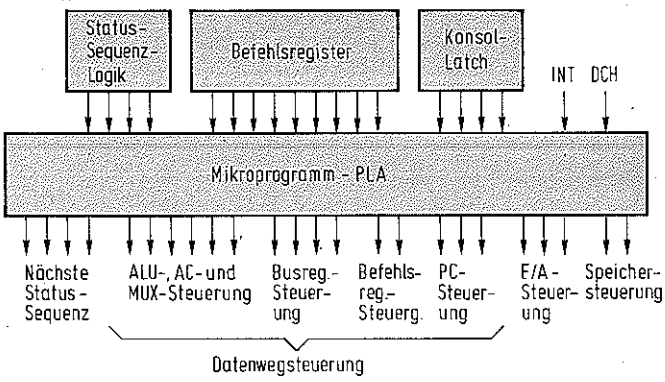


Bild 2. Ein- und Ausgänge des Mikroprogramm-PLA

Während eines Ausführungszyklus werden das Busregister und das Befehlsregister zu verschiedenen Zeiten geladen. Die Taktsignale für das Busregister (BRPCL = *Bus Register Parallel Clock*) und das Befehlsregister (IRPCL = *Instruction Register Parallel Clock*) sind in ihrem zeitlichen Verlauf in den *Bildern 3* und *4* dargestellt.

3 Adressierungsarten

Der Mikroprozessor kennt vier Grundadressierungsarten: Bei der absoluten Adressierung werden die ersten 256 Speicherplätze direkt adressiert. Bei den restlichen drei wird über die Register AC2 und AC3 sowie den Programmzähler PC relativ adressiert. Eine 8-bit-Relativadresse mit Vorzeichenerkennung wird zu einer 15-bit-Basisadresse in einem Indexregister (AC2, AC3 oder PC) addiert. Jede dieser Adressierungsarten kann direkt oder indirekt ausgeführt werden, dadurch ergibt sich eine Gesamtzahl von acht Möglichkeiten.

Bei der direkten Adressierung ist das errechnete 15-bit-Wort die direkt angesprochene Adresse, deren Operand entweder von dieser Adresse gelesen oder in die Adresse geschrieben wird. Bei der indirekten Adressierung handelt es sich um die Adresse einer Adresse. Die aus dem indirekt adressierten Speicherplatz ausgelesene Adresse kann die Endadresse oder auch eine weitere indirekte Adresse sein – dies hängt von dem höchstwertigen Bit in dem Wort ab.

4 Ein/Ausgabe-Steuerung

Ein/Ausgabe-Einheiten werden durch Programm- oder Interrupt-Steuerung angesprochen. Über den Datenkanal können Hochgeschwindigkeitseinheiten auch direkt zum Hauptspeicher zugreifen. Um Überschneidungen auf dem Informationsbus zu vermeiden, werden den Ein/Ausgabe-Einheiten Synchronisationssignale übermittelt. Über die Codierung der O-Leitungen können vier Überwachungsfunktionen ausgeführt werden (*Tabelle 1*).

Tabelle 1. Codierung der O-Leitungen

O_1	O_0	Funktion
0	0	Befehlsabruf
0	1	Rückmeldung Datenkanal
1	0	E/A-Ausführung
1	1	keine Ausführung

Tabelle 2. Codierung der C-Leitungen

C_3	C_2	C_1	C_0	Operation
0	0	0	0	AC0 prüfen
0	0	0	1	AC1 prüfen
0	0	1	0	AC2 prüfen
0	0	1	1	AC3 prüfen
0	1	0	0	Programmzähler laden
0	1	0	1	Speicherplatz prüfen
0	1	1	0	nicht benutzt
0	1	1	1	Halt
1	0	0	0	AC0 ändern
1	0	0	1	AC1 ändern
1	0	1	0	AC2 ändern
1	0	1	1	AC3 ändern
1	1	0	0	Programmzähler laden
1	1	0	1	Speicherplatz ändern
1	1	1	0	Programmfortführung/Start
1	1	1	1	keine Ausführung

Beim Betrieb der E/A-Einheiten unter Programmsteuerung liefert die CPU auf den Leitungen IB_0 und IB_1 zwei Statussignale: „Eingabe bereit" und „Eingabe gesendet". Da mehrere Einheiten gleichzeitig einen Interrupt erzeugen können, werden sie nach einer Prioritätenliste abgearbeitet. Die Zentraleinheit kann allerdings durch Programmierung der Interrupt-Maske das Interrupt-System ganz oder teilweise abschalten.

Für den direkten Speicherzugriff dienen drei Steuerleitungen. Die Anforderungsleitung für den Datenkanal steht allen Ein/Ausgabe-Einheiten zur Verfügung. Die Priorität wird ähnlich wie bei der Interrupt-Steuerung dadurch festgelegt, daß der Prioritätseingang des Datenkanals mit den Ausgabeleitungen der E/A-Geräte verbunden wird.

Die Bedienkonsole wird als spezielles Ein/Ausgabe-Gerät angesehen. Der Bediener kann über die vier C-Leitungen und die Rücksetzleitung verfügen. Die drei Statusleitungen des Bausteins 9440 sind unmittelbar mit der Bedienkonsole verbunden. Damit lassen sich Daten auf jeden der vier Akkumulatoren und auf den Programmzähler übertragen (*Tabelle 2*). Außerdem kann von jedem dieser Register der Inhalt abgefragt werden. Des weiteren kann der Inhalt eines vom Programmzähler adressierten Speicherplatzes abgefragt oder geändert werden. Man kann auch von der Konsole aus ein Programm starten oder stoppen.

5 Weitere Bausteine

Eine Erweiterung der Zentraleinheit 9440 ist der Baustein 9445. Er weist folgende Änderungen auf: 16-bit-Datenwege, hardwaremäßige Verdrahtung der arithmetischen Operationen Multiplikation und Division, Stack-Funktionen, Erhöhung der Geschwindigkeit um das Vierfache, Befehlssatz NOVA-4-kompatibel.

Folgende Peripheriebausteine stehen zur Verfügung (*Bild 5*): die bipolaren dynamischen RAMs

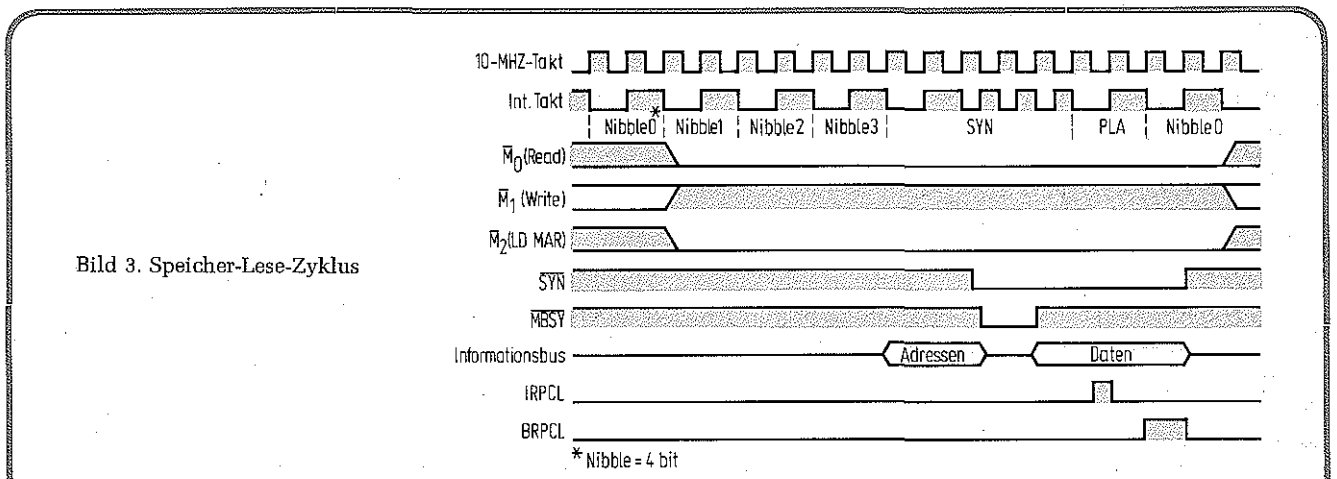


Bild 3. Speicher-Lese-Zyklus

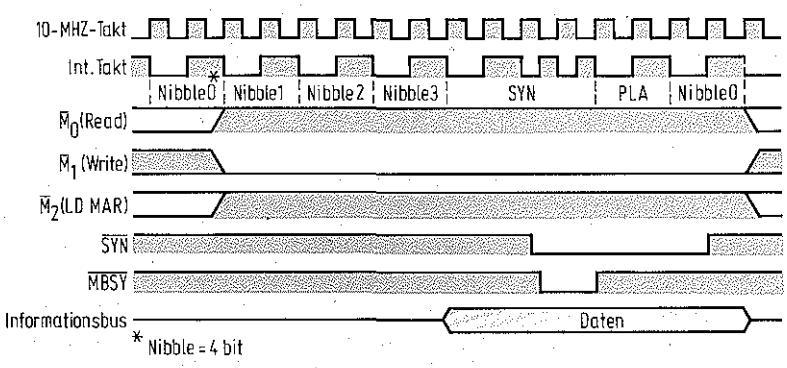


Bild 4. Speicher-Schreib-Zyklus

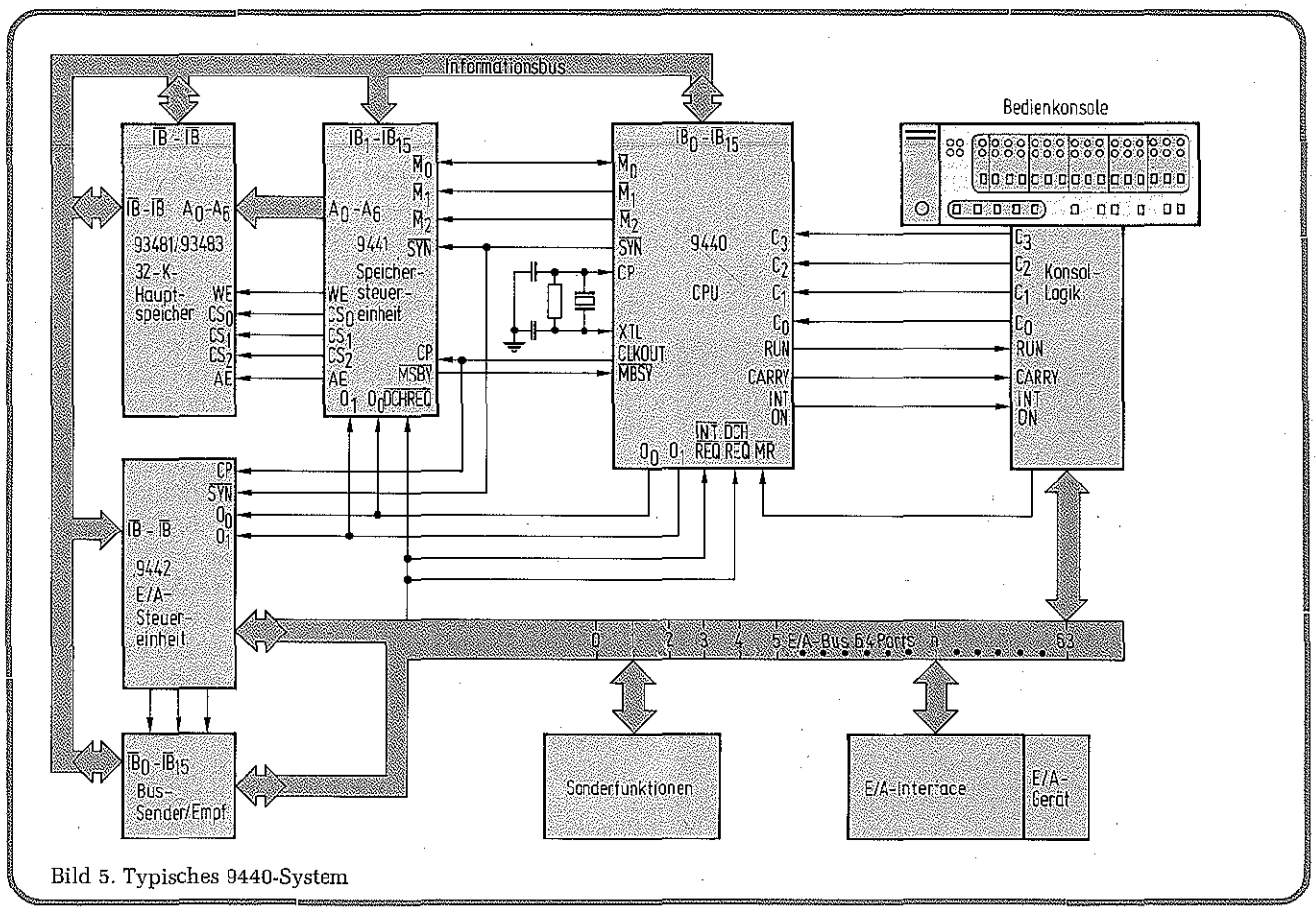


Bild 5. Typisches 9440-System

93481 und 93483, eine Speichersteuereinheit (9441) und eine Bussteuereinheit (9442).

Das 4-K- und das 16-K-RAM sind dynamische I^3L-Speicher, die mit nur einer Spannungsversorgung von +5 V auskommen, am Eingang TTL-kompatibel sind und einen Tristate-Ausgang mit einem „Fan-Out" von 10 haben. Sie haben eine typische Zugriffszeit von 100 ns, eine Verlustleistung von 400 mW im Betrieb und 70 mW im nichtaktivierten Zustand. Die Zykluszeit beträgt 240 ns, wobei in der speziellen Betriebsart „Page-Mode" sowohl die Zugriffs- als auch die Zykluszeit 65 ns beträgt.

Die Speichersteuereinheit 9441 erleichtert den Anschluß der bipolaren dynamischen RAMs an die Zentraleinheit. Sie enthält einen Adressenzähler für das Wiederauffrischen (*Refresh*) der Speicher und ein Speicheradreßregister (15 bit), dessen Ausgänge im Multiplexverfahren mit den Speicheradressen korrespondieren. Interne Zähler sorgen für die Adressenfreigabe, die Freigabe zum Schreiben und die Multiplexzeit zum Ansteuern der Spalten- und Zeilenadresse der Halbleiterspeicher. Ein interner Reihenfolgegenerator sorgt für das richtige Zeitverhalten der Ansteuerung und für die Erzeugung von Interface-Signalen für den direkten Speicherzugriff.

Die Bussteuereinheit 9442 dient zur Erweiterung des Ein/Ausgabe-Busses der Zentraleinheit. Mit ihrer Hilfe lassen sich periphere Einheiten wie Fernschreiber, Bildschirmeinheiten, Magnetbänder, Drucker, Floppy-Disk-Einheiten, Speichermoduln und Geräte zur Echtzeit-Verarbeitung anschließen. Ein/Ausgabe-Befehle werden gespeichert und decodiert. Die zeitliche Steuerung des Datenkanals wird vom Baustein ebenfalls übernommen. Die Steuersignale sind so ausgelegt, daß sie die Datenein- oder -ausgabe von irgendeinem der drei Ein/Ausgabe-Ports in einer peripheren Einheit anwählen. Diese Signale in Verbindung mit Start-, Lösch- und Ein/Ausgabe-Pulsen werden zur Anwahl eines speziellen Gerätes benutzt. Kommen die Ein/Ausgabe-Steuerbefehle von der Zentraleinheit, überträgt der Baustein die zugehörigen Signale über Interrupt-Betätigung, Ein/Ausgabe-Rücksetzen oder durch Maskensignale am Ausgang an die einzelnen peripheren Einheiten. Des weiteren kann der Baustein dazu verwendet werden, den Ein/Ausgabe-Bus der Zentraleinheit an einen zentralen Ein/Ausgabe-Bus anzuschließen.

Literatur

[1] *Weiß, M.:* I^3L: Eine Weiterentwicklung der I^2L-Technik. ELEKTRONIK 1979, H. 13.

[2] Mikroelektronik: Kein Ende der schnellen Weiterentwicklung. ELEKTRONIK-Gespräch mit *Dr. Tom Longo*. H. 1/79, S. 16.

Klaus Piotter wurde in Lauenburg/Pommern geboren. Er ist seit 1970 auf dem Gebiet der Digitaltechnik tätig. Seit 1976 befaßt er sich mit Mikroprozessoren. Anfang 1978 trat er in die Fairchild C & I Deutschland ein.
Sein Aufgabengebiet: die Hardware und Software von Mikroprozessor-Entwicklungssystemen.
Hobbys: Schießsport, Schach
Telefon: 089/32003222

Dipl.-Ing.
Horst Wältring

Für die Echtzeit-Datenverarbeitung stehen bei den meisten Mikroprozessoren Eingänge zur Programmunterbrechung (Interrupt) zur Verfügung. Will man in einem System mehr als eine unterbrechende Peripherieeinheit zulassen, ist es notwendig, unter mehreren gleichzeitig anstehenden Interrupt-Signalen Prioritäten zu setzen, da nicht alle anfordernden Einheiten gleichzeitig bedient werden können. Am Beispiel des programmierbaren Interrupt-Controllers 8259, der diese Aufgabe übernimmt, soll aufgezeigt werden, wie ein solcher Baustein arbeitet und welche Vielfalt an Möglichkeiten er bietet.

Der Interrupt-Controller – ein Baustein, der für Ordnung sorgt

1 Aufgabe

Im Rahmen eines Mikrocomputer-Systems (z. B. SBC 80/20) übernimmt der Interrupt-Controller die Aufgabe, entsprechend einem vom Programmierer festgelegten Modus, eingehende Unterbrechungs-Anforderungen zu speichern, die jeweiligen Prioritäten festzustellen und ein der unterbrechenden Einheit zugeordnetes Bedienprogramm aufzurufen (vektorieller Interrupt, CALL [4]).

2 Aufbau

2.1 Blockschaltung

Wie in der Blockschaltung (*Bild 1*) ersichtlich, gelangen die acht Interrupt-Anforderungs-Leitungen IR0...IR7 auf das Unterbrechungs-Anforderungs-Register (*Interrupt Request Register*, IRR). Hier werden die von außen kommenden Anforderungen gespeichert. Der Prioritäts-Analysierer (*Priority Resolver*) bestimmt die Anforderung mit der höchsten Priorität, und im In-Service-Register (ISR) wird gespeichert,

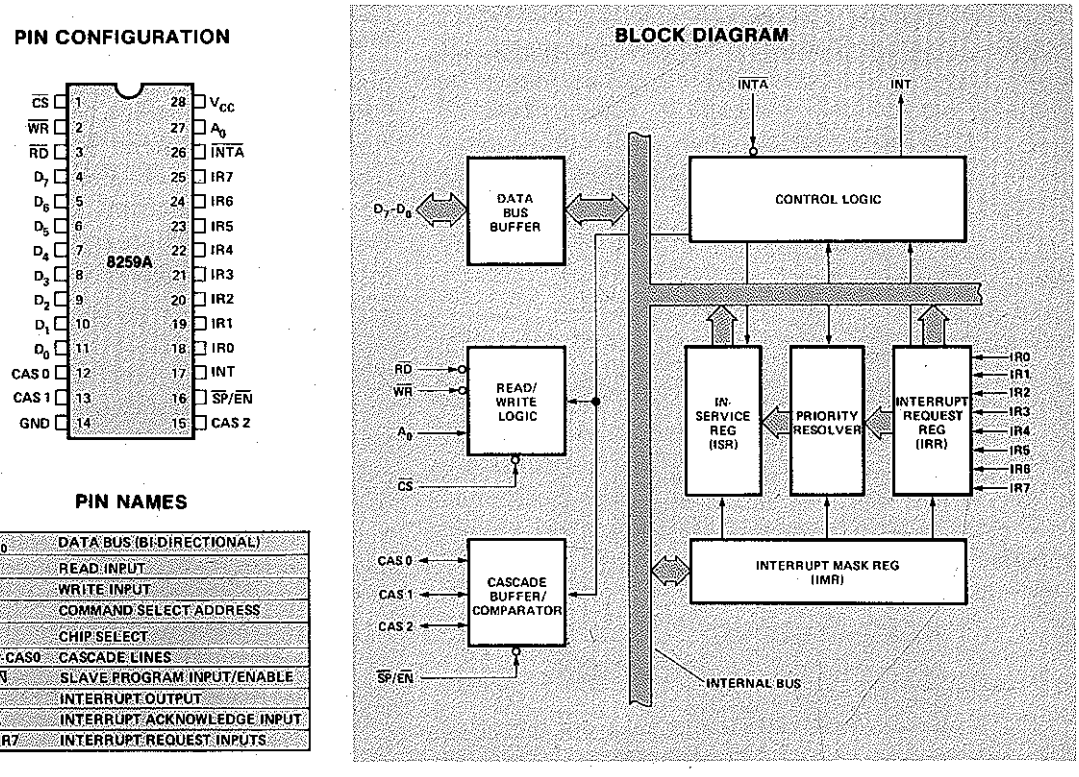


Bild 1. Blockschaltung und Anschlußbelegung des Interrupt-Controllers 8259 nach Original-Unterlagen des Herstellers (Intel)

PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A_0	COMMAND SELECT ADDRESS
$\overline{CS}$	CHIP SELECT
CAS2-CAS0	CASCADE LINES
$\overline{SP}/\overline{EN}$	SLAVE PROGRAM INPUT/ENABLE
INT	INTERRUPT OUTPUT
$\overline{INTA}$	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

welche der anstehenden Anforderungen gerade bearbeitet werden. Mit dem Interrupt-Masken-Register (*Interrupt Mask Register*, IMR) können bestimmte Anforderungen blockiert werden, so daß das Flipflop im Interrupt-Anforderungs-Register nicht gesetzt wird.

Mit der Steuerlogik werden alle notwendigen Steuersignale für den internen Datenverkehr generiert, außerdem wird hier das Signal $\overline{\text{INTA}}$ verarbeitet und das Signal INT erzeugt.

Über den Datenbus-Puffer (*Data Bus Buffer*) wird im Zusammenhang mit der Schreib/Lese-Logik (*Read/Write Logic*) der gesamte Datenverkehr zwischen dem Interrupt-Controller und der CPU abgewickelt. Außerdem beinhaltet die Schreib/Lese-Logik zwei Register für das Initialisierungs-Code-Wort (*Initialization Code Word*, ICW) und das Operations-Steuer-Wort (*Operation Command Word*, OCW). Mit dem Kaskadierungs-Puffer/Komparator (*Cascade Buffer/Comparator*) besteht die Möglichkeit, in einem System mehrere Interrupt-Controller zu installieren und somit bis zu 64 programmunterbrechende Peripherie-Einheiten anzuschließen.

2.2 Ein/Ausgangs-Signale und Funktion

IR0...IR7: Interrupt-Anforderungs-Leitungen.

Mit einer positiven Flanke an den Anforderungsleitungen ist ein Interrupt auslösbar, es sei denn, im Masken-Register ist das entsprechende Maskierungs-Flipflop gesetzt, das eine Interrupt-Auslösung sperrt.

D0...D7: Datenleitungen.

Die Datenleitungen stellen die Verbindung zum System-Datenbus her. Sie sind bidirektional ausgelegt und werden nur aktiviert, wenn der Baustein vom System aus angesprochen wird.

INT: Interrupt-Leitung.

Diese Leitung ist direkt mit der CPU verbunden und signalisiert der CPU einen anstehenden Interrupt.

$\overline{\text{INTA}}$: Interrupt-Bestätigung (*Interrupt Acknowledge*).

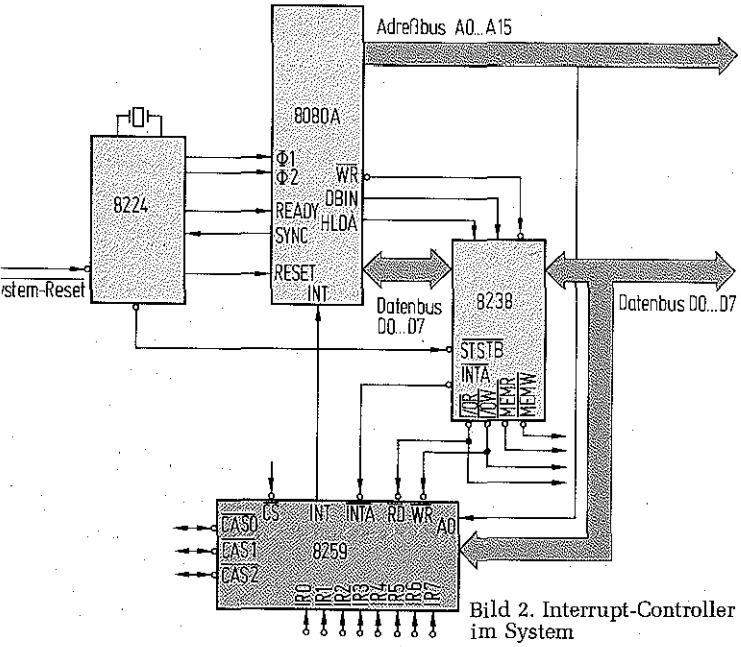


Bild 2. Interrupt-Controller im System

Im allgemeinen arbeitet der Interrupt-Controller gemäß *Bild 2* im System zusammen mit einem System-Steuer- und Bus-Treiber-Baustein (z. B. 8228 oder 8238). Nach der Akzeptierung des Interrupts durch die CPU sendet dieser Baustein drei $\overline{\text{INTA}}$-Impulse an den Interrupt-Controller, die diesen veranlassen, einen Unterprogrammaufruf (CALL) auf den Datenbus zu schalten und damit die CPU zu veranlassen, an dieser bestimmten Adresse mit dem Programm fortzufahren.

$\overline{\text{CS}}$: Baustein-Auswahl (*Chip Select*).

Durch einen L-Pegel (log. 0) an diesem Eingang wird der Baustein aktiviert. Ohne das Signal $\overline{\text{CS}}$ ist kein Schreib- oder Lesevorgang möglich.

$\overline{\text{WR}}$, $\overline{\text{RD}}$: Schreiben, Lesen (*Write, Read*).

Mit L an einem dieser Eingänge ist ein Einschreiben ($\overline{\text{WR}}$) von Steuer- und Initialisierung-Datenworten bzw. ein Auslesen ($\overline{\text{RD}}$) der Registerinhalte IRR, ISR und IMR möglich.

A0: Adresse.

Im Zusammenhang mit den Signalen $\overline{\text{WR}}$ und $\overline{\text{RD}}$ wird das Signal A0 benötigt, um bestimmte Registergruppen ansprechen und Schreib/Lese-Vorgänge durchführen zu können.

$\overline{\text{SP}}$: Untergeordneter Baustein (*Slave Programm*).

CAS0...CAS2: Kaskadierung.

Wie bereits erwähnt, ist eine Erweiterung mit zusätzlichen Interrupt-Controllern möglich (*Bild 3*). In diesem Fall arbeitet ein Baustein im übergeordneten Rang (*Master*) und die übrigen Interrupt-Controller im untergeordneten Rang (*Slave*). Mit H-Pegel am $\overline{\text{SP}}$-Eingang wird festgelegt, daß der entsprechende Baustein als Master arbeitet. Die übrigen Controller erhalten an diesem Eingang L. Die Leitungen CAS0...CAS2 fungieren im Falle von $\overline{\text{SP}}$ = H (Master) als Ausgang und im Falle von $\overline{\text{SP}}$ = L (Slave) als Eingang. Als Master schaltet der Baustein auf die drei CAS-Leitungen ein entsprechendes Identifizierungssignal an den anfordernden Interrupt-Controller.

Im Gesamtsystem ergibt sich folgender Funktionsablauf: An den Interrupt-Anforderungs-Leitungen (IR0...IR7) signalisieren eine oder mehrere Peripherieeinheiten, daß eine Programmunterbrechung erforderlich ist, um ein Bedienprogramm für die Peripherieeinheit zu starten. Der Interrupt-Controller akzeptiert die Anforderung, setzt die Priorität fest und gibt ein Unterbrechungssignal (INT) an die CPU ab.

Die CPU bestätigt das Unterbrechungssignal mit einem $\overline{\text{INTA}}$-Signal, das vom Systemsteuerbaustein (8228 oder 8238) weitergegeben wird und den Interrupt-Controller veranlaßt, einen Unterprogrammaufruf auf den Datenbus zu schalten. Dieser CALL-Befehl veranlaßt den Systemsteuerbaustein, zwei weitere $\overline{\text{INTA}}$-Impulse an den Interrupt-Controller zu senden, der dann beim jeweiligen Eintreffen des Impulses zwei weitere Datenworte, nämlich die Startadresse des Bedien-Unterprogramms, auf den Datenbus schaltet. Diese Startadresse ist vorher vom Programmierer nach bestimmten Regeln festzulegen. Der Interrupt-Controller wird im Verlauf des Hauptprogramms mit dieser Startadresse programmiert. Außerdem ist im Programmverlauf festzulegen, in welcher Betriebsart

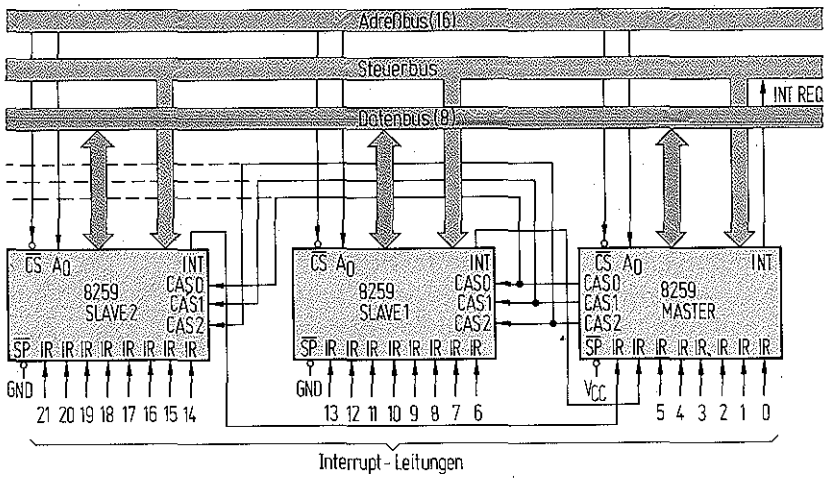


Bild 3.
Beispiel für eine Erweiterung auf 22 Interrupt-Eingänge

er arbeiten soll. Die Register des Interrupt-Controllers werden bezüglich der Adressierung wie eine Ein/Ausgabe-Einheit angesehen [3].

3 Programmierung und Betriebsarten

3.1 *Initialisierung und Initialisierungs-Code-Worte*

Bevor der Baustein 8259 irgendeine Funktion übernehmen kann, werden zwei Steuerworte benötigt, die angeben, welcher Adreßbereich im Speicher bei einem Interrupt zur Verfügung steht und wie dieser Adreßbereich aufgeteilt wird. Dazu muß noch gesagt werden, daß der Programmierer zwischen einer Speicherbereichslänge von vier oder acht Speicherplätzen pro Interrupt-Ebene wählen kann, wobei eine feste Zuordnung zwischen den Interrupt-Ebenen und der programmierten Anfangsadresse besteht.

Dazu ein Beispiel: Angenommen, vom Programmierer wurde die sedezimale Anfangsadresse 0B00 gewählt, so wird diese Programmadresse dem Interrupt-Level 0 (IR0) zugeordnet. Damit ergeben sich für die beiden Möglichkeiten der Speicherlänge pro Ebene die in *Tabelle 1* aufgeführten Anfangsadressen für das jeweilige Bedienprogramm.

Tabelle 1. Beispiel von Startadressen

	4 Speicherplätze	8 Speicherplätze
Level 0 (IR0) Adr.:	0B00	0B00
Level 1 (IR1) Adr.:	0B04	0B08
Level 2 (IR2) Adr.:	0B08	0B10
Level 3 (IR3) Adr.:	0B0C	0B18
Level 4 (IR4) Adr.:	0B10	0B20
Level 5 (IR5) Adr.:	0B14	0B28
Level 6 (IR6) Adr.:	0B18	0B30
Level 7 (IR7) Adr.:	0B1C	0B38

Für kurze Bedienprogramme reichen oft die acht Speicherplätze pro Ebene aus. Meistens wird es jedoch notwendig sein, längere Bedienprogramme aufzubauen. Dann wird es zweckmäßig sein, eine sogenannte Sprungtabelle aufzubauen, von wo aus in das jeweilige Interrupt-Programm gesprungen wird. Dazu genügt dann jeweils ein Speicherbereich von vier Speicherplätzen.

Wie in *Bild 4* ersichtlich, enthält das zweite Initialisierungs-Code-Wort (ICW2) das obere Adreßbyte der Anfangsadresse des Interrupt-Speicherbereichs. Das erste ICW enthält noch die Adreßbits A7...A5. Weitere Adreßangaben sind nicht notwendig, da der Interrupt-Controller die übrigen fünf bzw. sechs Bit in der im Beispiel angegebenen Form ergänzt (Tabelle 1) und den Interrupt-Ebenen zuordnet.

Bit D4 im ICW1 gibt mit Pegel H an, daß dieses Steuerwort das ICW1 ist. Mit D2 wird festgelegt, ob der Interrupt-Speicherbereich für jede Ebene vier oder acht Speicherplätze vorsieht. D1 gibt an, ob nur ein einziger Interrupt-Controller im System installiert ist oder mehrere. D0 und D3 liegen im ICW1 auf L-Pegel. So ergeben sich z. B. für ein System mit nur einem Interrupt-Controller, der Anfangsadresse des Interrupt-Speicherbereichs von 0B00 und mit einem Bereich von vier Speicherplätzen pro Ebene folgende Initialisierungs-Code-Worte (Bild 5):

ICW1: 00010110 = sedezimal 16
ICW2: 00001011 = sedezimal 0B

Für den Fall, daß mehrere Interrupt-Controller im System installiert sind, ist jeweils ein drittes Initialisierungs-Code-Wort (ICW3) notwendig. In diesem Code-Wort wird für den als Master fungierenden Controller angegeben, welcher Interrupt-Eingang mit einem Slave-Baustein belegt ist (Bild 3). Für die als Slave arbeitenden Controller wird im ICW3 eine Identifikationsnummer für jeden Slave-Baustein festgelegt (*Bilder 6* und *7*). Bei Interrupt-Anforderungen von einem dieser Bausteine schaltet der Master einen CALL-Befehl auf die Datenleitungen und auf die Leitungen CAS0...CAS2 die entsprechende Identifikationsnummer. Der Slave-Baustein sendet dann mit Eintreffen der $\overline{\text{INTA}}$-Impulse die dem Interrupt-Level

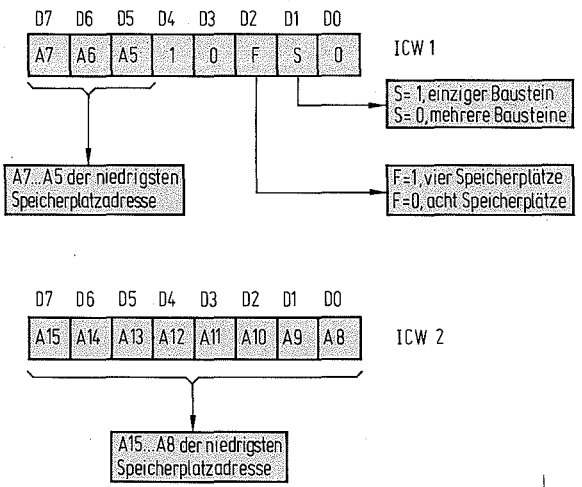


Bild 4. Format der Initialisierungs-Code-Worte 1 und 2

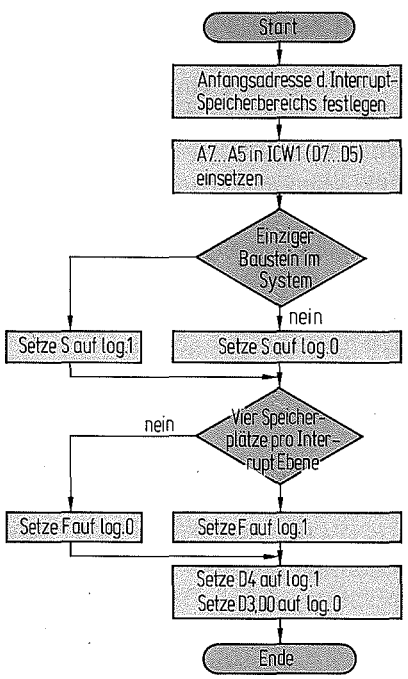


Bild 5. Auswahl des Initialisierungs-Code-Wortes 1

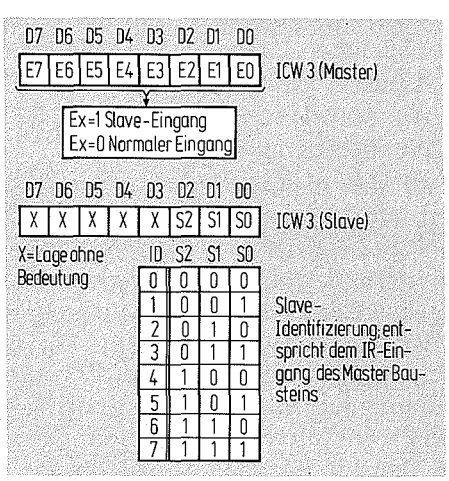


Bild 6. Format des Initialisierungs-Code-Wortes 3

zugeordnete Bedienprogrammadresse auf den Datenbus.

Da die Initialisierung für den Interrupt-Controller als Gesamtkomplex notwendig ist, müssen alle Initialisierungs-Code-Worte nacheinander an den Interrupt-Controller ausgegeben werden.

3.2 Operations-Modifikationen und Operations-Steuer-Worte

3.2.1 Verschachtelungs-Modus

Nach dem Einschreiben der Initialisierungs-Code-Worte befindet sich der Interrupt-Controller im sogenannten Verschachtelungs-Modus (*Nested Mode*) und benötigt zum Akzeptieren von Interrupts keine weiteren Steuerworte. In diesem Zustand hat der Interrupt-Eingang 0 (IR0) die höchste und der Interrupt-Eingang 7 (IR7) die niedrigste Priorität. Treten jetzt Interrupt-Anforderungen auf, wird die mit der höchsten Priorität zuerst bedient, und im In-Service-Register (ISR) wird das entsprechende Bit gesetzt. Solange dieses Bit gesetzt ist, sind Interrupt-Anforderungen mit niedrigerer Priorität gesperrt. Interrupt-Anforderungen mit höheren Prioritäten jedoch werden zugelassen, d. h. das gerade laufende Bedienprogramm wird erneut unterbrochen, um die Peripherieeinheit mit der höheren Priorität zu bedienen. Voraussetzung dafür ist natürlich, daß die CPU durch einen entsprechenden Befehl in den Zustand versetzt wurde, Interrupts anzunehmen (*Enable Interrupt*, EI).

Durch ein weiteres Steuerwort, das jetzt Operations-Steuer-Wort heißt (*Operation Command Word*, OCW) besteht die Möglichkeit, die Interrupt-Eingänge mit einer sogenannten Maske zu sperren. Dieses OCW1 besteht aus einem 8-bit-Wort, in dem die auf H liegenden Bits den zugehörigen Eingang sperren (*Bild 8*).

Für eine Initialisierung mit anschließender Maskierung ergibt sich z. B. für das System SBC 80/20 die Befehlsfolge:

```
      ⋮
INIT: DI          ;CPU-INTER.-EING. SPERREN
      MVI   A,16H ;STEUERWORTE FÜR START-
      OUT   0DAH  ;ADRESSE 0B00 UND SPEI-
                  CHER-
      MVI   A,0BH ;PLATZINTERVALL 4
      OUT   0DBH  ;AUSGEBEN.
      MVI   A,07H ;MASKE FÜR IR0 – IR2
      OUT   0DBH  ;SETZEN.
      EI          ;INTERRUPTSPERRE AUF-
      ⋮           HEBEN
```

Dabei ist zu beachten, daß sich nur ein Interrupt-Controller im System befindet und daß die beiden Steuer-Register die Adressen DA_{16} und DB_{16} haben. Eine interne Sequenz-Steuerung überwacht das Einschreiben der Steuer-Worte, so daß die Maske ins Interrupt-Masken-Register (IRM) gelangt, obwohl zweimal die Adresse DB_{16} angesprochen wurde. Weiterhin ist zu beachten, daß der Interrupt-Eingang der CPU während der Initialisierung und auch bei anderen Schreib/Lese-Operationen per Software gesperrt wird (DI,EI).

Wie bereits erwähnt, wird im In-Service-Register (ISR) bei der Bearbeitung eines Interrupts das entsprechende Bit gesetzt. Dadurch werden die Interrupt-Eingänge mit niedrigeren Prioritäten gesperrt. Um dieses Bit zurückzusetzen, ist ein weiteres Operations-Steuer-Wort (OCW2) notwendig. Der Zeitpunkt, zu dem es innerhalb des Interrupt-Bedienprogramms an den Controller ausgegeben wird, kann vom Programmierer bestimmt werden. Da es meistens zum Schluß des Bedienprogramms ausgegeben wird, nennt man dieses Wort auch „End-of-Interrupt" (EOI). Das Format gibt *Bild 9* wieder. Dieses Steuerwort hat in bezug auf die weiteren Betriebsmodifikationen mehrere Bedeutungen. Im Verschachtelungs-Modus kommt diesem Steuerwort folgende Bedeutung zu: EOI = H, SEOI = L, in diesem Fall, dem sogenannten nicht-spezifischen EOI wird automatisch das Bit im In-Service-Register (ISR) mit der höchsten Priorität zurückgesetzt, denn im Verschachtelungs-Modus war das letzte zu bearbeitende Bedienprogramm dasjenige mit der höchsten Priorität.

Somit lautet das OCW2 im nicht-spezifischen EOI:

$$0010\ 0000 = 20_{16}$$

Damit ergibt sich folgende Routine für diesen EOI (*Bild 10*):

```
        .
        .
        .
DI
MVI     A,20H   ;EOI-STEUERWORT
OUT     0DAH    ;AUSGEBEN
EI
        .
        .
        .
```

3.2.2 Modus mit rotierenden Prioritäten

In manchen Anwendungsfällen ist es von Vorteil, nach Beendigung eines Bedienprogramms der gerade unterbrechenden Einheit die niedrigste Priorität zuzuweisen, um den anderen Einheiten Vorrang zu gewähren. Im Extremfall muß dann diese Einheit warten, bis sieben andere Einheiten durch ein Interrupt-Programm bedient wurden. Hierbei handelt es sich um den Modus mit rotierenden Prioritäten. In diesem Modus sind zwei Betriebsarten möglich:

1. Automatische Rotation:

Die Rotation der Prioritäten erfolgt automatisch mit den beschriebenen Auswirkungen. Angenommen, das In-Service-Register hat folgenden Zustand:

```
Bit-Nr.: 7 6 5 4 3 2 1 0
         0 1 0 1 0 0 0 0
         n ──────────► h
```

n = niedrigste Priorität,
h = höchste Priorität.

Daraus ist entnehmbar, daß zur Zeit die Einheit mit dem Interrupt-Eingang IR4 bedient wird, da IR4 noch die höchste Priorität besitzt. Nach Beendigung des Bedienprogramms und entsprechender Rotation der Priorität hat der Eingang IR5 höchste und Eingang IR4 niedrigste Priorität:

```
Bit-Nr.: 7 6 5 4 3 2 1 0
         0 1 0 0 0 0 0 0
           h n ──────────►
```

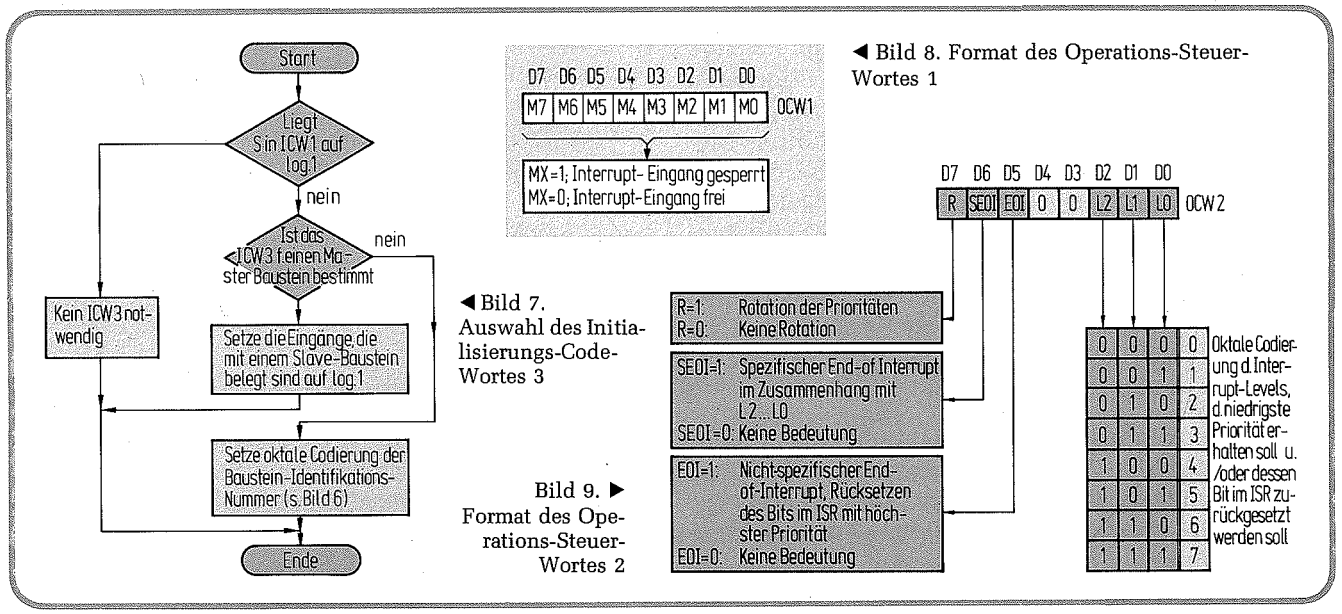
Ausgelöst wird diese Rotation durch einen EOI mit einem Operations-Steuer-Wort, in dem die Bits D7 (R) und D5 (EOI) auf H und D6 (SEOI) auf L liegen.

Damit ergibt sich folgendes OCW2 für eine automatische Rotation (Bild 10):

$$1\ 0\ 1\ 0\ 0\ 0\ 0\ 0 = A0_{16}$$

2. Spezifische Rotation

Durch das Setzen der Bits D7 (R) und D6 (*Specific End of Interrupt*, SEOI) wird dem Programmierer die Möglichkeit gegeben, mit dem OCW2 am Ende eines Interrupt-Bedienprogramms den Interrupt-Eingang mit der niedrigsten – und damit auch automatisch der höchsten – Priorität festzulegen. Wird z. B. IR5 mit der



◄ Bild 7. Auswahl des Initialisierungs-Code-Wortes 3

◄ Bild 8. Format des Operations-Steuer-Wortes 1

Bild 9. ► Format des Operations-Steuer-Wortes 2

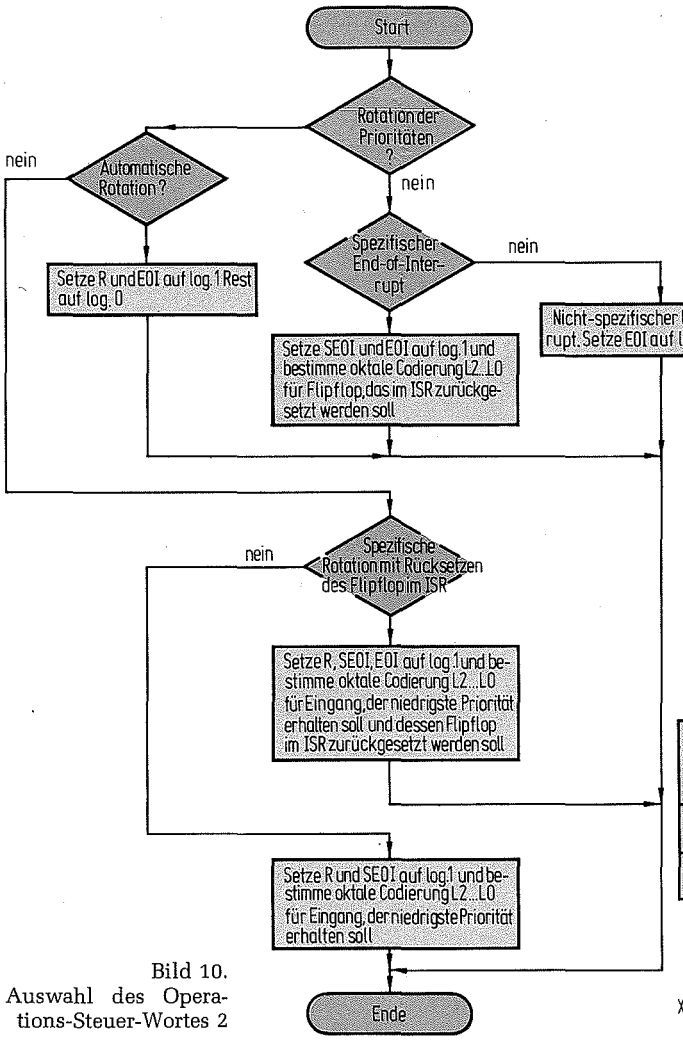
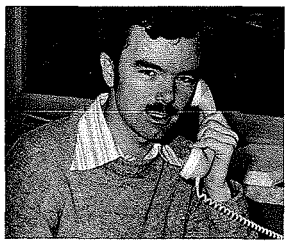


Bild 10.
Auswahl des Operations-Steuer-Wortes 2



Dipl.-Ing. Horst Wältring stammt aus dem Münsterland. Nach dem Studium der Nachrichtentechnik an der TU Hannover war er seit Anfang 1976 im Digitallabor der Dornier-System GmbH, Friedrichshafen, tätig und spezialisierte sich auf dem Gebiet der Prozeßrechnertechnik. Im Sommersemester 1978 übernahm er für das Fach „Programmierung von Mikroprozessoren" einen Lehrauftrag an der Fachhochschule in Konstanz. Seit Sept. 78 nimmt er die Aufgaben eines Projektbegleiters für vom BMFT geförderte Projekte der technischen Kommunikation bei der DFVLR in Köln wahr.
Hobbys: Filmen, Heimwerken, Zimmerpflanzen.
Diensttelefon (0 22 03) 6 01 29 73
ELEKTRONIK-Leser seit 1972

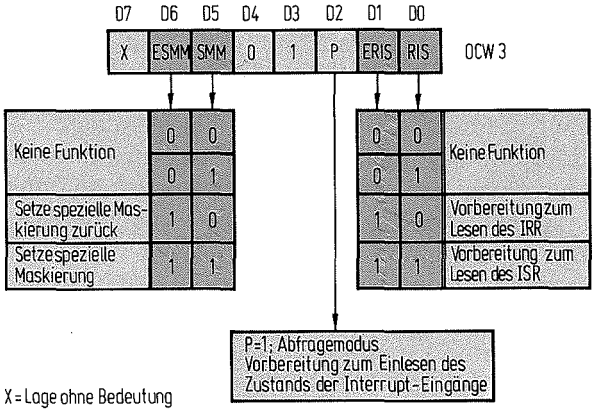


Bild 11. Format des Operations-Steuer-Wortes 3

niedrigsten Priorität belegt, so hat IR6 die höchste Priorität. Zur oktalen Codierung des Interrupt-Eingangs, der die niedrigste Priorität erhalten soll, dienen die Bits D2...D0 (L2...L0).

Ist im OCW2 auch das Bit D5 (EOI) gesetzt, so wird mit dem Einschreiben des OCW2 auch das Bit im ISR zurückgesetzt, das mit den Bits D2...D0 (L2...L0) bezeichnet wurde (*Tabelle 2*). Somit hat der Programmierer die Möglichkeit, auch ohne das entsprechende Bit im ISR zurückzusetzen (EOI=0), die Prioritäten z. B. im Hauptprogramm neu festzulegen. Umgekehrt ist es natürlich auch möglich, wenn im Rotations-Modus ein EOI gegeben werden muß, nur ein bestimmtes Bit im ISR zurückzusetzen, ohne die Prioritätslagen zu ändern (siehe Wort 10 in Tabelle 2). Hierbei handelt es sich um den spezifischen „End of Interrupt" (SEOI) mit D6 und D5 auf H und D2...D0 oktal codiert.

3.2.3 Modus mit spezieller Maskierung (*Special Mask Mode*, SMM)

Für diesen Modus wird das Operations-Steuer-Wort 3 (OCW3, *Bild 11*) benötigt. Will man im Verlauf eines Bedienprogramms einen Eingang mit niedrigerer Priorität z.B. IR4 sperren (maskieren) bei dem aber eventuell schon eine Anforderung ansteht (Bit 4 im IRR gesetzt), so wird beim Auftreten einer weiteren Anforderung mit noch niedrigerer Priorität z. B. IR6 die Annahme dieses Interrupts nach Beendigung des gerade laufenden Bedienprogramms dadurch verhindert, daß einerseits das Bit 4 im IRR gesetzt ist und damit bei der Prioritätsanalyse IR4 als Eingang mit höchster Priorität erkannt wird, andererseits dieser Eingang aber maskiert ist. Die Anforderung kann nicht an das ISR weitergegeben werden, da die später erfolgte Maskierung sich sowohl auf das IRR als auch auf das ISR auswirkt. Um diese Blockierung zu verhindern, kann mit dem OCW3 das SMM-Flipflop gesetzt werden. Dazu werden im OCW3 die Bits D5 und D6 (ESMM, SMM) auf H gesetzt. Das ergibt folgendes OCW3 (*Bild 12*):

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 0 = 68_{16}.$$

Dieser Modus bleibt bestehen, bis das Flipflop wieder zurückgesetzt wird mit D6 (ESMM) = H und D5 (SMM) = L. Höhere Prioritäten werden durch diesen Modus nicht beeinflußt.

3.2.4 Abfrage-Modus (*Polled Mode*)

In dieser Betriebsart wird der Zustand der Interrupt-Eingänge abgefragt und damit der direkte Zugriff der Peripherieeinheiten zu ihrem Bedienprogramm aufgehoben. Für diesen Modus ist die Interrupt-Fähigkeit der CPU vorher aufzuheben (DI). Mit einem OCW3, in dem das Bit D2 (P) gesetzt ist, wird der Inter-

rupt-Controller auf die Abfrage vorbereitet. Mit dem nächsten Leseimpuls (RD), den der Interrupt-Controller als eine Interrupt-Bestätigung auffaßt, wird die höchste Priorität der anstehenden Anforderungen bestimmt und das entsprechende Flipflop im In-Service-Register gesetzt. Im eingelesenen Datenwort ist das Bit D7 (I) dann gesetzt, wenn eine Anforderung von außen angelegen hat. Die Bits D2...D0 (W2...W0) geben in oktaler Codierung an, welcher anfordernde Interrupt-Eingang höchste Priorität hat und somit zuerst bedient werden soll (*Bild 13*). Eine entsprechende Programmsequenz würde dann wie folgt aussehen:

```
.
.
.
DI              ;CPU-INTERR. SPERREN
.
.
.
MVI   A,0CH  ;OCW3 LADEN (ABFRAGEMODUS)
OUT   0DAH   ;UND AUSGEBEN
.
.
.
IN    0DAH   ;STATUS EINLESEN
RAL          ;LAG EINE ANFORDERUNG AN?
JC    SUB    ;JA, SPRINGE IN INTERR.-PRG.
MVI   B,01   ;NEIN, WEITERMACHEN
.
.
.
```

4 Einlesen der Registerinhalte

Will man im Verlauf des Programms den Status der verschiedenen Register des Interrupt-Controllers feststellen, so ist dies möglich durch einen Lesevorgang, der vorher mit dem OCW3 vorbereitet wurde. Nur beim Einlesen des Masken-Registers (IMR) kann das OCW3 vorher entfallen.

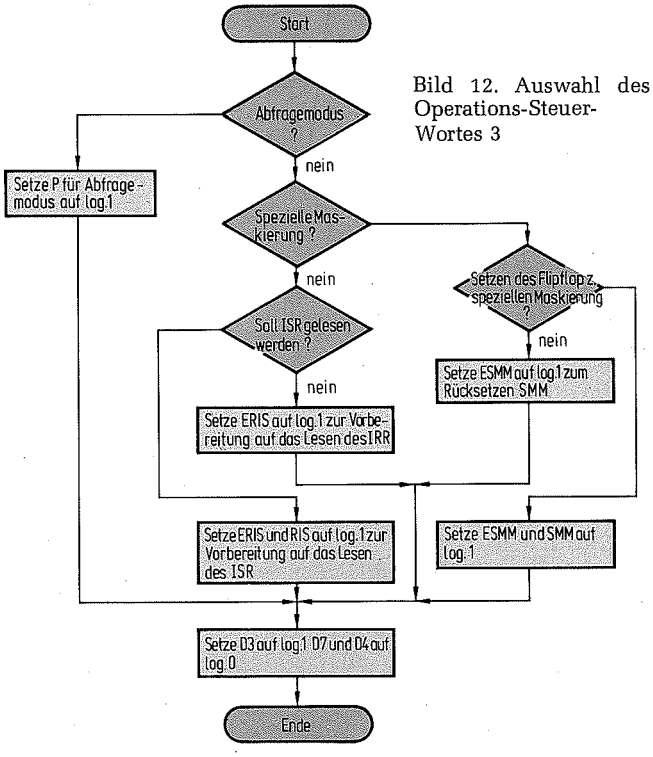


Bild 12. Auswahl des Operations-Steuer-Wortes 3

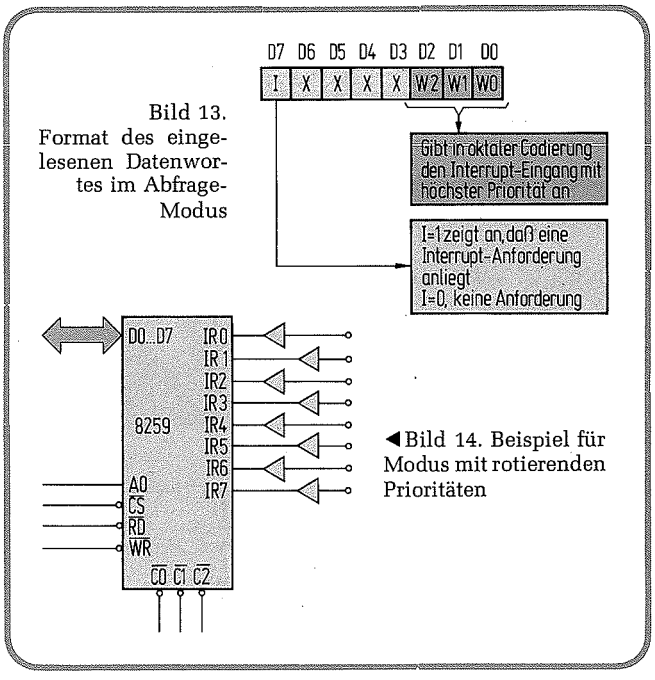


Bild 13. Format des eingelesenen Datenwortes im Abfrage-Modus

◀ Bild 14. Beispiel für Modus mit rotierenden Prioritäten

Für das Einlesen des Interrupt-Request-Registers (IRR) muß das Bit D1 (ERIS) auf H und das Bit D0 (RIS) auf L gesetzt werden (Bild 11 und Tabelle 2). Für das Einlesen des In-Service-Registers (ISR) müssen beide Bits auf H liegen. So ergibt sich z. B. für das Einlesen des IRR folgende Befehls-Sequenz:

```
.
.
.
MVI   A,0AH  ;OCW3 LADEN UND ZUR VOR-
OUT   0DAH   ;BEREITUNG
             ;AUF LESEN IRR AUSGEBEN
IN    0DAH   ;IRR EINLESEN
.
.
.
IN    0DAH   ;IRR EINLESEN
.
.
.
```

Es ist nicht notwendig, vor jedem Einlesen eines Registers eine Schreiboperation (OUT ...) durchzuführen, wenn wiederum das gleiche Register eingelesen werden soll.

Nur im Abfrage-Modus ist vor jedem Einlesen ein entsprechendes OCW3 notwendig. Sind in einem OCW3 sowohl das Bit D2 (P) als auch D1 (ERIS), so hat der Abfrage-Modus Vorrang.

Bei kaskadierten Bausteinen benötigt jeder eine auf ihn zugeschnittene Initialisierung und auch jeweils ein entsprechendes Operations-Steuer-Wort (z. B. OCW2, EOI).

5 Anwendungen

Bei Erweiterungen sind 7n + 1 Interrupt-Eingänge mit Peripherie-Einheiten belegbar. Dabei ist n die Anzahl der im System installierten Interrupt-Controller. In Bild 3 sind also beispielsweise 22 Interrupt-Eingänge beschaltbar. Im Maximalausbau sind neun In-

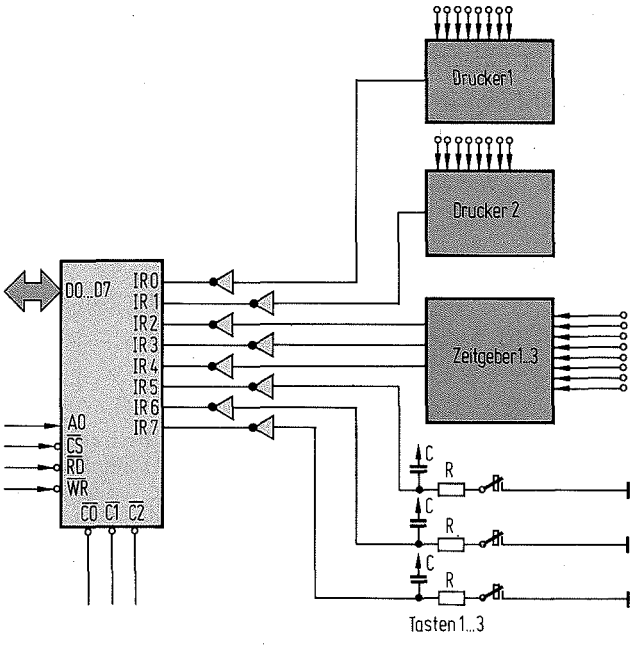


Bild 15. Beispiel für den Verschachtelungs-Modus

terrupt-Controller in einem System einsetzbar. In dem Fall sind alle Eingänge des übergeordneten Bausteins mit Interrupt-Leitungen der untergeordneten Bausteine belegt.

Im Bild 14 sind acht Datenübertragungskanäle angeschlossen. Der Interrupt-Controller arbeitet im Modus mit automatisch rotierenden Prioritäten, so daß im Extremfall jeder Datenkanal auf die sieben anderen Kanäle warten muß, bis er wieder bedient wird (Warteschlange).

Bild 15 zeigt eine gemischte Belegung der Interrupt-Eingänge im Verschachtelungs-Modus. Dabei hat der Drucker 1 immer höchste Priorität, gefolgt von Drucker 2 und den Zeitgebern 1...3. Taste 3 wurde die niedrigste Priorität zugeordnet.

Literatur

[1] Intel Single-Board-Computer SBC 80/20 Hardware Reference Manual.

[2] Intel Data Catalog 77.

[3] Siemens, Mikroprozessor-Bausteine.

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

Tabelle 2. Initialisierungs-Code-Worte (ICW) und Operations-Steuer-Worte (OCW)

Nr.	Steuer-wort	Art	A0*	D7	D6	D5	D4	D3	D2	D1	D0	Beschreibung
1	ICW1	4E	0	A7	A6	A5	1	0	1	1	0	Initialisierungs-Byte 1, 4 Speicherplätze, einziger Baustein
2	ICW1	4K	0	A7	A6	A5	1	0	1	0	0	Initialisierungs-Byte 1, 4 Speicherplätze, Kaskadierung
3	ICW1	8E	0	A7	A6	A5	1	0	0	1	0	Initialisierungs-Byte 1, 8 Speicherplätze, einziger Baustein
4	ICW1	8K	0	A7	A6	A5	1	0	0	0	0	Initialisierungs-Byte 1, 8 Speicherplätze, Kaskadierung
5	ICW2		1	A15	A14	A13	A12	A11	A10	A9	A8	Initialisierungs-Byte 2, MSB der niedrigsten Adresse
6	ICW3	M	1	E7	E6	E5	E4	E3	E2	E1	E0	Initialisierungs-Byte 3, übergeordneter Baustein
7	ICW3	S	1	0	0	0	0	0	S2	S1	S0	Initialisierungs-Byte 3, untergeordneter Baustein
8	OCW1		1	M7	M6	M5	M4	M3	M2	M1	M0	Maskierung der Interrupt-Eingänge
9	OCW2	E	0	0	0	1	0	0	0	0	0	Nicht-spezifischer EOI
10	OCW2	SE	0	0	1	1	0	0	L2	L1	L0	Spezifischer EOI, L2...L0 ist oktaler Code für das Flipflop im ISR, das zurückgesetzt werden soll
11	OCW2	RE	0	1	0	1	0	0	0	0	0	Automatische Rotation der Prioritäten
12	OCW2	RSE	0	1	1	1	0	0	L2	L1	L0	Spezielle Rotation der Prioritäten, L2...L0 ist oktaler Code für das Flipflop im ISR, das zurückgesetzt werden soll und dessen Eingang niedrigste Priorität erhalten soll
13	OCW2	RS	0	1	1	0	0	0	L2	L1	L0	Spezielle Rotation der Prioritäten, L2...L0 ist oktaler Code für den Eingang, der niedrigste Priorität erhalten soll
14	OCW3	A	0	x	0	0	0	1	1	0	0	Abfrage-Modus
15	OCW3	LIS	0	x	0	0	0	1	0	1	1	Vorbereitung zum Lesen des ISR
16	OCW3	LIR	0	x	0	0	0	1	0	1	0	Vorbereitung zum Lesen des IRR
17	OCW3	SM	0	x	1	1	0	1	0	0	0	Modus mit spezieller Maskierung setzen
18	OCW3	RSM	0	x	1	0	0	1	0	0	0	Modus mit spezieller Maskierung zurücksetzen

* Adreßbit A0 x = Lage ohne Bedeutung

William E. Nicholson,
Richard W. Blasco,
Kadiri R. Reddy

Für arithmetische Operationen brauchen Mikroprozessoren im allgemeinen so lange, daß verschiedene Anwendungen von vornherein nicht möglich sind. Aus dieser Misere soll ein neuer Baustein heraushelfen, der Addition, Multiplikation und Abspeicherung von 12-bit-Zahlen in 300 ns schafft und damit als schnelles Zahlenrechenwerk bezeichnet werden kann. Der SPP (***Signal Processing Peripheral***), wie der Hersteller den Baustein nennt, ist optimiert für digitale Filterung, schnelle Fourier-Transformation (FFT) und Matrizen-Berechnungen. Ein anderes Zusatzrechenwerk, das allerdings völlig unterschiedliche Anwendungen hat, wurde in [6] besprochen.

Schnelles Rechenwerk erweitert Mikroprozessor-Systeme

Digitale Datenverarbeitung war ursprünglich auf teure und umfangreiche Systeme begrenzt. Die LSI-Technologie hat dieses Problem verringert, aber nicht beseitigt. Ein Haupthindernis war, daß weder LSI-Bausteine noch Mikroprozessoren schnell und flexibel genug waren, um komplexe Algorithmen, z. B. schnelle Fourier-Analysen, zu implementieren. Hierfür werden Hochgeschwindigkeits-Minicomputer oder bipolare Anordnungen benötigt.

Die bisherige Schwelle, die für den Ersatz analoger durch digitale Techniken vorhanden war, wurde durch technologische Entwicklungen durchbrochen.

Die Fortschritte sind:

- Spezial-Architekturen, die die Implementierung von Algorithmen für die digitale Signalverarbeitung erlauben;
- VMOS-Technologie;
- Entwicklung eines Hochgeschwindigkeits-Multiplizierers auf demselben Chip;
- Schaltungstechnik für dichte Speicher auf dem Chip mit geringem Leistungsverbrauch;
- billiger monolithischer Analog/Digital-Umsetzer.

Der SPP nutzt die angegebenen Fortschritte aus. Er wird an den Mikroprozessor 6800 als Peripherie-Baustein angeschlossen und ist unmittelbar kompatibel mit dessen Bus, kann aber auch an andere µPs angeschlossen werden. Die Fähigkeit des SPP, Zahlen sehr schnell zu verarbeiten, versetzt ein Standard-µP-System in die Lage, komplizierte digitale Algorithmen in Realzeit zu verarbeiten. In einem µP-System kann der SPP als „Hardware"-Unterprogramm behandelt und aufgerufen werden. Er arbeitet dann unabhängig vom µP. Der SP hat einen direkten Serien-Eingang für einen Analog/Digital-Umsetzer für hohe Geschwindigkeiten. Dadurch können Realzeit-Rechnungen mit abgetasteten Analogsignalen durchgeführt werden, ohne den µP selbst zu belasten. *Bild 1* zeigt den SPP in Verbindung mit einem µP und anderen Bausteinen.

1 Architektur des SPP

Die Ausführung des SPP stellt einen Kompromiß zwischen Anwendungsmöglichkeiten und Kosten dar. Daraus ergab sich die Innenschaltung nach *Bild 2*, die folgende grundsätzliche Einheiten und Eigenschaften zeigt:

- ROM und RAM,
- Hochgeschwindigkeits-Parallel-Multiplizierer,
- Serien- und Parallel-Ein/Ausgabe-Leitungen,
- vier Adressiermöglichkeiten,
- spezielle Register für die Implementierung von Iterations-Schleifen, Unterprogrammen und Tabellen, indirekte Sprünge, blockweise Übertragung über Ein- und Ausgänge, Ein-Bit-Verzögerungen (z^{-1}) für Filter-Algorithmen (Z-Transformation),
- Vielfach-Bus-Struktur für Parallel-Operationen.

1.1 Speicher

Der Speicher enthält ein 256 x 17-bit-ROM, in welchem das Programm des SPP gespeichert ist, und Ar-

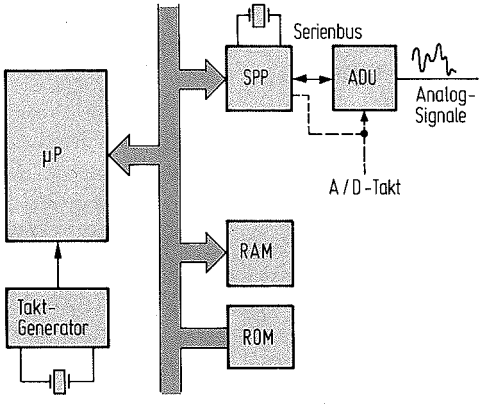


Bild 1. SPP in einem µP-System

beitsspeicher mit 128 x 16 bit RAM und 128 x 16 bit ROM. Dieser Speicher kann nicht erweitert werden. Dadurch werden die Anwendung eines größeren Bausteins, die Vergrößerung der Chip-Fläche, eine Leistungserhöhung und andere Nachteile vermieden. Ein µP kann aber mit mehreren SSPs zusammenarbeiten. So konnten ein Gehäuse mit nur 28 Anschlüssen und ein Leistungsverbrauch von nur 0,5 W erreicht werden.

Der Datenspeicher ist so organisiert, daß zwei Operanden in einem einzigen Zyklus geholt werden. Die 256 Datenwörter sind in einer Wort-Matrix von 32 (Basis) x 8 (Versatz) organisiert. Nur die Basis-Information wird zum RAM/ROM-Kern geführt. Alle acht Versatzworte, die dieser Basis zugeordnet sind (128 bit) sind parallel zugänglich. Zwei unabhängige Multiplexer für den Versatz wählen zwei Operanden von den acht Ausgangsworten aus. Innerhalb einer Basis mit acht Worten hat der Speicher deshalb drei Ports. Der Speicher ist so aufgeteilt, daß jede Basisgruppe aus 4 Worten im ROM und 4 Worten im RAM besteht.

Diese Anordnung von RAM und ROM erlaubt die Anwendung fester Koeffizienten, die in vorteilhafter Weise gespeichert werden können, so daß die Chip-Abmessungen sich erheblich verringern. Bei Anwendungen für Filter oder große Matrizen, wie schnelle Fourier-Analyse, wird die Block-Übertragung eingesetzt, um die Möglichkeiten des internen RAM-Speichers zu erweitern. Bei den meisten Anwendungen sind etwa die Hälfte der gespeicherten Daten Koeffizienten, und die Mischung von RAM und ROM wäre inkonsequent.

1.2 Spezialregister

Ein Spezialregister (*Scratchpad*) sorgt dafür, daß auf allgemeine Daten ebenso effektiv zugegriffen werden kann, wie auf Daten, die in einer 8-Wort-Basis enthalten sind. Ein zusätzlicher Multiplexer beim Port U des Speichers erlaubt einen Zugriff auf das Spezialregister anstelle des Hauptspeichers. Da dieser Vorgang unabhängig von der Basis-Gruppe ist, kann der Inhalt des Spezialregisters als frei verfügbare Basis-Gruppe aufgefaßt werden. Diese Eigenschaft verdoppelt den Wirkungsgrad von Vergleichsvorgängen und ähnlichen Programmen.

Die Register VT und VP sehen eine Verzögerung für Daten vor, die vom Port V des Speichers kommen. Der Lesezyklus für den Speicher geht dem Schreibzyklus voran. Daten des n-ten Lesezyklus werden im VT-Register gespeichert. Während des nächsten Zyklus gehen Daten des Zyklus n+1 in das VT-Register, während die Daten des Befehls n in das VP-Register gehen. Diese Daten können während des Schreibzyklus n+1 in den Speicher zurückgeführt werden, wofür ein besonderer Befehl erforderlich ist. So können Verzögerungen für Z-Transformationen (z^{-1}) mit minimaler Software implementiert werden.

1.3 SPP-Programme

Das SPP-Programm ist in einem ROM mit 256 x 17 bit enthalten. Es besteht die Möglichkeit, *ein* Unterprogramm einzufügen, da das Register RAR (*Return Adress Register*) die Rücksprungadresse speichert. Damit werden die Programme mit max. 256 Schritten

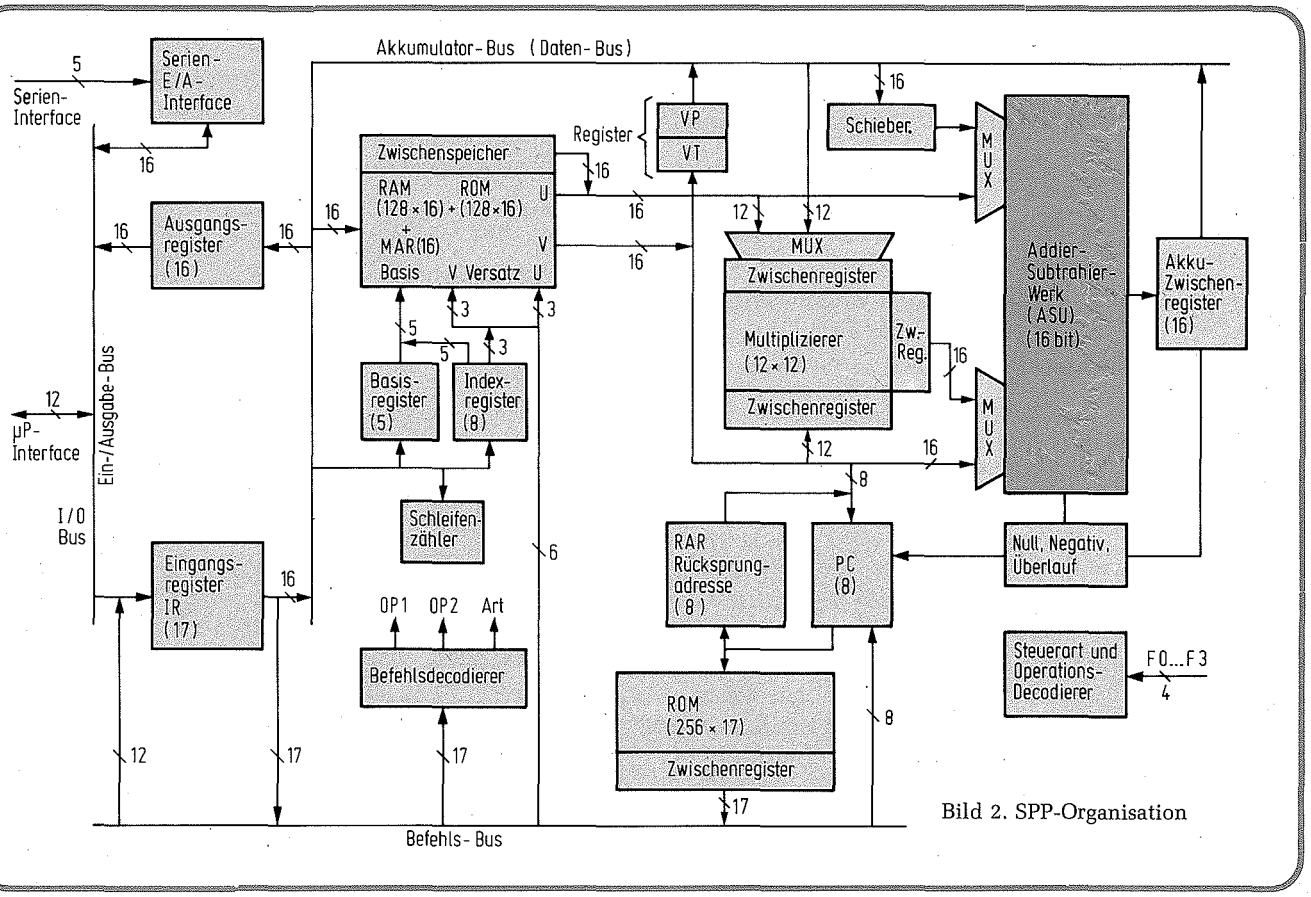


Bild 2. SPP-Organisation

leistungsfähiger. 17-bit-Befehle ermöglichen Mehrfach-Operationen, so daß die Leistungsfähigkeit weiter vergrößert wird. Es hat sich gezeigt, daß die meisten Programme mit 256 Schritten auskommen.

1.4 Schleifenzähler

Weitere Register erhöhen die Beweglichkeit des SPP. Ein Schleifenzähler erlaubt bis zu 32 Iterationen. Besondere Sprungbefehle (Sprungbedingung abhängig vom Schleifenzähler) erlauben die Iteration ohne zusätzliche Programmschritte. Der Schleifenzähler vermeidet die Software, die sonst für die erste Ebene von Schleifen erforderlich ist. Trotzdem können per Software Schleifen verschachtelt werden. Im Basisregister kann aufwärts oder abwärts gezählt werden, wodurch die Beweglichkeit in der Adressierung erhöht wird. Das Basisregister und der Schleifenzähler können vom Programmspeicher über das Eingangsregister geladen werden. Diese Möglichkeit erlaubt das Setzen des Basisregisters und Schleifenzählers über den Eingangsdaten-Port. Dadurch ist während der Ausführung eines Programms eine Veränderung der SPP-Routine möglich.

1.5 Indexregister

Das Indexregister wird benutzt, um auf Tabellen zuzugreifen. Es wird vom Akkumulator geladen und kann mit einem Befehl inkrementiert werden. Instruktionen von solchen Tabellen können verwendet werden, um den Inhalt des Indexregisters als Basis für den Datenspeicher zu verwenden. Der Inhalt der Tabelle kann sowohl für Daten- als auch für Sprungadressen für berechnete Sprung-Operationen benutzt werden. Besondere Befehle erlauben dem Basisregister, mit dem Indexregister zusammenzuarbeiten, wodurch Doppelbasisadressen ermöglicht werden. Das Indexregister wird auch verwendet, um schrittweise mit dem Datenspeicher zu arbeiten, wenn ein Block übertragen wird.

1.6 Addier-Subtrahier-Werk (ASU) und Akkumulator

Das Herz des SPP ist ein Addier-Subtrahier-Werk (ASU), welches mit einer Zweierkomplement-Arithmetik arbeitet. Vorgesehen sind Null-, Negativ- und Überlauf-Prüfung. Die Schaltung der Addierzelle mit einer Logik für hohe Geschwindigkeit zeigt *Bild 3*. Eine Summe mit 16 bit wird in einer Zeit von 40 µs ausgerechnet. An diesen Addierer ist ein Akkumulator angeschlossen. Ein Schieberegister erlaubt, den Inhalt des Akkumulators um 2 bit nach rechts zu schieben, um eine genaue Teilung durch 4 zu ermöglichen.

1.7 Multiplizierer

Ein modifizierter Multiplizierer nach einem Verschachtelungs-Algorithmus ist vorgesehen, um schnelle Multiplikation zu ermöglichen. Die Ausbreitungsverzögerung des Ablaufs durch den Multiplizierer beträgt 300 ns. Daten, die während des Befehls multipliziert werden, ergeben ein Produkt, das während des Befehls n + 1 verfügbar ist.

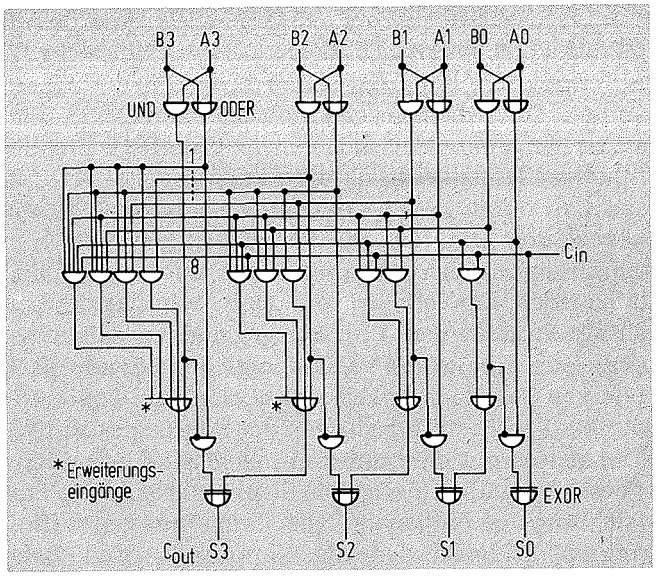


Bild 3. Addierwerk des SPP (S 2811 von der Fa. AMI)

Mit einiger Praxis kann diese Verzögerung ohne Verlust der Wirksamkeit des Programmes behandelt werden. Die Multiplizier-Verzögerung bestimmt die Zeit für den einen Befehlszyklus.

Getrennte Ein- und Ausgangsregister tauschen die Daten mit den SPP-Daten-Ports aus. Eine Serien-Zwischenlogik setzt die parallelen Zweierkomplement-Daten in serielle Zweierkomplement- oder vorzeichenbehaftete Zahlen um. Das Datenformat und die Quelle (Serien- oder Parallel-Port) sind durch Software wählbar.

1.8 Vielfach-Bus-Struktur

Eine Vielfach-Bus-Struktur verbindet die genannten Einheiten und erlaubt Paralleloperationen. Das Bus-System besteht aus Akkumulator-(Daten-)Bus, Befehls-Bus, zwei Multiplikations-Eingangsbussen und zwei ASU-Eingangs/Ausgangs-Bussen.

Durch die Vielfach-Bus-Struktur können Multiplikation, Addition und Speichern innerhalb eines einzigen Befehls-Zyklus erfolgen. Zusätzlich erlaubt der Eingangs/Ausgangs-Bus den Datenaustausch mit den Daten-Ports, gleichzeitig mit der Durchführung von Prozessen im SPP.

2 VMOS-Technologie

Neben der Architektur sorgt vor allem die VMOS-Technologie für die Leistungsfähigkeit des SPP. Die Strukturen zeigt *Bild 4*. Dabei werden V-förmige Nuten in die Silizium-Oberfläche geätzt. Der Ätzprozeß ist an den Nutenkanten schnell, aber senkrecht dazu langsam. Wenn die Nut voll geätzt ist, hört sie an den Kanten auf, geht aber senkrecht dazu weiter.

Der Transistor-Kanal wird nicht einheitlich dotiert. Er wird durch P-Diffusion und eine leicht dotierte PI-Epitaxial-Schicht gebildet. Beide Prozesse (und damit die Kanallänge) sind intensiv überwachte Fertigungsschritte. Die Kanalbreite wird durch die vertikale Struktur erhöht, ohne daß mehr Fläche benötigt wird. Dadurch und durch die unterschiedliche Dotie-

rung kann ein Transistor mit hoher Verstärkung auf kleiner Fläche realisiert werden. Die vertikale Struktur ermöglicht außerdem hohe Ströme pro Flächeneinheit. Die Arbeitsgeschwindigkeit eines MOS-Bausteins hängt hauptsächlich von der Gate-Kapazität der inneren Elemente ab, da die Logik-Vorgänge hauptsächlich durch Laden und Entladen der Belastungskapazitäten erfolgen. Die VMOS-Struktur mit ihrer verminderten Oberfläche verkleinert auch die Kapazitäten. Das Ergebnis ist ein Geschwindigkeits-Leistungs-Produkt von 1 pJ. Bei gegebenem Leistungsverbrauch arbeitet die VMOS-Logik mit einer dreifach höheren Geschwindigkeit als übliche MOS-Logik. Die direkte Verbindung zwischen der Source des VMOS-Transistors mit dem Substrat macht eine gesonderte Verbindung überflüssig. Die Struktur erlaubt die Entwicklung von N-Planar-Transistoren und von diffundierten Widerständen als Kopplungs- und Belastungselemente.

Nach Bild 4 ergeben sich vier Verbindungsebenen: Metall, Polysilizium, P-Diffusion und das Substrat selbst für VMOS-Source-Anschlüsse. Diese Eigenschaft ist für den SPP sehr wichtig, weil die Architektur sehr viele innere Verbindungen verlangt. Die Vertikalstruktur und die verschiedenen Arten von inneren Verbindungen erlauben eine Verminderung der Abmessungen um 40 % im Vergleich zu herkömmlichen Strukturen.

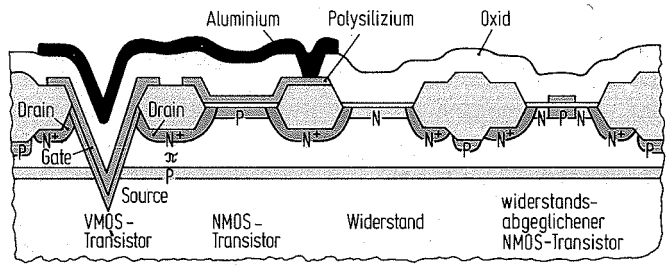


Bild 4. VMOS-Struktur

3 Multiplizierer

Der Multiplizierer des SPP arbeitet nach einem modifizierten Verschachtelungs-Algorithmus. Die Multiplikation wird durch Erzeugung von Teilprodukten und deren Addition ausgeführt (*Bild 5*). Die Teilprodukte können leicht binär gebildet werden, da sie entweder gleich dem Multiplikanden oder Null sind. Der Additionsaufbau ist komplexer, abhängig etwa vom Quadrat der Einzelprodukte. Ein 16-bit-Multiplizierer ist deshalb etwa viermal so komplex wie ein 8-bit-Typ.

Der modifizierte Verschachtelungs-Algorithmus [1] vermindert die Zahl der Teilprodukte auf die Hälfte. Die Grundlage dieses Algorithmus besteht darin, daß eine kontinuierliche Folge der Werte „1" in einem Binärwort mit den Stellen $2^m \ldots 2^n (m \geq n)$ einen Zahlenwert von $2^{m+1} - 2^n$ hat. Zur Erzeugung der Partialprodukte muß daher 2^nx (Multiplikand X) subtrahiert werden, wenn die Folge der Einsen an der Stelle 2^n beginnt, und 2^{m+1}x (Multiplikand X) muß addiert werden, wenn die Folge der Einsen an der Stelle 2^m endet. Der Algorithmus wurde in 3-bit-Gruppen (*Bild 6*) nach den Regeln in *Tabelle 1* implementiert. Es ist zu bemerken, daß für einen 12-bit-Multiplizierer nur sechs Partialprodukt-Terme nötig sind. Tabelle 1 wird leichter verstanden, wenn man sich erinnert, daß verschiedene Kombinationen wechselseitig ausgeschlossen sind. Die Kombinationen 100 und 110 können z. B. nicht zusammen erscheinen, weil der Algorithmus jeweils die Verschiebung um 2 bit verlangt.

Die Eingänge des Multiplizierers sind auf 12 bit begrenzt und der Ausgang auf 16 bit, um die erforderliche Fläche des Chip zu vermindern. Diese Begrenzung ergibt ein Produkt mit einer Genauigkeit von 11 bit im ungünstigsten Fall. Ausführungen höherer Genauig-

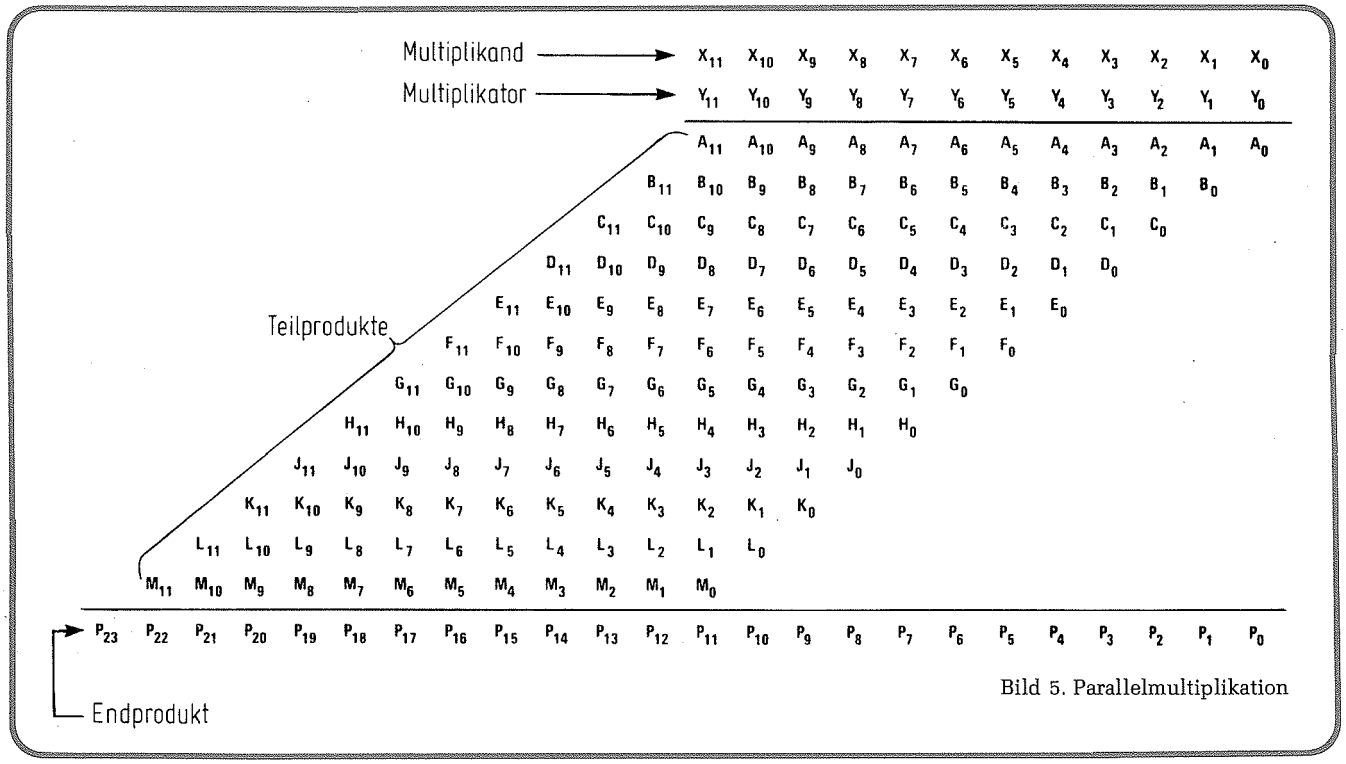


Bild 5. Parallelmultiplikation

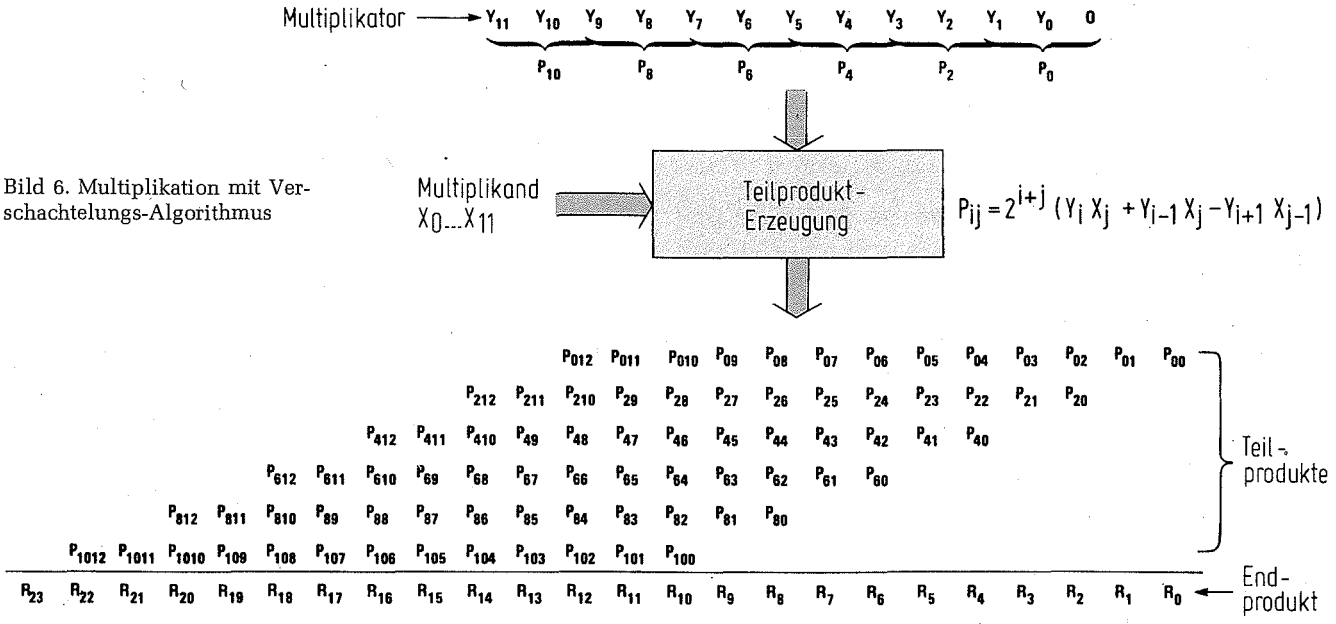


Bild 6. Multiplikation mit Verschachtelungs-Algorithmus

keit sind geplant. Der Multiplizierer bestimmt in erster Linie die Genauigkeit des gesamten SPP.

Die 16 ersten Stellen des Produktes werden verwendet, so daß alle Zahlen als Brüche kleiner als 1 angegeben werden. Der imaginäre Radixpunkt (Komma) steht also auf der linken Seite vor dieser Zahl. Die Darstellung als Bruch und die Festkomma-Arithmetik verlangen also eine gesonderte Stellenbestimmung, um die volle Genauigkeit des SPP auszunutzen. Ein Vorteil dieser Darstellung ist aber, daß kein Überlaufauftritt, da das Produkt von zwei Zahlen, die kleiner als 1 sind, immer kleiner als 1 ist.

Bild 7 zeigt das Blockdiagramm des Multiplizierers. Seine Stromversorgung wird nur stufenweise eingeschaltet. Die Belastungs-Kapazitäten zwischen den Stufen wirken als dynamische Speicher, so daß jede Stufe arbeitet, auch wenn die Versorgung der vorhergehenden Stufe abgeschaltet ist. Um eine sichere Multiplikation zu gewährleisten, stoppt der Oszillator nur nach einer vollständigen Multiplikation. Durch den wechselseitigen Energiebezug wird die zugeführte Energie auf 250 mW gegenüber 650 mW vermindert. Dies ist ein maßgebender Faktor für die gesamte Leistungsreduzierung des SPP.

4 Speicher

Die Anwendung der VMOS-Technik ergibt eine sehr wirkungsvolle Speicherstruktur (*Bild 8*). Um den Leistungsverbrauch des RAM-Bereichs zu vermindern, werden Lastwiderstände aus Polysilizium verwendet. Das RAM selbst arbeitet völlig statisch. Um den Leistungsverbrauch weiter zu senken, werden die Ausgangs-Verstärker nur mit Leistung versorgt, wenn sie gebraucht werden. Der SPP-Oszillator stoppt nur bei Abschluß eines Speicher-Zyklus, um eine einwandfreie Arbeitsweise dieser pseudo-dynamischen Anordnung sicherzustellen. Der Leistungsverbrauch des Datenspeichers wurde um 250 mW gesenkt.

5 Befehlssatz

Das Befehlsformat zeigt *Bild 9*. Es besteht aus zwei Operationsteilen, der Adreßinformation und Operand (falls erforderlich). Nach *Bild 10* sind verschiedene Formate je nach Befehlstyp möglich.

5.1 *OP1-Befehle*

Die OP1-Befehle sind in *Tabelle 2* zusammengestellt. Sie steuern den Datenfluß und manipulieren

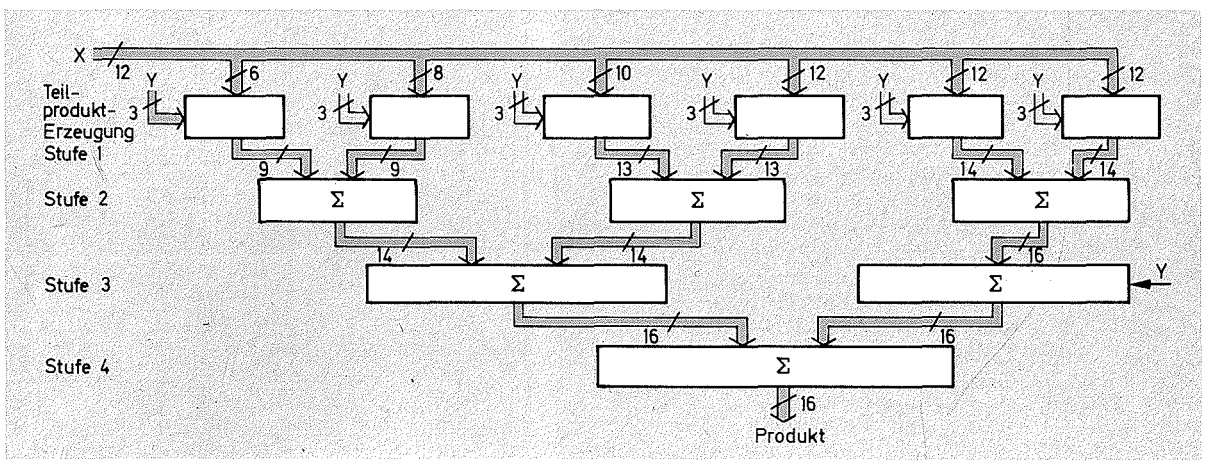


Bild 7. Blockschaltung des Multiplizierers

verschiedene Zähler und Register. Eingeschlossen sind einige Makrobefehle, wodurch die Programmschritte vermindert werden, die bei weitverbreiteten Digital-Algorithmen benötigt würden. Sprünge können durch den Inhalt des Addier-Subtrahier-Werkes bedingt sein.

Wichtig für manche Algorithmen ist die Art der Behandlung des Akkumulator-Überlaufs. Im SPP ist ein Überlaufschutz vorgesehen, wobei der Akkumulator in seinem größten positiven oder negativen Wert vor dem Überlauf verbleibt. Diese Betriebsart ist bei der Implementierung von Digital-Filtern nützlich, aber bei der Anwendung von Modulo-Arithmetik nicht erwünscht. CLAC gibt den Überlauf-Schutz frei, COVP setzt ihn außer Betrieb.

Der SPP muß synchronisiert werden, wenn in Datensystemen mit synchroner Abtastung gearbeitet wird. Dafür sind die Befehle WAIF und WAOF vorgesehen. Sie stoppen den SPP, wenn das korrespondierende Ein- oder Ausgangsregister noch nicht bereit ist, Daten zu übertragen. Die internen Taktimpulse werden bis zur Vervollständigung des Speicherzugriffs und der Multiplizierzyklen unterbrochen, um den Ablauf nicht zu stören. Die normale Operation wird wieder aufgenommen, wenn am Eingangsregister ein neuer Wert ansteht oder das Ausgangsregister ausgelesen wurde. Wenn die Datenübertragung abgeschlossen wurde, bevor der Wartebefehl gegeben wurde, wird einfach zum nächsten Programmschritt weitergegangen.

Externe Einrichtungen können mit dem SPP durch die Befehle LACO und WAOF synchronisiert werden. Der Programmierer fügt mit dem LACO-Befehl eine Scheinausgabe ein, gefolgt von WAOF. Dadurch wird der Interrupt-Request-Ausgang (IRQ) gesetzt und der SPP gestoppt.

Die externe Einrichtung kann dann den SPP wieder anstoßen, wenn ein Daten-Lesebefehl oder ein EXECUTE-Befehl gegeben wird.

5.2 OP2-Befehle

Die OP2-Befehle sind in *Tabelle 3* zusammengestellt. Sie werden zugleich mit den OP1-Befehlen gegeben und steuern das Addier-Subtrahier-Werk. Einige besondere Operationen sind für den Akkumulator vorgesehen. ABS und NEG sind leicht einzusehen. SHR aktiviert das Register für 2-bit-Verschiebung, um eine Division durch 4 zu erreichen. SGV veranlaßt, daß der Akkumulator-Inhalt das Vorzeichen des Inhalts von RAM-V annimmt, d. h. daß der Inhalt des Akkumulators komplementiert wird, wenn die Vorzeichen unterschiedlich sind.

Tabelle 1. Modifizierter Verschachtelungs-Algorithmus (Regeln)

Y_{i+1}	Y_i	Y_{i-1}	Bemerkungen	F_i
0	0	0	Keine „1"-Folge	0
0	0	1	Ende der „1"-Folge bei i-1	+1
0	1	0	1-bit-„1"-Folge	+1
0	1	1	Ende der „1"-Folge bei i	+2
1	0	0	Anfang der „1"-Folge bei i+1	–2
1	0	1	Ende der „1"-Folge und Beginn einer neuen	–1
1	1	0	Anfang einer „1"-Folge bei i	–1
1	1	1	Mitte der „1"-Folge	0

$F_i = Y_i - 2Y_{i+1} + Y_{i-1}$
$P_i = 2^i \cdot F_i \cdot \overline{X}$

Tabelle 2. OP 1-Befehle

Mnemonische Bezeichnung	Operation
NOOP	Keine Operation
Lade-Befehle	
LLTI	Lade-Daten in das Eingangsregister
* LIBL	Lade-Basisregister, Schleifenzähler vom Eingangsregister
LAIX	Inhalt vom Akkumulator in das Indexregister
* LAXU	Inhalt vom Akkumulator in das Indexregister, RAM U
* LAXV	Inhalt vom Akkumulator in das Indexregister, RAM V
LACO	Inhalt vom Akkumulator in das Ausgangsregister, setze Interrupt-Anforderung

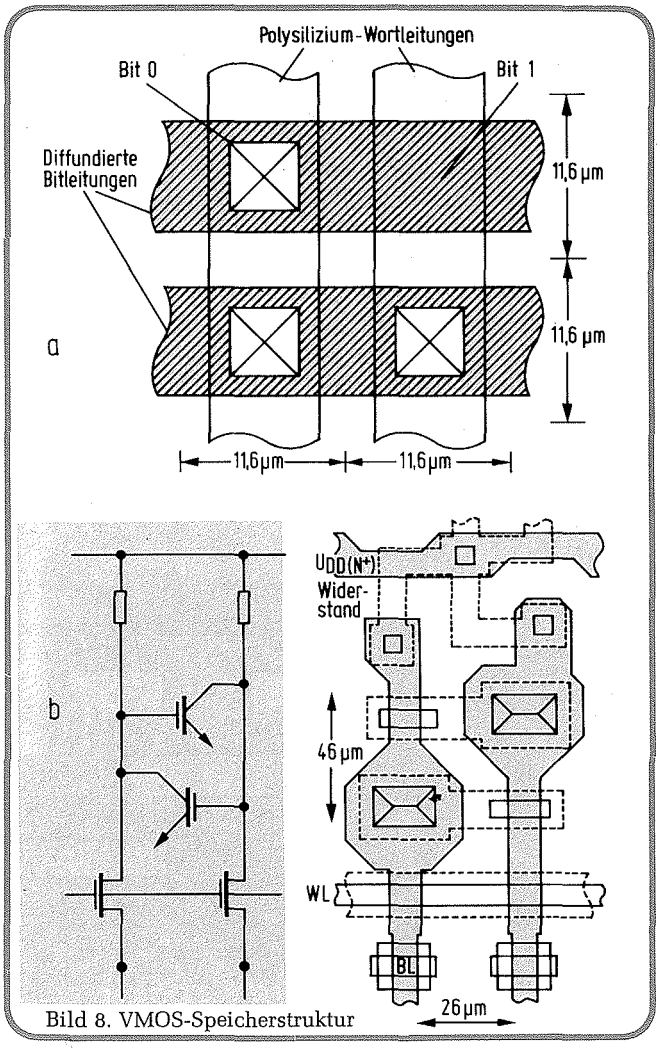


Bild 8. VMOS-Speicherstruktur

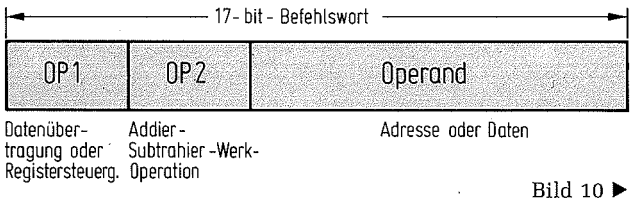


Bild 9. Befehlsformat des SPP

Bild 10 ▶ Befehlstypen

Versatz-Adressierung	OP1	OP2	DU	DV	Typ	0
	5 bit	4 bit	3 bit	3 bit	1 bit	1 bit
Direkte Adressierung	OP	OP	Adresse			1
	5 bit	4 bit	7 bit			1 bit
Direkter Transfer	OP1	OP2	Übertragungs-Adresse			
	5 bit	4 bit	8 bit			
Daten im Adreßfeld	OP1	Datenwort				
	5 bit	12 bit				

Mnemonische Bezeichnung	Operation
Daten-Transport-Befehle	
TACU	Übertrage vom Akkumulator zum RAM U
TACV	Übertrage vom Akkumulator zum RAM V
TIRV	Übertrage vom Eingangsregister zum RAM V
TVPV	Übertrage vom VP-Register zum RAM V
TAVI	Übertrage vom Akkumulator zum RAM V, bei Verwendung des Indexregister-Inhalts als Basis
Register-Befehle	
INIX	Erhöhe das Indexregister
DECB	Erniedrige das Basisregister
INCB	Erhöhe das Basisregister
SWAP	Vertausche die Funktionen von Basis- und Indexregister
Akkumulator-Befehle	
* CLAC	Lösche den Akkumulator
COVP	Lösche den Überlaufschutz
Unbedingte Sprünge	
JMUD	Springe direkt unbedingt
* JMUI	Springe indirekt unbedingt (Index zeigt auf Sprungtabelle)
Bedingte Sprünge	
* JMCD	Springe direkt, abhängig vom Schleifenzähler (Springe, wenn LC ≠ 0, LC wird dekr.)
* JMCI	Springe indirekt, abhängig vom Schleifenzähler
SKNZ	Überspringe den nächsten Befehl, wenn Akkumulator ≠ 0
SKNN	Überspringe den nächsten Befehl, wenn der Akkumulator nicht negativ ist
SKNO	Überspringe den nächsten Befehl, wenn der Akkumulator keinen Überlauf hat
Unterprogramm-Befehle	
JMSR	Springe zum Unterprogramm
RETN	Springe vom Unterprogramm zurück
Komplexe Befehle	
* JCII	Springe bedingt indirekt und erhöhe das Basisregister
* JCDT	Springe bedingt direkt (Zweifach-Adr.), erhöhe Basis- und Indexregister
* JCIB	Springe bedingt indirekt, erhöhe Basis-register und übertrage Inhalt von Register VP nach RAM V
Halt-Befehle	
WAIF	Warte auf das Eingangsflag
WAOF	Warte auf das Ausgangsflag

* Makro-Befehle

Tabelle 3. OP 2-Befehle

Mnemonische Bezeichnung	Operation
NOP	Keine Operation
Akkumulator-Befehle	
ABS	Absolutwert des Akku-Inhalts
NEG	Negieren des Akku-Inhalts
SHR	Akku-Inhalt 2 bit nach rechts schieben
SGV	Akku-Inhalt komplementieren, wenn Polarität vom Inhalt des RAM V abweicht
Addier-Befehle	
AVZ	Addiere RAM V zu Null
AVA	Addiere RAM V zum Akku-Inhalt
AUV	Addiere die Inhalte von RAM U und RAM V
LVI	Lade Inhalt V in den Akkumulator bei Verwendung des Index als Basis (Tabelle)
Subtrahier-Befehle	
SVA	Subtrahiere Akku-Inhalt von RAM V
SVU	Subtrahiere RAM U von RAM V
Multiplizier-/Addier-Befehle	
APZ	Addiere das Produkt und Null
APA	Addiere Produkt und Akku-Inhalt
APU	Addiere Produkt und RAM U
Multiplizier-/Subtrahier-Befehle	
SPA	Ziehe Akku-Inhalt vom Produkt ab
SPU	Ziehe RAM U vom Produkt ab

Tabelle 4. Steuerarten

Mnemonische Bezeichnung	Operation
CLEAR	Rücksetzen der Steuerarten auf NORMAL
RESET	Software-Haupt-Reset. Alle SPP-Register rücksetzen, Start bei 00
EXECUTE	Starte Programmausführung
DATA L/H	Spezifiziere LSB des Datenwortes
DATA U/H	Spezifiziere MSB des Datenwortes
SER. INP.	Gib den Serien-Eingangs-Port frei
SER.OUT.	Gib den Serien-Ausgangs-Port frei
S/M INP.	Bilde Zweierkomplement des Serien-Eingangs
S/M OUT	Übertrage das Zweierkomplement der internen Daten an den Serien-Ausgang
BLOCK	Freigabe der Übertragung von Blockdaten
EXT. ROM	Steuerung des SPP durch externes Befehls-ROM; vor allem zum Testen
ROM VERIFY	Erlaubt das Auslesen des internen ROM-Inhalts zum Testen

Tabelle 5. Ergebnisse von Testprogrammen

Aufgabenstellung	Zahl der Befehle	Ausführungszeit
Filter 2. Ordnung	8	2,4 µs
DTMF-Decodierer	51	54 µs je Abtastung
V.27-Modem (4800 bit/s)	248	497,7 µs/Baud
Kompl. FFT (128 Pkt.)	254	10,7 ms

5.3 Steuerbefehle

Tabelle 4 zeigt die Befehle, die unmittelbar vom steuernden Prozessor kommen. Die Steuerart wird durch vier Adreßleitungen ausgewählt. Der SPP ist somit ein Peripherie-Baustein, der 16 Adressen des Mikroprozessor-Speicherbereichs belegt. Die Steuerarten und der Befehl LIBL ermöglichen eine Echtzeit-Modifizierung des Programms. Auf diese Weise kann ein einziges SPP-Programm für verschiedene Anwendungen benutzt werden. Wird der SPP z. B. als Universal-Digital-Filter programmiert, mit gewählter Grenzfrequenz, Filterart und Datenquelle (über Seriell- oder Parallel-Port), so kann die Ausführungszeit über einen µP gesteuert werden.

6 Adressierungsarten

Der SPP ermöglicht vier Arten des Datenzugriffs. Bei der direkten Adressierung wird die volle Adresse der Daten angegeben. Infolge der begrenzten Befehlslänge kann nur auf ein Datenwort gleichzeitig zugegriffen werden. Bei der Adressierung relativ zur Basis muß das Basisregister durch die Befehlsfolge LLTI/LIBL aufgerufen werden, so daß auf zwei Datenworte gleichzeitig zugegriffen werden kann (U- und V-Versatz werden im Befehl angegeben). Daten können im Zwischenspeicher (*Scratchpad*) gespeichert oder aberufen werden, indem die Adressierungsart hierfür aufgerufen und Scratchpad sowie V-Port mit dem Versatz geladen werden. Auf die V-Port-Daten wird relativ zum Basisregister zugegriffen. Die Scratchpad-Daten werden genauso behandelt wie die Daten des U- und V-RAM-Ports, der 8-Worte-Block des Scratchpad wird dabei anstelle des U-Datenblocks verwendet. Die vierte Methode ist die Zweifachbasis-Adressierung. Sie wird vor allem bei schnellen Fourier-Transformationen und der Matrizenbehandlung verwendet.

7 Vergleichsprogramme

Für den SPP wurde eine Reihe von Vergleichsprogrammen geschrieben, die in *Tabelle 5* zusammengestellt sind. Wenn ein Speichertransfer, eine Addition oder eine Multiplikation als Einzeloperation betrachtet werden, so ermöglicht der SPP durchschnittlich 11 Operationen/µs für das Programm „V.27-Modem". Das bedeutet, daß der Durchsatz des SPP äquivalent zu einem konventionellen µP-System ist, bei dem sehr schnelle Parallel-Multiplizierer mit einer Basis-Zykluszeit von 90 ns benutzt werden. Dieser Durchsatz entspricht dem der besten bipolaren „Bit-Slice"-Systeme bei niedrigeren Kosten, geringerer Gehäusezahl und drastisch vermindertem Leistungsverbrauch. In *Tabelle 6* ist der SPP mit weiteren Kriterien gegenüber anderen Lösungen aufgeführt.

8 Anwenderunterstützung

Da der SPP ein anwenderprogrammierbarer Baustein ist, wird der Unterstützung des Programmierers größte Aufmerksamkeit geschenkt. Z. B. ist jetzt bereits ein kompletter In-Circuit-Emulator erhältlich. Allerdings bedingt die komplexe Struktur des SPP Schwierigkeiten, von denen man den Anwender nicht befreien kann – beispielsweise die Speicherverwaltung, deren Effektivität sehr stark von der Kreativität des Programmierers abhängt. Trotzdem wird im Augenblick an einem Assembler gearbeitet, und es sollen Hilfsmittel wie Editor, Debug-Programm, Floppy-Disk-Speicher und Terminal hinzugekommen. Diese Dinge sollen den Benutzer in die Lage versetzen, sich auf die eigentlichen Probleme zu konzentrieren.

Der SPP in dieser Form ist nur ein erster Schritt. Künftige Produkte werden ein EPROM enthalten und genauere Multiplikationen ausführen, sie werden mehr Speicherplatz haben und noch schneller sein. Aber der Anfang ist gemacht, und der Weg für die Zukunft der digitalen Signalverarbeitung ist aufgezeigt.

Literatur

[1] *Waser, S., Peterson, A.*: Real-time Processing Gains Ground with Fast Digital Multiplier. Electronics, 29. September 1977, S. 93 ff.

[2] *Cooley, J., Tukey, J.*: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation, Bd. 19, H. 90, S. 297...301.

[3] G-AE Subcommittee on Measurement Concepts. What is the Fast Fourier Transform? IEEE Trans. Audio Electroacoustics, Bd. AU-15, Juni 1967, S. 45...55.

[4] *Mulrooney, T.*: Microprogramming a Minicomputer for Fast Signal Processing. Electronics, 16. März 1978, S. 136 ff.

[5] *Schmidt, L.*: What is a Digital Filter? Hewlett-Packard Journal, September 1978, S. 17...18.

[6] *Mauermann, W.*: „Zahlenknacker" entlastet CPU von der Arithmetik. ELEKTRONIK 1978, H. 5, S. 73...74.

Im Gegensatz zum SPP, der ihm in der Geschwindigkeit überlegen ist, führt dieser Baustein kompliziertere math. Berechnungen (z. B. Sinus) durch.

Tabelle 6. Der SPP im Vergleich

Problem	Lösung	Ausführ.-Zeit	ICs	Rel. Kosten	Verlustleistung
V. 27-Modem 4800 bit/s	Bit-Slice-µP	Echtzeit	~35	> 1,75	20 W
	SPP	Echtzeit	6	1	1...2 W
Kompl. FFT 128 Punkte	Minicomp.	20 ms	viele	> 160	> 1000 W
	SPP	11 ms	7	1	1...2 W

Dr.-Ing.
Elmar Groschupf

Die Programmierung von Mikroprozessor-Systemen befindet sich in der Anfangsphase und ist vergleichbar mit der Softwareerstellung für Minicomputer zu deren Anfangszeit. Es ist daher nicht verwunderlich, daß auch die gleichen Erfahrungen gemacht und die gleichen Lernphasen durchlaufen werden. Die im folgenden angestellten Betrachtungen berücksichtigen diese Tatsache.

Regeln für die Erstellung von Mikrocomputer-Software

1 Grundlegende Erkenntnisse

Nur das Zusammenwirken von Hardware und Software kann die Leistungsfähigkeit eines Mikroprozessor-Systems gewährleisten. Es hat sich aber gezeigt, daß Software sehr teuer und schlecht kalkulierbar ist – Softwarekosten entstehen in ganz anderer Höhe und viel unerwarteter als die gewohnten Hardwarekosten. Deshalb beginnt sich allmählich der Gedanke durchzusetzen, daß der Ablauf der Softwareproduktion in Bahnen gelenkt werden sollte, die sich steuern lassen, z. B. durch Auswahl geeigneter Programmiersprachen, Programmiermethoden und Dokumentationsregeln.

2 Einsatzbereich für Mikroprozessoren

2.1 Systemkonfiguration mit Mikroprozessor

Der allgemeine Aufbau eines Mikroprozessor-Systems (typische Wortbreite von 8 bit) geht aus *Bild 1* hervor. Der Mikroprozessor ist an einen sogenannten internen Bus angeschlossen, der nicht über den Bereich des Systems hinausragt. Auf diesen Bus, der neben Daten und Adressen auch Steuersignale führt, greifen noch die Unterbrechungslogik zur Interruptsteuerung, die Kopplungslogik für den seriellen Ein/Ausgabe-Bus und den parallelen Ein/Ausgabe-Bus sowie Programm- und Datenspeicher zu (ROM bzw. RAM).

Zur schnellen Arithmetikverarbeitung kann noch eine „*Arithmetik Processing Unit*" (APU) angeschlossen werden, die Arithmetikfunktionen hardwaremäßig realisiert. Über den seriellen und parallelen Ein/Ausgabe-Bus – letzterer wird auch Anlagenbus genannt – kann der Mikroprozessor an den zu bearbeitenden Prozeß, an Peripheriegeräte wie Fernschreiber oder auch an weitere Mikroprozessor-Systeme angeschlossen werden.

Die Karteneinteilung gilt nur grob für Systeme mittlerer Größe (einige KByte, davon etwa 80 % ROM, der Rest RAM). Bei kleineren Mikroprozessor-Systemen wird nur eine einzige Karte meist im Doppeleuropakarten-Format verwendet.

2.2 Aufgabenrahmen

Die betrachteten Mikroprozessor-Systeme werden alleine oder im Verbund in folgenden Bereichen eingesetzt [1, 2, 3]:

– Steuerungstechnik,
– Regelungstechnik,
– Datenverarbeitung im engeren Sinne.

Die Aufgaben liegen sowohl im Anlagenbereich als auch in der Fernwirktechnik. Dementsprechend breit gefächert stellt sich der Aufgabenrahmen dar.

Einfache Systeme arbeiten zyklisch ein „Stand-Alone"-Programm ab. Wird laufender Eingriff des Bedienpersonals nötig, so muß eine Fernschreibmaschine oder ein Konsolgerät angeschlossen werden. Über Programmier- und Testfunktionen kann Software ohne zusätzliche Geräte implementiert werden. Breiter Datenperipherieanschluß erfordert Verwaltungsaufgaben. Systeme mit hohen Sicherheitsanforderungen benötigen Überwachungs- und Kontrollroutinen. Komplizierte Systeme mit „Multitasking" erfordern ein Betriebssystem mit Betriebsmittel- und Auftragsverwaltung sowie mit Unterbrechungsverar-

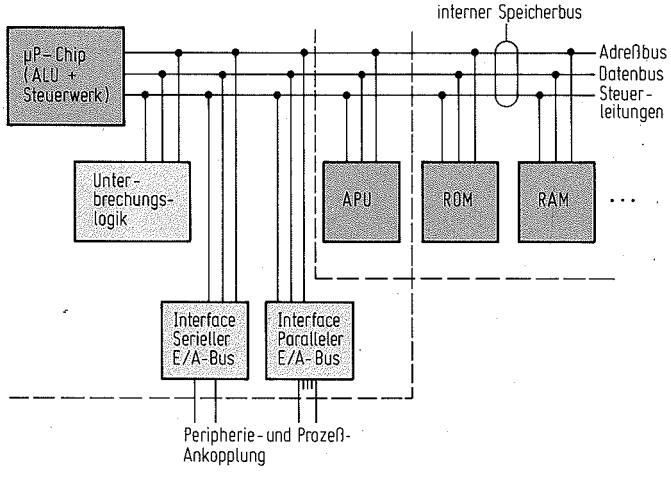


Bild 1. Aufbau eines Mikroprozessor-Systems – Karteneinteilung

beitung. Die Programme reichen von einfachen Logik- oder Arithmetikverarbeitungen, Treiberfunktionen mit und ohne Realzeitverarbeitung über Peripherieverwaltung bis zu Betriebssystemfunktionen mit interaktivem Betrieb.

Nicht zu vergessen ist sogenannte Testsoftware, die zum Prüfen von Mikroprozessor-Systemkomponenten wie z. B. Speicherkarten dient. Diese Software ist nötig, um bei kleineren oder mittleren Stückzahlen Hardwarekomponenten ohne teure Prüfanlagen im Prüffeld zu testen.

Bei allen Einsatzmöglichkeiten muß der wirtschaftliche Einsatzbereich von Mikroprozessor-Systemen gegenüber der Minicomputeranwendung abgegrenzt werden. Es gelten die Kosten/Leistungs-Beziehungen nach *Bild 2* [4]. Daraus geht hervor, daß der Mikroprozessor in kleinen Systemen mit fest umrissenen Aufgaben (Kartencomputer) und im Bereich von Mehrrechnersystemen mit verteilten Aufgaben kostengünstiger ist als der Minicomputer. Letzterer deckt am günstigsten einen Bereich mit nicht zu umfangreichen, variablen und mit einigem Handhabungskomfort ausgerüsteten Problemlösungen ab (Bereich kleiner und mittelgroßer Prozeßrechner). Bei Mehrrechnersystemen fällt der Minicomputer gegenüber dem Mikroprozessor-System wieder zurück, denn bei ersterem steigen die Systemkosten überproportional mit der oft komplizierten Leistungserweiterung an, während bei Mikroprozessor-Systemen Leistungserweiterungen durch gleichartige Zusatzsysteme erreicht werden, die fast lineare Kostensteigerung bewirken. Im Bereich der reinen Minirechner ist der Mikrocomputer dagegen unbedingt unterlegen. Das liegt nicht nur an seiner für ein Einzelsystem geringen Verarbeitungsleistung, sondern auch an dem relativ aufwendig zu erreichenden Komfort an Handhabung und Peripheriebedienung, der in diesem Bereich eine Rolle spielt.

3 Produktionsweg für Mikroprozessor-Software

Der übliche Weg, um Mikroprozessor-Software herzustellen, nämlich aus einer Problemstellung heraus über ein Flußdiagramm Code zu erstellen, der erst nach dem Test oder nach mehreren Änderungen dokumentiert wird, führt dazu, daß nur der betroffene Entwickler seine Arbeit in der Herstellungsphase überblickt. Daß dabei Terminhaltung, Leistungserfüllung und Wartung zum Problem anwachsen, liegt klar auf der Hand.

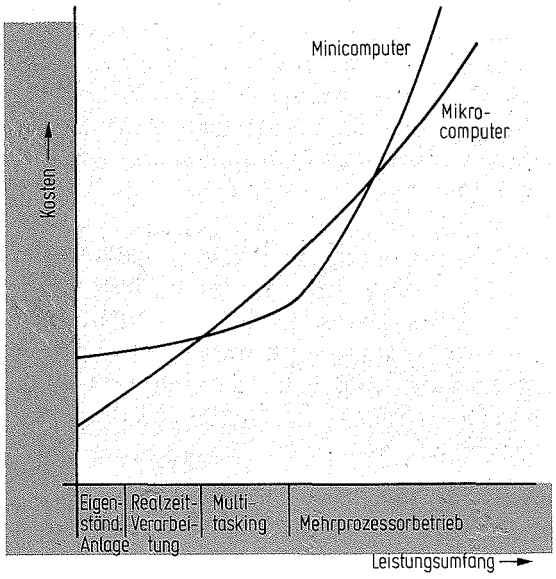
Mikroprozessor-Software muß wie ein technisches Produkt rationell, aufgabengerecht, sicher und termingerecht hergestellt werden können. Dazu muß ein festgelegter und handhabbarer Produktionsgang vorliegen, wie er in *Bild 3* gezeigt wird. Die einzelnen Vorgänge müssen aufgegliedert und analysiert werden, damit sich für jeden Schritt geeignete Arbeitsmethoden bestimmen lassen. So ist es möglich, jedes Hilfsmittel für diesen Arbeitsablauf richtig einzuschätzen. Man erkennt dann zum Beispiel, daß die Methode der oft zitierten strukturierten Programmierung keine umfassende Hilfe bedeutet, da sie hauptsächlich in der Einengung des Umfangs einer Programmiersprache besteht (z. B. Verbot von Sprungbefehlen) und somit nur einen Teil der Problematik betrifft [5, 6, 7].

3.1 *Aufgabenstellung*

Diese Anfangsphase auf dem Weg zur Programmierung ergibt den Aufgabenrahmen für die zu erstellende Software meist in Form eines Pflichtenheftes und zeigt somit auf, was eigentlich geleistet werden muß. Sie wird meist nicht zur eigentlichen Softwareproduktion gerechnet. Das ist jedoch falsch, denn aus einer gut gestellten Aufgabe geht am leichtesten eine schlüssige Lösung und damit ein übersichtliches Programm hervor.

Die Aufgabenstellung muß wirklich umfassend und detailliert sein, denn jede hier unberücksichtigte Einzelheit führt zu späteren Änderungen und Einschränkungen und damit auch zu Terminverzügen. Jede Aufgabe ist in Teilaufgaben aufzuspalten, die in einer Hierarchie angeordnet werden. Je durchdachter und detaillierter die Aufgabenstellung vorliegt, desto leichter lassen sich alle weiteren Produktionsgänge realisieren. Die angesprochene Vorgehensweise führt vom großen, übergeordneten Problem zum feineren, untergeordneten Problem. Sie wird auch als Top-Down-Approach bezeichnet und hat gegenüber anderen Methoden wie z. B. Bottom-Up-Approach den Vorteil, eine durchgängige logische Entwicklung zu erzwingen.

Aufgabenteilungen ermöglichen später modulare Lösungen und führen somit auf natürliche Weise zu



◀Bild 2. Kosten-Leistungs-Vergleich von Mini- und Mikrocomputern

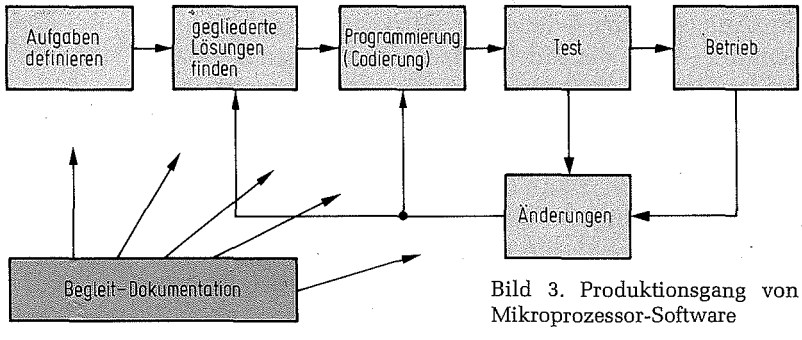


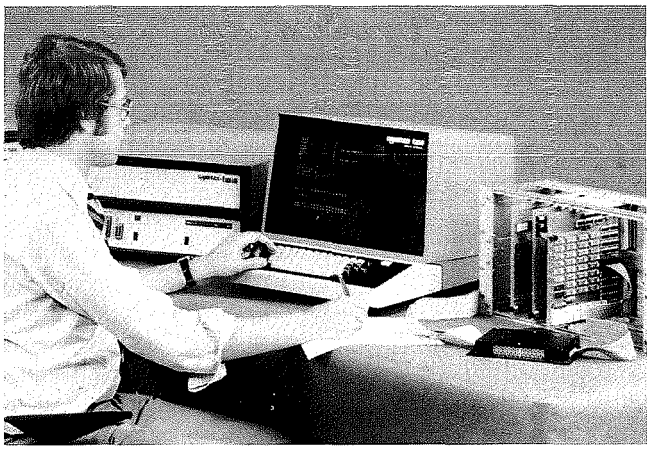
Bild 3. Produktionsgang von Mikroprozessor-Software

blockweiser Programmierung. Daran kann Arbeitsteilung angeknüpft werden.

Bei jeder Teilaufgabe müssen folgende Dinge berücksichtigt werden:

- zu erbringende Leistungen,
- Benutzerschnittstellen,
- Systemkonfiguration,
- Fehlerverhalten,
- Parameterangaben,
- Rechenzeit- und Speicherplatzgrenzen,
- Normierungseigenschaften,
- Sprachauswahl.

Die schriftlich fixierte Aufgabenstellung ist eine erste Dokumentation und sollte als solche den ganzen weiteren Produktionsgang begleiten.



Trotz umfangreicher Hilfsmittel ist der Aufwand für Mikrocomputer-Software immer noch schlecht kalkulierbar

3.2 Problemlösung

Auf der Aufgabenstellung und ihrer Einteilung aufbauend entstehen hier Funktionsbeschreibungen bzw. Funktionsbäume, also Angaben, wie Abläufe stattfinden. In der Regel werden Flußdiagramme erstellt, die für den Entwickler anschaulich und aussagekräftig sind. Interne Konstruktionen, benutzte Algorithmen und Lösungsmethoden müssen in der Dokumentation festgehalten werden.

Es ergibt sich automatisch eine Feingliederung in Funktionsblöcke auf letzter Ebene, die später in Programmblöcke umgesetzt werden können. Es ist dabei sehr nützlich, alle von der Aufgabenstellung übernommenen und jetzt weiterhin anfallenden Daten nach Typ, Umfang, Verwendungszweck und Gültigkeitsbereich zu strukturieren – ebenso wie ein Programm. Meist zeigt sich dabei eine Parallelität von Daten- und späterer Programmstruktur. Das wird z. B. in [8] für eine neue Methode ausgenutzt, Programme nach Datenstrukturen zu erstellen. Daten werden eingeteilt in:

- Variable (mit und ohne Anfangswert),
- Konstante,
- Parameter,
- Zwischenvariable,
- Lauf- oder Zählworte,
- Sprungziele, Unterprogrammnamen.

Man muß auch schon problem- und sprachgerechte Namensgebung praktizieren, die konsequent bis zur später folgenden Codierung gehen muß und die somit die Dokumentation unterstützt.

Aus Funktionsabläufen und Datengliederung ergeben sich festzuhaltende Hinweise auf interne Schnittstellen, Benutzung von dynamischem und statischem Speicher (Stack bzw. Datenspeicher).

Da der Entwickler in der vorliegenden Phase den Funktionsablauf am besten überblickt, lassen sich auch ohne Mühe Hinweise für spätere Funktionstests und Plausibilitätskontrollen festhalten, die später nur mit Mühe erhalten werden können.

Man erkennt hier deutlich das grundsätzliche Vorgehen:

Der ganze Arbeitsablauf wird so detailliert aufgelöst und jede anfallende oder mögliche Einzelheit so ausgenützt, daß sie nicht nur im laufenden, sondern auch in zukünftigen Schritten die Entwicklung logisch unterstützt und gleichzeitig die Dokumentation bestreitet.

Es sei hier auf eine Methode zur Programmkonzipierung verwiesen, auf das sogenannte HIPO-Verfahren (*Hierarchy plus Input-Process-Output*) [9], das eine Unterstützung bei der Software-Produktion durch grafische Darstellung von Programmfunktionen und -daten liefert.

Eine Zeitabschätzung für die Programmierung ist erst in dieser Phase möglich. Ab diesem Produktionsgang einschließlich aller noch folgenden Herstellungsgänge gilt z. B. die Faustregel, daß ein Programmierer effektiv etwa 40 Minuten benötigt, um einen Assemblerbefehl zu formulieren.

3.3 Programmierung

Die bei der Problemlösung gefundenen Funktionsabläufe müssen nun in Programmsprachen formuliert und codiert werden. Diesem Schritt, der eigentlich der kleinste im ganzen Produktionsgang ist, wurde in der Vergangenheit die größte Aufmerksamkeit geschenkt, wie man an den zahlreichen Diskussionen über Programmiersprachen erkennt [10].

Selbstverständlich hat die Programmiersprache großen Einfluß auf die Softwareproduktion. Eine Fach- oder Hochsprache*) ermöglicht oft direktes Abschreiben des Lösungsweges während der Assemblerprogrammierer die zu realisierenden Funktionsabläufe erst mit einiger Erfahrung in maschinenabhängige Befehlsfolgen umsetzen kann. In jedem Falle aber gilt die Regel, daß ein gutes und funktionsgerechtes Programm um so schneller erstellt werden kann, je gründlicher die Punkte der Aufgabenstellung und der Problemlösung durchdacht wurden.

Ist die Problemlösung in Blöcke unterteilt, so besteht auch das Programm aus verschiedenen Moduln, die sich funktionsgerecht aneinanderreihen und aufrufen lassen. Bei komfortabler Softwareproduktion können bereits vorhandene Moduln aus Dateien oder Bibliotheken benutzt werden, so daß hier Programmierarbeit gespart wird und die Aufgabenteilung die ersten Früchte tragen kann.

*) Fachsprachen unterscheiden sich von universellen Hochsprachen durch einen fachbezogenen und damit eingeschränkten Sprachschatz (z. B. für Steuerungstechnik).

Der Programmausdruck stellt im allgemeinen die ganze hier anfallende Dokumentation dar. Bei Programmiersprachen mit schlechter Aussagekraft wie bei Assemblersprache darf der Programmierer nicht mit Kommentar geizen! Dieser sollte dabei an die Lösung und nicht an den Programmvorgang anknüpfen. Ebenso sollten hier so viele Adreßmarken (*Label*) wie möglich verwendet werden, um Übersicht und spätere Testmöglichkeiten zu verbessern.

Es fallen bei der Programmierung weitere Hinweise für spätere Tests, besonders Teilsystemtests ab. Außerdem sollten alle Hinweise über Registerbenutzung, gültige Speicherbelegung sowie Peripheriebedienung dokumentiert werden.

3.4 Test und Wartung

Zum Testen der erzeugten Software gibt es verschiedene Möglichkeiten, die alle genutzt werden müssen:

a) Funktionstests einzelner Programmblöcke mit Test der inneren Vorgänge in diesen Blöcken;
b) Funktionstest des ganzen Systems mit implementierten Programmblöcken.

Da jeder neu implementierte Programmblock auch dann Fehler im System bedingen kann, wenn er für sich alleine ausgetestet ist, folgt nach jedem Blocktest ein neuer Systemtest.

Bei allen Untersuchungen müssen beachtet werden:

- Funktionskontrolle,
- Schnittstellenkontrolle,
- Zeitverhalten,
- Sicherheitsbestimmungen,
- Fehlerbehandlung.

Nicht implementierte Blöcke oder nicht angeschlossene Systemkomponenten können in der Testphase durch Simulation ersetzt werden, z. B. durch Dummyroutinen bzw. externe Testinterfaces.

Die Erkenntnisse, die beim Testen gewonnen werden, fließen in die Punkte Problemlösung und Programmierung ein. Bei ungenügender Aufgabenstellung ist auch diese betroffen. Da jetzt durch Wiederholungen der letzten Vorgänge in der Softwareproduktion neue Versionen des Produkts entstehen, ist es wichtig, die zugehörige Dokumentation auf dem aktuellen Stand zu halten. Änderungen müssen also in allen Stufen der Dokumentation berücksichtigt werden. Dieser Vorgang belastet zwar den Entwickler, aber er ist nötig um spätere Wartungskosten in Grenzen zu halten.

Daraus ergibt sich, daß für die Softwarewartung der gleiche Produktionsablauf wie für Software allgemein gilt. Der Vorgang des Testens und die Rückkopplung von Änderungen auf die Punkte Problemlösung und Programmierung ist dabei besonders wichtig. Daß der Zugriff zu aussagekräftiger Dokumentation und deren Aktualisierung Entwicklungskosten spart, liegt auf der Hand.

Stehen die zur Softwareproduktion benutzten Hilfsmittel wie z. B. Entwicklungssystem bei der Wartung nicht zur Verfügung, so muß in dem zu warten-den System entsprechende Hilfssoftware zur Programmierung und zum Testen vorhanden sein, was bei Mikroprozessoren nicht unbedingt selbstverständlich ist.

3.5 Gefahren auf dem Produktionsweg

Zwischen den Punkten Aufgabenstellung und Problemlösung, also zwischen der Objektbeschreibung der zu realisierenden Leistungen („was") und der Funktionsbeschreibung des Lösungsweges („wie") klafft eine Lücke. Hier fehlt noch eine Methodik mit stetigem Übergang. In diese Lücke stößt die Intuition des Entwicklungsingenieurs und bewirkt die Vielfalt der Lösungswege. Hilfen wie rechnerunterstützte Programmierung, Precompiler oder anwendungsorientierte Fachsprachen schließen diese Lücke nicht, denn diese Hilfsmittel gehen vom Punkt der Programmierung aus und unterstützen den Entwickler in Richtung Problemlösung.

Analoges gilt für die „strukturierte Programmierung", die lediglich die Programmierung unterstützt, indem sie durch Einschränkung im Sprachgebrauch Tricks beim Programmieren verhindert und überschaubare Programmblöcke liefert [5].

Auch die Anwendung von Hochsprachen erleichtert nur die Verbindung der Punkte Programmierung und Problemlösung. Sie verhindert allerdings viele Fehlermöglichkeiten, die bei Assemblersprache möglich und üblich sind, und erhöht somit die Sicherheit und die Wartbarkeit der Software. Hierauf wird noch gesondert eingegangen.

Wird Assemblersprache eingesetzt, so ist wegen der Maschinenabhängigkeit der Sprache die Wissensspanne der Entwickler besonders groß, da in der Regel alle Produktionsgänge für Mikroprozessor-Software vom gleichen Personenkreis bearbeitet werden. Die Gefahr der Problemferne ist dann besonders groß und die Software gerät zum Kunstwerk einiger Entwickler.

Den genannten Gefahren begegnet man am besten, wenn die einzelnen Produktionsgänge in ihrem Ablauf streng getrennt und detailliert ausgeführt werden. Vorgehen und Dokumentation müssen zwingend nach festen Regeln erfolgen.

Ganz besonders ist darauf zu achten, daß die Dokumentation zu jedem Punkt sauber, vollständig und konsequent ist, auch dann, wenn sich der Entwickler belastet fühlt. Alle Dinge, die bisher bei den einzelnen Produktionsgängen besonders herausgestellt wurden, müssen in die Dokumentation eingehen – egal, ob sie gerade für wichtig gehalten werden oder nicht. Kommentare in Quellprogrammen alleine genügen nicht, denn sie sagen oft nur einen Teil der Überlegungen aus, der häufig auch nicht der entscheidende Teil war. Nur so erspart man sich bei Test und Wartung, also bei kostenintensiven Punkten teures Neudurchdenken aller Vorleistungen.

4 Hilfsmittel zur Programmierung

Zur Quellprogrammerstellung stehen verschiedene Arten von Programmiersprachen zur Verfügung. Quellprogramme führen in der Regel durch Überset-

zung mit Hilfe eines Übersetzerprogramms (z. B. Assembler oder Compiler) zu Maschinenprogrammen, die durch block- oder schrittweise kontrollierbares Abarbeiten auf ihre Funktionsfähigkeit geprüft werden. Diese Vorgänge können auf dem Mikroprozessor-System selbst oder auf einem fremden System ablaufen. Im letzteren Fall muß das Zielsystem zum vollständigen Test oder auch zur Programmimplementation an das Fremdsystem angeschlossen werden. Enthält das Fremdsystem den gleichen Mikroprozessor wie das Zielsystem, dann wird ersteres auch Entwicklungssystem genannt. Ist das Fremdsystem dagegen ein andersartiger Rechner (meist ein Großrechner), so fungiert er als Gastrechner und die für die nötigen Abläufe bereitstehende Software heißt Cross-Software.

4.1 Programmiersprachen

Stehen keine Übersetzerprogramme zur Verfügung, dann muß in der einfachsten und mühsamsten Art direkt in Maschinensprache programmiert werden, d. h. der Objektcode wird durch Aneinanderreihen der entsprechenden Bitkombinationen erzeugt. Einziges Hilfsmittel ist also ein PROM-Programmiergerät mit Eingabe.

Bequemer ist ein Zeilenassembler, ein Umsetzer, der es ermöglicht, das Programm wenigstens zeilenweise (meist auch befehlsweise in Mnemocode (Assemblerbefehle) anzugeben. Der Zeilenassembler bildet daraus durch sofortiges Umsetzen den Maschinencode, jedoch ohne Symbolangabe oder Referenzen für Adressen und Daten, er kann lediglich Syntaxfehler bemerken.

Symbolbenutzung, Vor- und Rückwärtsreferenzen sowie Übersetzungssteuerung durch Bedingungsangaben für den Übersetzer sind erst bei Assemblerprogrammen möglich. Bis zu dieser Stufe sind die Quellprogramme noch sehr maschinenabhängig.

Das ändert sich, wenn Hochsprachen wie z. B. PEARL oder FORTRAN benutzt werden. Hier sind mehrstufige Umsetzerprogramme nötig (Compiler). Da die Sprachmittel meist arithmetikähnliche Ausdrucksweise erlauben, kommen die Quellprogramme der ingenieursmäßigen Denkweise sehr entgegen. Sie sind außerdem maschinenunabhängig und lassen sich bei vorhandenen Compilern leicht auf verschiedene Systeme übertragen (Portabilität).

Fachsprachen wie z. B. DOLOG von AEG-Telefunken oder STEP3 von Siemens – beide für die Steuerungstechnik – benötigen zur Objektcodeerzeugung spezielle Umsetzer, die ein Quellprogramm in fachspezifischer Notation erwarten. Diese Fachsprachen sind zur Erzeugung von Anwendersoftware für Ingenieure gedacht, die an sich mit der eigentlichen Programmierung nichts zu tun haben sollen. Solche Fachsprachenprogramme können nicht nur durch Makroübersetzer in Objektcode gewandelt werden; sie können auch durch laufende Interpretation mit einem Interpreterprogramm zu laufend abzuarbeitenden Maschinenbefehlsfolgen führen.

Weiterführende Übersetzer wie Precompiler oder Entwurfssprachen (Stichwort „Interaktive Programmierung") werden hier nicht behandelt, da sie für den Mikroprozessor-Programmierer in der Regel nicht zur Verfügung stehen. Die Zusammenstellung in der *Tabelle* soll die Vor- und Nachteile der verschiedenen Sprachen verdeutlichen.

Da Assemblersprachen und Hochsprachen die verbreitetsten Hilfsmittel sind, werden diese nochmals miteinander verglichen: *Bild 4* veranschaulicht für Erstellungszeit, Testzeit, Statementanzahl, erzeugte Objektcodelänge und Programmlaufzeit die mit Assemblersprache und mit der Hochsprache PL/M für Mikroprozessor-System AEG ALU 80 mit Intel 8080 erhaltenen Ergebnisse. Die Werte wurden aus mehreren Programmen jeweils in der Größe von etwa 2 KByte ermittelt.

Vergleicht man die bisherigen Angaben, so lassen sich folgende Aussagen zur Auswahl der Programmiersprache treffen [10]:

- Maschinensprache wird nur benutzt, wenn keine Hilfsmittel vorliegen (z. B. Teststart ohne Hilfsmittel).
- Assemblersprache ist zu empfehlen bei zeit- und speicherplatzkritischen Programmen (Realzeitverarbeitung, Treiber und Systemsoftware). Infolge hoher Programmierkosten und -zeiten aber effektiver Programme wird Assemblersprache besonders bei Systemen mit hohen Stückzahlen verwendet.
- Hochsprachen sollten bei größeren Programmlängen, kleineren Stückzahlen und besonders im Bereich der Anwendersoftware benutzt werden. Das gilt verstärkt für Anwendungen mit Arithmetik.
- Fachsprachen sollten dort zum Einsatz kommen, wo Hochsprachen nicht effektiv genug sind, aber die

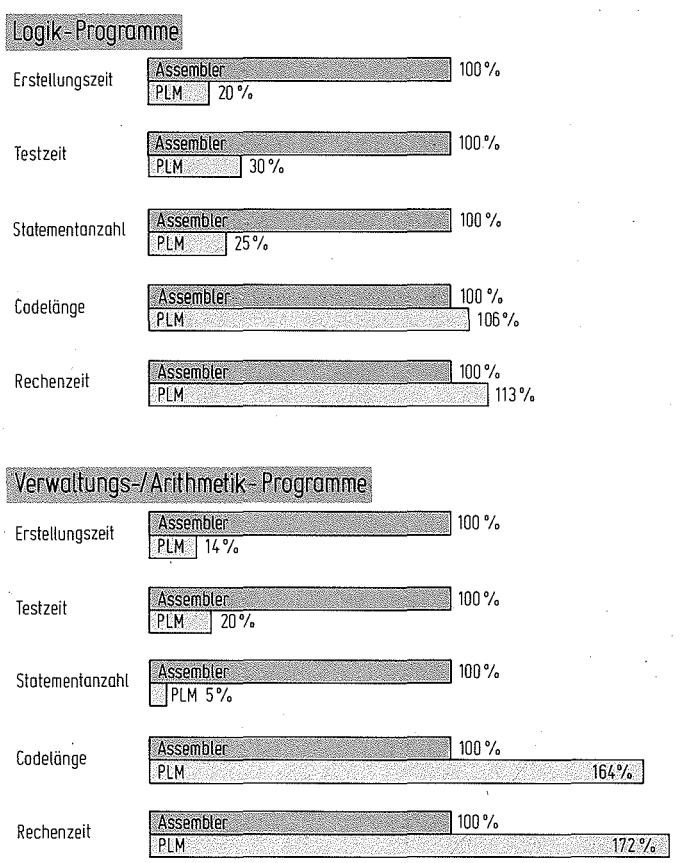


Bild 4. Vergleich von Assembler- und Hochsprache am Beispiel der Programmierung von AEG-ALU 80

Anwender dennoch problemnah und maschinenunabhängig arbeiten wollen.

4.2 Cross-Software

Wird die Programmentwicklung auf einem Gastrechner durchgeführt, so muß dessen Software folgende Dienste umfassen:

- Texterstellung für Quellprogramme,
- Übersetzerprogramme (Assembler, Compiler) mit relativ adressiertem Codeergebnis,
- Testmöglichkeiten (Simulator),
- Dokumentation aller Entwicklungsvorgänge,
- Bibliothekshaltung von Programm-Moduln,
- Bindemöglichkeit von Programm-Moduln,
- Ausgabe von implementierbaren Objektcode-Moduln für das Mikroprozessor-Zielsystem.

Die als Gastrechner benutzten Großcomputer – Minirechner arbeiten hier zu uneffektiv – stellen dazu den ganzen Komfort einer Großrechneranlage zur Verfügung, so daß umfangreiche Texthaltung, Dialogbetrieb und Dokumentation ermöglicht werden. Außerdem können mehrere Entwickler quasi parallel ihre Aufgaben bearbeiten, ohne jeweils ein ganzes System mit Quellprogrammerstellung, Übersetzung und Programmtest zu blockieren. Auf einer Großrechneranlage läßt sich sämtliche Cross-Software bequem auf ganze Mikroprozessorfamilien ausdehnen. Bibliotheksbetrieb ermöglicht es, einmal entwickelte Programmteile, wie z. B. Routinen für Gleitpunktarithmetik, immer wieder zu benutzen und allgemein zugänglich zu machen. Bindefähigkeit muß dazu natürlich gegeben sein.

Wird ein Simulator zur Verfügung gestellt, dann kann ein übersetztes Programm zweckmäßigerweise im effektiven Dialogbetrieb auf seinen fehlerfreien Ablauf überprüft werden. Der Testbetrieb sollte die Möglichkeit bieten, zwischen RAM- und ROM-Bereich im simulierten Speicher zu unterscheiden.

Beim Testbetrieb zeigt sich der Nachteil der Cross-Software. Alle Tests können hier nur auf die Software beschränkt werden, da die Hardware des zu untersuchenden Mikroprozessor-Systems nicht vollständig und auch nicht mit allen seinen Realzeiteigenschaften nachgebildet werden kann. Der Simulator bildet nur den Mikroprozessor-Chip und den Speicher nach und nicht die speziellen Randbausteine wie z. B. Peripherie-Controller. Damit entfallen für die Cross-Software Testmöglichkeiten für Software zu Rechnerkopplung, Realzeitverarbeitung und Peripheriebedienung.

Tabelle der Vor- und Nachteile verschiedener Programmiersprachen

Sprache	Vorteile	Nachteile
Maschinensprache	kein Umsetzer nötig effektiver Code direkt testbar leichte Programmänderung ohne Übersetzer möglich	keine Lesbarkeit keine Dokumentation größte Fehlermöglichkeit keine Symbolverarbeitung sehr zeitraubende Programmierung keine Portabilität Programme nicht verschiebbar
Assemblersprache	Symbolverarbeitung effektiver Code keine Syntaxfehler möglich Programmänderungen ohne Umsetzer noch möglich Makroprogrammierung oft möglich verschiebbare Programme oft mögl. Bindbarkeit oft möglich	geringe Lesbarkeit Assemblerprogramm nötig noch sehr maschinennah und somit problemfern geringe Dokumentation problembezogene Fehler häufig langsame Programmierung keine Portabilität
Hochsprache	gute Lesbarkeit schnelle Programmerzeugung problemnahe Programmierung gute Dokumentation automatisch keine Syntaxfehler möglich problembezogene Fehler sehr stark eingeschränkt Programme verschiebbar Bindemöglichkeit gegeben Portabilität nur an Compiler gebunden	Compiler nötig (hoher Speicheraufwand) uneffektiver Code Programmänderungen nur mit vorliegender Dokumentation und Neuübersetzung möglich
Fachsprache	sehr problemnahe Programmierung gute Dokumentation schnelle Programmerzeugung Lesbarkeit gegeben keine Syntaxfehler auch problembezogene Fehler eingeschränkt Portabilität mit Übersetzer gegeben	Übersetzer oder Interpreter nötig nur fachgebundener Einsatz bei Interpretation geringer Verlust an Effektivität

Dennoch wird Cross-Software vorteilhaft benutzt für Erstellung, Übersetzung und Vortest besonders von umfangreichen Programmen. Endgültige Tests mit der nötigen Hardwarenähe können dann am Mikroprozessor-System selbst oder auf entsprechenden Entwicklungssystemen durchgeführt werden.

4.3 Entwicklungssystem

Ein Entwicklungssystem bietet folgende Möglichkeiten:

- Quellprogrammerstellung,
- Übersetzung (Assembler, Compiler),
- Binden von Programm-Moduln,
- Testen von Programmen und Hardware.
- PROM-Programmierung.

Zum Testen wird das betreffende Mikroprozessor-System an das Entwicklungssystem angeschlossen, wobei letzteres Teile des ersten (Speicher, Prozessor-Chip) teilweise oder ganz nachbilden kann. Damit ist es möglich, Programme zusammen mit den Hardwarekomponenten sowohl statisch als auch dynamisch zu testen, was besonders bei Realzeitverarbeitung wichtig ist. Für den an das Mikroprozessor-System anzukoppelnden Prozeß muß jedoch ein Prozeßsimulator angeschlossen werden (z. B. Eingaben an Prozeßbus, Unterbrechungseingänge). Die anzuschließende Peripherie kann mit in den Testlauf einbezogen werden. Ein solches System ist jedoch mit jedem einzelnen Vorgang wie Programmerstellung oder Übersetzung voll belegt, da kein Timesharing möglich ist. Die Möglich-

keiten des Entwicklungssystems werden dann optimal genutzt, wenn Vorarbeiten wie Texthaltung und Programmübersetzung mittels Cross-Software erledigt werden, so daß nur die langwierigen Tests und Korrekturen übrigbleiben. Dazu ist es von Vorteil, wenn die Übersetzer der Cross-Software die gleichen Operationscode-Ergebnisse bringen wie diejenigen des Entwicklungssystems.

Für Servicezwecke oder Pflege von Software auf Anlagen vor Ort, also in Verbindung mit angeschlossenem Prozeß ist das Entwicklungssystem allerdings wegen seines noch großen Geräteumfanges nicht geeignet. Hier können nur wenig komfortable Programmiergeräte mit Maschinenprogrammeingabe oder entsprechende Hilfssoftware im Mikroprozessor-System selbst helfen.

4.4 Autarkes Mikroprozessor-System

Soll das Mikroprozessor-System ohne fremde Hilfsmittel in der Lage sein, Programmierung, Test und Dokumentation (mit Hilfe eines Konsolgerätes) durchzuführen, so wird entsprechende Hilfssoftware benötigt, die diese Leistungen ermöglicht. Solche Dienste werden in einem sogenannten Bediensystem zusammengefaßt, das folgende Funktionen bereitstellen muß:

- Zeilenassembler, } Primitivprogrammierung
- Rückübersetzer, } Dokumentation
- Lochstreifen stanzen, } Dokumentation
- Lochstreifen lesen,
- Lochstreifen vergleichen,
- EPROM programmieren
- EPROM-Inhalt vergleichen mit Vorlage,
- Speicherinhalt anzeigen und ändern,
- Speicherbereich mit Konstante beschreiben, } Programmtest
- Speicherinhalt verschieben, } Programmtest
- verschobenes Programm umrechnen, (Sprungziele) } Programmtest
- Register anzeigen und ändern, } Programmtest
- Sedezimal-Rechnung, } Programmtest
- Programmstart mit Haltepunktmarkierungen. } Programmtest

Neben diesen Funktionen, die einen Speicherbereich von etwa 3 KByte verbrauchen, stehen dem Anwendungsprogrammierer damit auch Hilfsroutinen zur Verfügung wie Dual/ASCII-Umsetzung und Treiberfunktionen, die er in seinen Programmen benutzen kann.

Zum reinen Programmierbetrieb ist der Komfort solcher Bediensoftware natürlich gering, jedoch im Servicefall für Test, Änderungen und Pflege genügt er meistens. Normalerweise stellt ein Mikroprozessor-System selbst keine Hardwarehilfe zur Verfügung, die es ermöglicht, Tests neben normalem Prozeßbetrieb zu fahren. Diese Möglichkeit nutzt man bei Minicomputern und Prozeßrechnern häufig; bei Mikroprozessor-Systemen der heute üblichen Größe hat man sie noch nicht aufgegriffen. Damit fällt auch die Notwendigkeit für Interruptverarbeitung in der Bediensoftware weg, so daß diese sehr einfach wird.

5 Ausblick

Der Trend auf dem Mikroprozessorgebiet weist von 8-bit-Prozessoren zu 16-bit-Typen, die effektivere Befehle besitzen und mehr Randbaugruppen wie z. B. Interrupt-Controller und eventuell auch EPROM-Speicher auf dem Prozessor-Chip vereinigen. Die Verarbeitungsleistung und auch der Anwendungsbereich werden damit größer. Dennoch ist der Einsatzrahmen für Mikroprozessoren fest abgesteckt. Auch der zu erwartende Einsatz der Mikroprozessoren in Mehrrechnersystemen wird ihre Aufgaben nicht erweitern, sondern die Konfigurationen der zu bewältigenden Aufgaben ändern. Man sollte sich hüten, Mikroprozessor-Systeme mit Minicomputern konkurrieren zu lassen. Es lassen sich viele Erfahrungen aus dem Minirechnerbereich bei Mikroprozessoren verwenden, aber stets bleibt die Domäne des Mikroprozessors im Bereich fest umrissener, unveränderlicher Aufgaben.

Eine Parallele dazu liefert auch der Personenkreis der Anwender von Mikroprozessoren. Im Bereich der Software arbeiten hier meist Entwicklungsingenieure aus der Hardware oder der Anlagentechnik, die nur zögernd Methoden aus dem Gebiet mittlerer oder größerer Rechnersysteme annehmen. Hier ist zu hoffen, daß mit der Zeit ein Verschmelzen der Erfahrungen und Techniken aus der Informationsverarbeitung und der Elektronik stattfindet, zugunsten optimaler Ausnutzung der angebotenen Mikroprozessoren.

Literatur

[1] *Düll, E. H.*: Die Anwendung von Mikroprozessoren zum Steuern und Rechnen. Die Anwendung von Mikroprozessoren in der Meß-, Steuer- und Regelungstechnik. VDE-Verlag, Berlin 1977.

[2] *Balling, H.*: Die Anwendung von Mikroprozessoren zum Regeln. Die Anwendung von Mikroprozessoren in der Meß-, Steuer- und Regelungstechnik. VDE-Verlag, Berlin 1977.

[3] *Freyberger, F.*: MSR-Systeme mit Mikrorechnern und ihre Konzeption. Die Anwendung von Mikroprozessoren in der Meß-, Steuer- und Regelungstechnik. VDE-Verlag, Berlin 1977.

[4] *Monrad-Krohn, L. M.*: Neue Trends in der Prozeßrechnerhardware. Regelungstechnische Praxis 1977, H. 2, S. 36...42.

[5] *Schuchmann, H. R.*: Strukturierte Programmierung – ein pragmatischer Ansatz für eine umfassende Software-Technologie. Elektronische Rechenanlagen 1975, H. 1, S. 35...39.

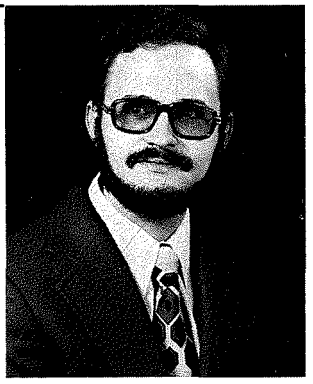
[6] *Hackstein, Schmitz*: Strukturierte Programmierung und ihre graphische Interpretation. ONLINE 13 (1975), S. 660...664.

[7] *Heubach, F.*: Strukturierte Programmierung auch bei Mikrocomputern. ELEKTRONIK-Sonderheft II, Mikroprozessoren/Software, S. 118...123. Franzis-Verlag, München 1977.

[8] *Jackson, M.*: Structured program design technique. Infotech. International 1976.

[9] *Melekian, N.*: Neue Methoden und Techniken der Programmierung, Teil 7: Methodische Programmentwicklung (2). IBM Nachrichten, 26. Jahrgang (1976), H. 230, S. 147...155.

[10] *Ogdin, C. A.*: µC programming languages: When use which and why? EDN, H. 20, November 1977.

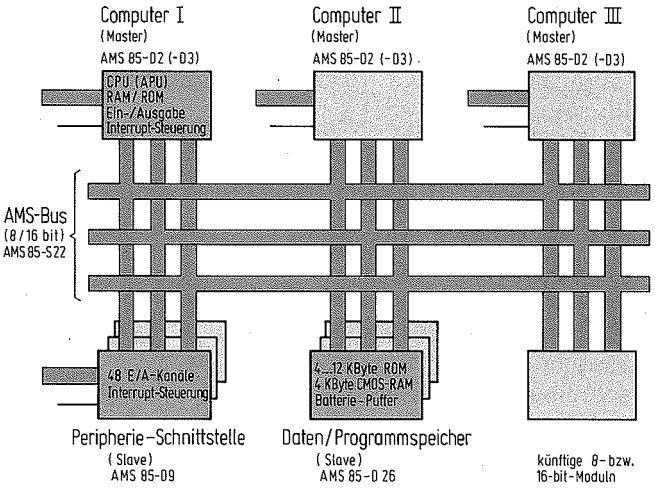


Dr.-Ing. Elmar Groschupf, gebürtiger Siebenbürger, studierte Nachrichtentechnik in Darmstadt (Diplom 1970). Nach einem Jahr praktischer Hardwareentwicklung von Prozeßrechnerperipherie am Institut für Technische Kernphysik der TH Darmstadt trat er in einen Sonderforschungsbereich der gleichen Hochschule ein und promovierte dort mit Simulationsuntersuchungen für HGÜ. Seit 1976 beschäftigt er sich bei AEG-TELEFUNKEN in Seligenstadt mit Grundsoftware für Mikroprozessor-Systeme – zuletzt als Abteilungsleiter. Hobbys: Tischtennis, Geologie, Familie mit 2 Kindern
ELEKTRONIK-Leser seit 1969

Ein 8- und 16-bit-Mikrocomputer-Baugruppensystem für Multicomputer-Anwendungen

Mikrocomputer-Baugruppensysteme sind zur Zeit überwiegend als reine 8-bit-Systeme konzipiert.
Das Baugruppensystem AMS, das von der Firma Siemens vorgestellt wurde, ermöglicht auf Grund seiner Konzeption auch den Aufbau von 16-bit-Mikrocomputern. Die Konstruktion sowohl des AMS-Bus als auch aller AMS-Baugruppen ist von vornherein darauf gerichtet, daß heutige 8-bit-Funktionseinheiten mit künftigen 16-bit-Moduln uneingeschränkt in einer Systemkonfiguraton zusammen arbeiten können. Darüber hinaus bietet AMS die Möglichkeit eines leistungsfähigen Multicomputerbetriebes. Hierbei können mehrere AMS-Computer (Master) innerhalb eines Systems völlig unabhängig voneinander einzelne Aufgaben oder im Verbund miteinander jeweils verschiedene Aufgabenteile eines Gesamtproblems bearbeiten. Diese Master-Baugruppen können je nach Anforderung mit Hilfe sogenannter intelligenter oder reiner Slave-Baugruppen zu einem komplexen hierarchischen Computersystem erweitert werden.
Die AMS-Baugruppen sind im Doppel-Europa-Format gehalten und mit direkten Steckverbindungen versehen. Damit wird den besonderen Anforderungen des europäischen Industrie-Standards entsprochen, der sowohl ungenormte Aufbaumaße als auch die anfälligeren direkten Steckerverbindungen, die oft bei US-Fabrikaten üblich sind, in aller Regel ablehnt.

□ Hersteller: Siemens AG, ZVW 104, Postf. 103, 8000 München 1.



Floppy-Disk-Speicher für doppelte Aufzeichnungsdichte

Unter der Bezeichnung RX02 kündigt die Firma Digital Equipment ein Disketten-Doppellaufwerk an, das mit einseitig beschriebenen Floppy-Disks einfacher und doppelter Dichte betrieben werden kann. Es ist für den Einsatz mit PDP-8- und PDP-11-Rechnern sowie dem Mikrocomputer LSI 11 konzipiert und entspricht in der Einschubgröße dem vorangegangenen Modell RX01. Da Disketten mit einfacher und doppelter Dichte gemischt verwendbar sind, können Datensätze unter Programmsteuerung auf dem System von einfacher in doppelte Dichte transferiert werden. Für den Betrieb mit verschiedenen Zentraleinheiten steht eine Reihe neuer Software-Pakete zur Verfügung. Das System führt beim Einschalten einen automatischen Selbsttest aus. Die Daten werden durch einen Spannungsausfallschutz gesichert, der die Zentraleinheit „warnt", wenn der Controller einen zunehmenden Spannungsabfall im Floppy-Disk-System feststellt. Das Laufwerk hat eine durchschnittliche Suchzeit von 180 ms und eine Spur-zu-Spur-Positionierzeit von 6 ms. Die Datenübertragungsrate beträgt 62 kByte/s. Das System wird ab 11 000 DM angeboten.

□ Vertrieb: Digital Equipment GmbH, Wallensteinplatz 2, 8000 München 40, Tel. (0 89) 35 03-64 47.

Dipl.-Ing.
Günter Schmitt

Programme für Mikrocomputer werden in der Industrie heute nahezu ausschließlich mit Hilfe von Assemblern erstellt. Der Grund liegt vor allem darin, daß man gegenüber höheren Programmiersprachen im allgemeinen kürzere und schnellere Programme erhält. Wie der Assembler den Maschinencode erzeugt, beantwortet dieser Beitrag.

Aufgabe und Arbeitsweise eines Assemblers

1 Problemstellung

In zunehmendem Maße werden technsiche Geräte durch Mikrocomputer gesteuert. Das Programm, das den Arbeitsablauf festlegt, liegt in einem Nur-Lese-Speicher (ROM). Es besteht aus Befehlen und konstanten Daten in Form von digitalen Signalen, die durch den Mikroprozessor ausgeführt werden. Ein Lese-Schreib-Speicher (RAM) nimmt die variablen Daten auf, Ein-Ausgabe-Bausteine verbinden den Mikrocomputer mit dem Gerät. Beim Programmieren sind die Befehle und konstanten Daten sowie ihre Adressen festzulegen. Das Programm muß in Form von digitalen Signalen vorliegen, wenn man es in einem Entwicklungssystem testet oder über entsprechende Geräte die endgültigen ROMs herstellt.

Einfache Lernmodelle arbeiten mit einer *binären* Programmeingabe. Über Kippschalter – sie sind meist mit 0 und 1 beschriftet – gibt man das Programm binär codiert ein; so z. B. mit 00111101 für die Verminderung des Akkumulators um 1.

Einfache Bausätze arbeiten mit einer *sedezimalen* Programmeingabe. Dazu faßt man jeweils vier Bit der binären Codierung zu einer Sedezimalziffer zusammen. In einer Liste findet der Programmierer für den Befehl die entsprechenden zweiziffrige Sedezimalzahl, so z. B. 3D für die Verminderung des Akkumulators um 1. Die Adressen und Konstanten legt er ebenfalls sedezimal fest. Die Eingabe kann in dieser Form auch über einen Fernschreiber oder ein Sichtgerät erfolgen.

Wesentlich bequemer ist die *symbolische* Eingabe. Hier verwendet man leicht merkbare Bezeichnungen anstelle der sedezimalen Befehle und Adressen, z. B. die Zeichen DCR A als Abkürzung für *DeCR*ementiere *A*kkumulator anstelle von 3D bzw. 00111101. Für die Übersetzung der Symbole in den Maschinencode benötigt man jedoch ein Programm, den Assembler [1]. Entwicklungssysteme für Mikroprozessoren enthalten residente Assembler, die die symbolischen Programme sofort übersetzen und speichern, so daß sie anschließend getestet werden können. Cross-Assembler laufen auf einer Großrechenanlage und erzeugen eine Übersetzungsliste und Lochstreifen, die dann zur weiteren Eingabe dienen. Die nächste Stufe ist die Programmierung in einer problemorientierten Sprache ähnlich FORTRAN oder PL/1 [2].

Bild 1 zeigt als Beispiel ein einfaches Programm in Assembler-Schreibweise auf einem Entwurfs-Formular, *Bild 2* zeigt die entsprechende Übersetzungsliste eines Cross-Assemblers.

2 Aufgaben des Assemblers

2.1 Aufbereitung der Eingabe

Bei der Eingabe des Programms über einen Fernschreiber oder ein Sichtgerät steht jede Anweisung auf einer neuen Zeile (Bild 1). Im Namensfeld kann man den Daten und Befehlen frei wählbare symbolische Namen geben. Das Operationsfeld enthält festgelegte Kennwörter für Befehle, Datenvereinbarungen und Assembler-Anweisungen. Im Operandenfeld stehen entweder Konstanten oder die Namen der angesprochenen Speicherstellen oder bei Sprungbefehlen die Namen von Befehlen (Sprungmarken). Das Bemerkungsfeld nimmt Erläuterungen des Programmierers auf. Da bei den meisten Assemblern keine feste Spalteneinteilung vorgeschrieben ist, müssen zunächst die vier Felder voneinander getrennt werden. Nach dieser Aufbereitung stehen der Name, die Operation und der Operand getrennt zur weiteren Verarbeitung zur Verfügung.

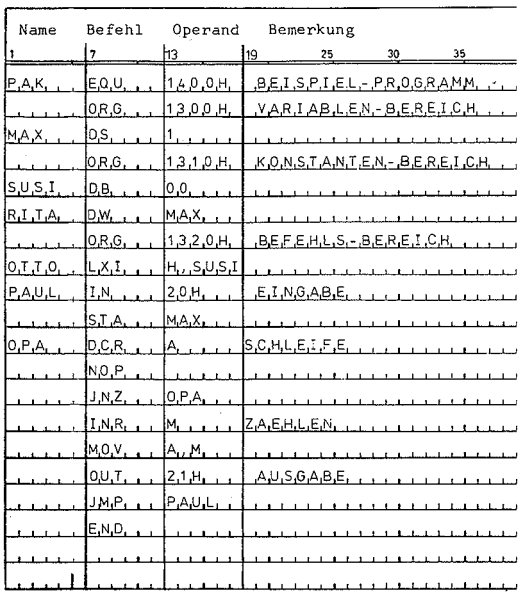
2.2 Aufbau der Namensliste

Der Programmierer vergibt im Namensfeld symbolische Adressen, die der Assembler mit Hilfe eines Adreßzählers in sedezimale Adressen übersetzt. Der Adreßzähler zählt von einem einstellbaren Anfangswert aus die Daten- und Befehlsadressen weiter. Die Namensliste enthält die symbolischen Namen und den Stand des Adreßzählers. Sie kann auf Wunsch auch ausgedruckt werden. *Bild 3* zeigt die Namensliste des Beispiels.

2.3 Verarbeitung des Operationsfeldes

Durch Suchen in einer fest vorgegebenen Symboltabelle stellt der Assembler fest, ob der Operationsteil einen Befehl, eine Datenvereinbarung oder eine Assembler-Anweisung enthält. Hat der Programmierer ein nicht vereinbartes Symbol benutzt, so erscheinen Fehlermeldungen.

Enthält das Operationsfeld einen Befehl, so ist der entsprechende sedezimale Wert einzusetzen. In einigen Fällen muß dazu noch der Operandenteil mit herangezogen werden, so z. B. bei Befehlen, die auf verschiedene Register angewendet werden können oder die verschiedene Adressierungsarten zulassen. *Bild 4* zeigt die Befehlsliste für das Beispiel.



Name	Befehl	Operand	Bemerkung
PAK	EQU	1400H	BEISPIEL-PROGRAMM
	ORG	1300H	VARIABLEN-BEREICH
MAX	DS	1	
	ORG	1310H	KONSTANTEN-BEREICH
SUSI	DB	0,0,	
RITA	DW	MAX,	
	ORG	1320H	BEFEHLS-BEREICH
OTTO	LXI	H,SUSI	
PAUL	IN	20H	EINGABE
	STA	MAX	
OPA	DCR	A	SCHLEIFE
	NOP		
	JNZ	OPA	
	INR	M	ZAEHLEN
	MOV	A,M	
	OUT	21H	AUSGABE
	JMP	PAUL	
	END		

◀ Bild 1. Symbolisches Programm auf einem Entwurfs-Formular

```
ISTEN-ASSEMBLER INTEL 8080 VERSION 1

 1.                      PAK   EQU   1400H   BEISPIEL-PROGRAMM
 2.                            ORG   1300H   VARIABLEN-BEREICH
 3.   1300               MAX   DS    1
 4.                            ORG   1310H   KONSTANTEN-BEREICH
 5.   1310  00           SUSI  DB    00
 6.   1311  00           RITA  DW    MAX
             13
 7.                            ORG   1320H   BEFEHLS-BEREICH
 8.   1320  21           OTTO  LXI   H,SUSI
             10
             13
 9.   1323  DB           PAUL  IN    20H     EINGABE
             20
10.   1325  32                 STA   MAX
             00
             13
11.   1328  3D           OPA   DCR   A       SCHLEIFE
12.   1329  00                 NOP
13.   132A  C2                 JNZ   OPA
             28
             13
14.   132D  34                 INR   M       ZAEHLEN
15.   132E  7E                 MOV   A,M
16.   132F  D3                 OUT   21H     AUSGABE
             21
17.   1331  C3                 JMP   PAUL
             23
             13
18.                            END
```

▲ Bild 2. Übersetzungsliste eines Cross-Assemblers

Im Zusammenhang mit Datenvereinbarungen und Direktoperanden setzt der Assembler sedezimale Konstanten in Daten- bzw. Befehls-Bytes ein. Der Programmierer kann sie im Operandenfeld in beliebiger Form (dezimal, dual, oktal, sedezimal, Zeichen, Rechenausdruck) eingeben, die Umrechnung auf sedezimale Werte führt der Assembler durch. *Bild 5* zeigt Beispiele für verschiedene Eingaben, die alle die sedezimale Konstante 41 ergeben.

Assembler-Anweisungen steuern den Übersetzungsvorgang und werden nicht in Befehls- oder Daten-Bytes übersetzt. Dazu zählen Anweisungen, die

- den Adreßzähler auf einen Anfangswert setzen,
- einen bestimmten Wert unter einem frei wählbaren Namen in die Namensliste eintragen,
- die Art der Ausgabe (Papiervorschub, Namensliste, Zugriffstabellen, Lochstreifen usw.) steuern.

2.4 *Verarbeitung des Operandenfeldes*

Enthält das Operandenfeld symbolische Befehls- oder Datenadressen, so muß der Assembler die von ihm aufgebaute Namensliste (Abschnitt 2.2) durchsuchen und die entsprechenden sedezimalen Adressen in den Befehl einsetzen. Dies ist das Hauptproblem. Da das Programm zeilenweise verarbeitet wird, können symbolische Adressen, die erst später definiert werden (Vorwärtssprünge, Konstanten am Ende des Programms) zum Zeitpunkt ihres Auftretens im Operandenfeld noch nicht übersetzt werden, da die entsprechende Eintragung in der Namensliste fehlt. Deshalb arbeiten die meisten Assembler in zwei Durchläufen. Der erste Durchlauf baut die Namensliste auf, der zweite setzt dann für die symbolischen Adressen die sedezimalen Werte ein. Das bedeutet in den meisten Fällen, daß das Programm auf einem externen Speicher (Band/Platte) zwischengespeichert werden muß.

▼ Bild 3. Namensliste eines Cross-Assemblers

NAMENSLISTE	
PAK	1400
MAX	1300
SUSI	1310
RITA	1311
OTTO	1320
PAUL	1323
OPA	1328

Befehl		Code
DCR	A	3D
IN		DB
INR	M	34
JMP		C3
JNZ		C2
LXI	H	21
MOV	A,M	7E
NOP		00
OUT		D3
STA		32

```
MAX    DB   65          DEZIMAL
OTTO   DB   01000001B   DUAL
PAUL   DB   101Q        OKTAL
RITA   DB   41H         SEDEZIMAL
SUSI   DB   'A'         ZEICHEN
LILI   DB   60+5        RECHENAUSDRUCK
```

▲ Bild 5. Eingabe von Konstanten

◀ Bild 4. Auszug aus der Befehlsliste eines Cross-Assemblers

2.5 *Makros*

Komfortable Assembler bieten die Möglichkeit, oft benötigte Programmteile in einem Makro-Befehl zu definieren und beliebig oft durch den Assembler in das Programm einbauen zu lassen. Die Makros können bei der Definition offene Stellen enthalten, die erst beim Aufruf durch aktuelle Werte ersetzt werden. *Bild 6* zeigt ein Beispiel für die Definition und den Aufruf eines solchen Makros. Der Assembler muß also die Makrodefinitionen besonders speichern und beim Aufruf in den Code einbauen.

2.6 *Externe Unterprogramme*

Die Unterprogrammtechnik bietet die Möglichkeit, ein bestimmtes Programmstück mehrmals anzuspringen und anschließend wieder an die Stelle des Aufrufs zurückzukehren (*Bild 7*). Die Übersetzung eines internen Unterprogramms zusammen mit dem Hauptprogramm bietet keine Schwierigkeit. Unterprogrammname und Rücksprungadresse werden wie die anderen Namen über die Namensliste verarbeitet. Oft übersetzt man jedoch die Unterprogramme getrennt vom Hauptprogramm oder entnimmt sie einer Programmbibliothek. Dann sind der Unterprogrammname bzw. die Rücksprungadresse in dem jeweils zu übersetzenden Programm nicht definiert. Das zweite Problem ist die Anordnung der verschiedenen Programme, die meist relativ zu einer willkürlichen Anfangsadresse adressiert sind. Ein besonderes Bindeprogramm (*Linker, Linkage-Editor*) setzt die fehlenden Adressen ein und nimmt eine gemeinsame Neuadressierung (*Relocation*) der Programmteile vor. Diese Arbeit kann auch in einem dritten Durchlauf vom Assembler ausgeführt werden. Vorher sind jedoch alle Adressen, mit denen später diese Verschiebung durchzuführen ist, besonders zu markieren.

3 Praktische Arbeit mit dem Assembler

Am Anfang stehen die genaue Aufbereitung des Problems, die Entwicklung eines Programmablaufplans und die Erstellung des Assembler-Programms, Quellprogramm genannt. Bei der Verwendung eines Entwicklungssystems [3, 4] wird dann das Quellprogramm – oft mit Hilfe eines Editors – über einen Fernschreiber (TTY) oder ein Datensichtgerät eingegeben und zunächst gespeichert (*Bild 8*). Der Editor ist ein Programm, das die Eingabe und Änderung von Quellprogrammen erleichtert. Anschließend übersetzt der Assembler das Quellprogramm in den Objektcode, d. h. er erstellt das sedezimale Maschinenprogramm. In der ersten Entwicklungsphase wird man sich oft nur die Übersetzungsliste ausgeben lassen und mit Hilfe des Editors das Quellprogramm so lange verbessern, bis es keine formalen Fehler mehr enthält. Anschließend folgt der Test auf logische Fehler mit Hilfe des Entwicklungssystems oder mit einem Prototyp des zu entwickelnden Gerätes. Schließlich kann das fertige Objektprogramm auf einem Lochstreifen ausgegeben werden, der dann zur Herstellung des ROMs mit dem endgültigen Programm dient.

Bei der Verwendung eines Großrechners (*Bild 9*) wird wieder das Quellprogramm mit Hilfe eines Editors in einer Hilfsdatei abgelegt oder direkt über Lochkarten eingegeben. Der Cross-Assembler erzeugt dann das Objektprogramm in einer Datei und gibt die Übersetzungsliste aus. Den Test auf logische Fehler kann hier ein Simulationsprogramm (Simulator) durchführen, das die Arbeitsweise des Mikrocomputers nachbildet und den Programmablauf protokolliert. Eine Lochstreifenausgabe stellt dann die Verbindung zum Entwicklungssystem her oder dient zur endgültigen Programmierung der ROMs.

Literatur:

[1] *Barron, D. W.*: Assembler und Lader. Carl Hanser Verlag, München 1970.

[2] *Koch, G. R.*: 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 1976, H. 3, S. 74...85.

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

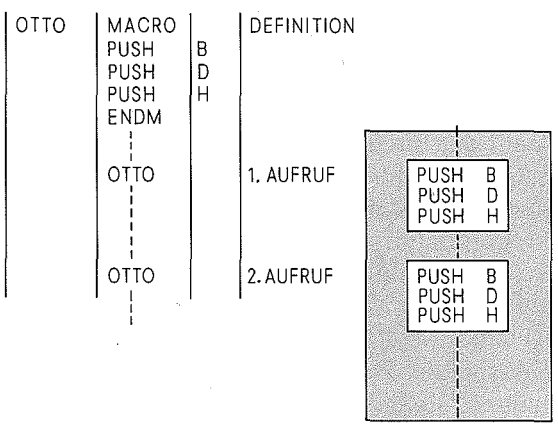


Bild 6. Definition und Aufruf eines Makros

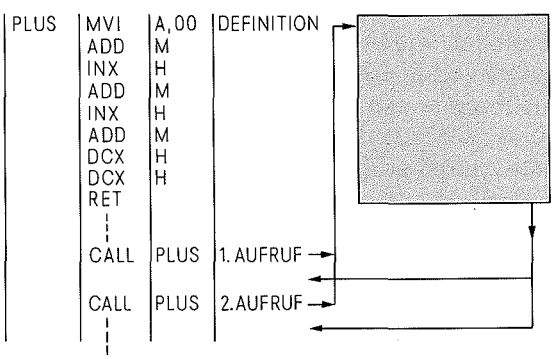


Bild 7. Definition und Aufruf eines Unterprogramms

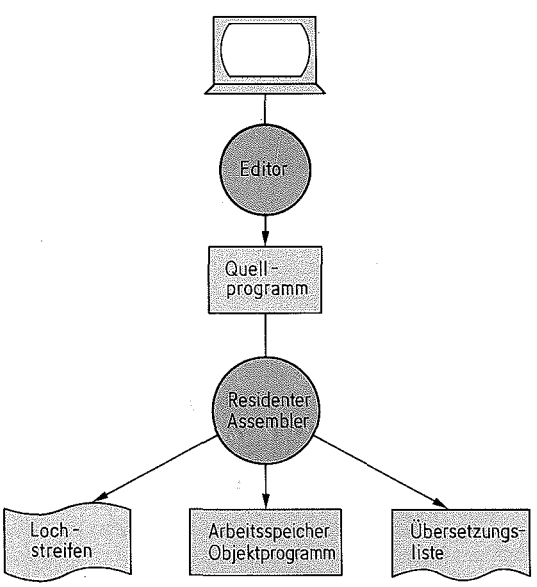


Bild 8. Ablauf der Programmerstellung mit residentem Assembler eines Entwicklungssystems

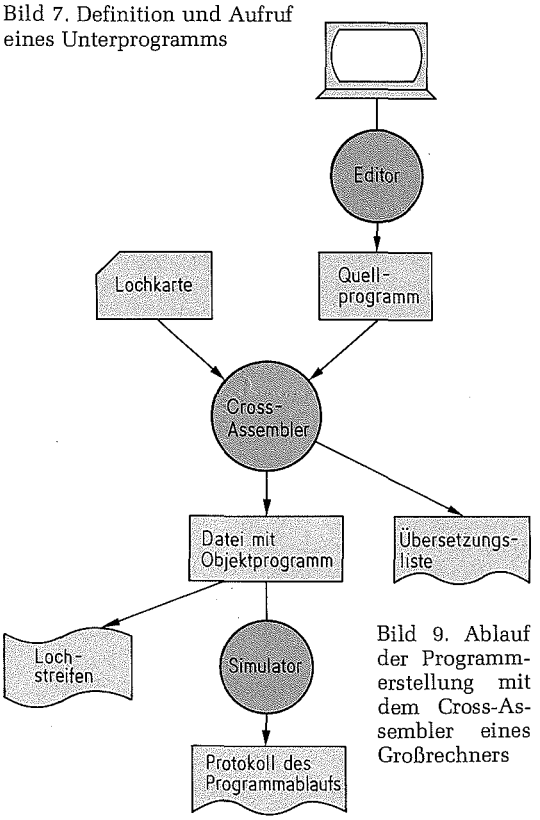


Bild 9. Ablauf der Programmerstellung mit dem Cross-Assembler eines Großrechners

Dipl.-Ing.
Günter Schmitt

Cross-Assembler sind Programme, die auf einem Großrechner laufen und den mnemonischen Code der Assembler-Sprache eines anderen Rechners (z. B. Mikroprozessor) in dessen Maschinencode übersetzen [1, 2]. Anhand eines Cross-Assemblers für den Mikroprozessor 8080 wird im folgenden ihre Arbeitsweise erklärt.

Einfacher Cross-Assembler für Mikroprozessoren

1 Aufgabenstellung

Der hier vorgestellte Cross-Assembler wurde vom Verfasser an der Fachhochschule der DBP Dieburg entwickelt und läuft dort auf einer Großrechenanlage UNIVAC 1108. Er wurde in der allgemein verfügbaren Programmsprache FORTRAN geschrieben und so kurz und einfach wie möglich gehalten. Die symbolischen Bezeichnungen der Firma Intel wurden im wesentlichen übernommen; für die Assembler „Motorola M6800" und „KIM-1", die nach dem gleichen Verfahren arbeiten, waren größere Umstellungen in der Eingabe erforderlich.

Ein Assembler hat folgende Aufgaben:

1. Vergabe von sedezimalen Adressen für Daten und Befehle mit Hilfe eines Adreßzählers,
2. Aufbau des Datenbereiches,
3. Übersetzung der symbolischen Befehle in sedezimalen Code durch Suchen in einer fest eingegebenen Befehlsliste,
4. Übersetzung von frei wählbaren symbolischen Adressen in sedezimale Adressen durch Suchen in einer von ihm selbst aufzubauenden Namensliste,
5. Ausgabe von Fehlermeldungen bei fehlenden Adressen oder nicht definierten Befehlen.

Bild 1 zeigt ein Beispiel für die symbolische Eingabe, *Bild 2* zeigt die vom Assembler ausgegebene Umwandlungsliste.

2 Eingabevorschriften

Für die Eingabe ist folgende Spalteneinteilung fest vorgeschrieben (Bild 1):

Ab Spalte 1: symbolische Namen
Ab Spalte 7: Befehl oder
Assembler-Anweisungen ORG, EQU, END oder
Datenvereinbarungen DS, DB, DW
Ab Spalte 13: Operand bzw. symbolische Adressen
Ab Spalte 19: Bemerkungen

Daten und Befehle können symbolische Namen erhalten, die aus maximal sechs Zeichen bestehen dürfen. Die symbolische Speicheradresse im LXI-Befehl darf jedoch nur aus vier Zeichen bestehen. Die symbolischen Bezeichnungen der Befehle entsprechen denen der Firma Intel.

Die Assembler-Anweisung ORG veranlaßt den Assembler, die folgenden Daten bzw. Befehle von einer bestimmten sedezimalen Anfangsadresse ab fortlaufend anzuordnen. Dies geschieht durch einen Adreßzähler, der jedem Byte eine sedezimale Adresse zuweist. Im Operandenfeld ist vor die vierstellige Sedezimalzahl der Buchstabe H zu setzen.

Mit der Assembler-Anweisung EQU ist es möglich, für symbolische Adressen, die in dem zu übersetzenden Programmteil nicht vorkommen, beliebige vierstellige sedezimale Werte zu vereinbaren. Im Operandenfeld sind vor die vierstellige Sedezimalzahl die Buchstaben EX zu setzen.

Die Assembler-Anweisung END steht am Ende des Programms. Sie beendet den Übersetzungsvorgang.

Die Datenvereinbarung DS reserviert eine beliebige Anzahl von Bytes. Im Operandenfeld steht der Buchstabe D vor der Dezimalzahl.

```
Name    Befehl  Operand  Bemerkung
1       7       13       19  25  30  35  40  45
                         BEISPIEL FUER UMWANDLUNGSLISTE
PUMP    EQU     EX4711
        ORG     H0100    VARIABLEN-BEREICH
IST     DS      D10      10 BYTES VARIABLEN
        ORG     H1300    KONSTANTEN-BEREICH
SOLL    DB      50       ZAHLENKONSTANTE
        DW      SOLL     ADRESSKONSTANTE
        ORG     H1310    BEFEHLS-BEREICH
ZEIT    DCR     C        VERMINDERE C-REGISTER UM 1
        JZ      PUMP     SPRINGE BEI NULL
        LDA     IST      LADE IST WERT
        CPI     50       VERGLEICHE MIT SOLLWERT
        JZ      ZEIT     SPRINGE BEI NULL
        LOLA    SOLL     FEHLERHAFTER CODE
        JZ      OPA      NICHT DEFINIERTE ADRESSE
        END
```

Bild 1. Symbolische Eingabe

```
LISTEN-ASSEMBLER INTEL 8080

  1.                                          BEISPIEL FUER UMWANDLUNGSLISTE
  2.                          PUMP   EQU   EX4711
  3.                                 ORG   H0100     VARIABLEN-BEREICH
  4.          0100           IST    DS    D10       10 BYTES VARIABLEN
  5.                                 ORG   H1300     KONSTANTEN-BEREICH
  6.          1300  50       SOLL   DB    50        ZAHLENKONSTANTE
  7.          1301  00 13           DW    SOLL      ADRESSKONSTANTE
  8.                                 ORG   H1310     BEFEHLS-BEREICH
  9.          1310  OD       ZEIT   DCR   C         VERMINDERE C-REGISTER UM 1
 10.          1311  CA 11 47       JZ    PUMP      SPRINGE BEI NULL
 11.          1314  3A 00 01       LDA   IST       LADE ISTWERT
 12.          1317  FE 50          CPI   50        VERGLEICHE MIT SOLLWERT
 13.          1319  CA 10 13       JZ    ZEIT      SPRINGE BEI NULL
 14.CODE                            LOLA  SOLL      FEHLERHAFTER CODE
 15.U23       131C  CA OP          JZ    OPA       NICHT DEFINIERTE ADRESSE
 16.                              END

                              symbolische Eingabe

                    sedezimaler Code

              Stand des Adreßzählers sedezimal

       Fehlermeldungen

  Eingabezeile
```

Bild 2. Umwandlungsliste

Im Operandenteil der Befehle steht entweder eine fest vereinbarte Registerbezeichnung oder eine frei wählbare symbolische Adresse oder ein Direktoperand in Form einer zweistelligen Sedezimalzahl ohne Zusatz.

Die Datenvereinbarung DB legt eine zweistellige sedezimale Konstante von einem Byte Länge ab. Sie steht ohne Zusatz im Operandenfeld.

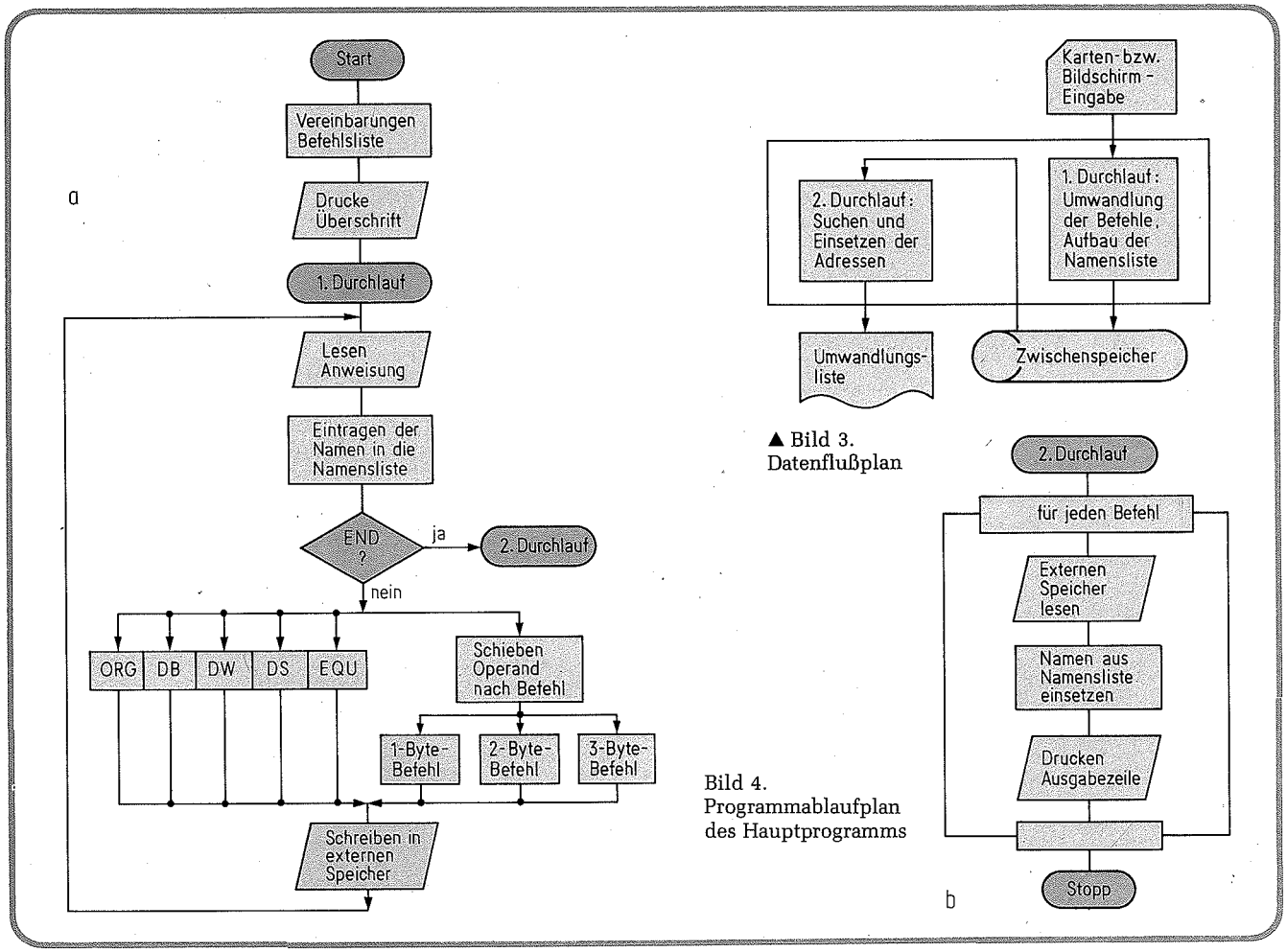
Die Datenvereinbarung DW legt eine Adreß-Konstante von zwei Byte Länge ab. Im Operandenfeld steht eine symbolische Adresse.

3 Hauptprogramm

Die Anweisungen werden im ersten Durchlauf einzeln von Lochkarten oder einem Bildschirmgerät gelesen, bis auf die sedezimalen Adressen umgewandelt und auf einer Platten- bzw. Trommeldatei zwischengespeichert. Im zweiten Durchlauf durchsucht der Assembler die Namensliste, setzt die sedezimalen Adressen ein und gibt die Umwandlungsliste aus. *Bild 3* zeigt den Datenflußplan, *Bild 4* einen vereinfachten Programmablaufplan und *Bild 5* das FORTRAN-Hauptprogramm.

Name, Befehl und Operand werden in je einem Wort aus sechs Zeichen gespeichert und getrennt verarbeitet. Bei 1-Byte-Befehlen ohne Operandenteil durchsucht der Assembler nur die Befehlsliste und setzt den sedezimalen Code ein. Bei 1-Byte-Befehlen mit Operandenteil müssen die Registerbezeichnungen des Operandenteils in den Befehlscode eingebaut werden. Ein Unterprogramm verschiebt Teile des Operanden in den Befehlsteil, das Hauptprogramm durchsucht dann die Befehlsliste nach diesen zusammengesetzten Symbolen und setzt den sedezimalen Code ein.

Bei 2-Byte-Befehlen mit einer Konstanten im Operandenteil setzt der Assembler in das erste Byte den



▲ Bild 3. Datenflußplan

Bild 4. Programmablaufplan des Hauptprogramms

Code aus der Befehlsliste und in das zweite Byte die sedezimale Konstante ein. Bei 2-Byte-Befehlen mit einem Register und einer Konstanten im Operandenteil müssen die Registerbezeichnungen in den Befehlscode eingebaut werden. Ein Unterprogramm verschiebt Teile des Operanden in den Befehl, das Hauptprogramm setzt in das erste Byte den Code aus der Befehlsliste und in das zweite Byte die sedezimale Konstante ein.

Bei 3-Byte-Befehlen mit einer Speicheradresse im Operandenteil setzt der Assembler in das erste Byte den Code aus der Befehlsliste und in das zweite und dritte Byte die sedezimale Adresse aus der Namensliste ein. Dabei stehen die werthöchsten Stellen der Adresse im dritten Byte und die wertniedrigsten im zweiten Byte. Bei dem 3-Byte-Befehl LXI, der im Operandenteil eine Register- und eine Speicheradresse enthält, muß vorher die Registerbezeichnung in den Befehlsteil verschoben werden. Dabei darf die symbolische Speicheradresse aus höchstens vier Zeichen bestehen.

```
C  ASSEMBLER INTEL 8080
      IMPLICIT INTEGER(A-Z)
      DEFINE FILE 10(1000,17,U,KONTR)
      DIMENSION Z(17),NALI(2,200),VORB(21),CODE(2,244)
      DATA BZ/0/
      DATA VORB/
     *'MOV   LDAX  POP   LXI   STAX  PUSH  MVI   INR',
     *'DCR   INX   DCX   ADD   ADC   DAD   SUB   SBB',
     *'ANA   ORA   XRA   CMP   RST   '/
C  1-BYTE-BEFEHLE
      DATA ((CODE(I,J),I=1,2),J=1,52)/
     *'NOP   00    STAXB 02    INX B 03    INR B 04',
     *'DCR B 05    RLC   07    DAD B 09    LDAXB 0A',
     *'DCX B 0B    INR C 0C    DCR C 0D    RRC   0F',
     *'STAXD 12    INX D 13    INR D 14    DCR D 15',
     *'RAL   17    DAD D 19    LDAXD 1A    DCX D 1B',
     *'INR E 1C    DCR E 1D    RAR   1F    INX H 23',
     *'INR H 24    DCR H 25    DAA   27    DAD H 29',
     *'DCX H 2B    INR L 2C    DCR L 2D    CMA   2F',
     *'INX S 33    INR M 34    DCR M 35    STC   37',
     *'DAD S 39    DCX S 3B    INR A 3C    DCR A 3D',
     *'CMC   3F    MOVB,B40    MOVB,C41    MOVB,D42',
     *'MOVB,E43    MOVB,H44    MOVB,L45    MOVB,M46',
     *'MOVB,A47    MOVC,B48    MOVC,C49    MOVC,D4A'/
      DATA ((CODE(I,J),I=1,2),J=53,104)/
     *'MOVC,E4B    MOVC,H4C    MOVC,L4D    MOVC,M4E',
     *'MOVC,A4F    MOVD,B50    MOVD,C51    MOVD,D52',
     *'MOVD,E53    MOVD,H54    MOVD,L55    MOVD,M56',
     *'MOVD,A57    MOVE,B58    MOVE,C59    MOVE,D5A',
     *'MOVE,E5B    MOVE,H5C    MOVE,L5D    MOVE,M5E',
     *'MOVE,A5F    MOVH,B60    MOVH,C61    MOVH,D62',
     *'MOVH,E63    MOVH,H64    MOVH,L65    MOVH,M66',
     *'MOVH,A67    MOVL,B68    MOVL,C69    MOVL,D6A',
     *'MOVL,E6B    MOVL,H6C    MOVL,L6D    MOVL,M6E',
     *'MOVL,A6F    MOVM,B70    MOVM,C71    MOVM,D72',
     *'MOVM,E73    MOVM,H74    MOVM,L75    MOVM,A77',
     *'MOVA,B78    MOVA,C79    MOVA,D7A    MOVA,E7B',
     *'MOVA,H7C    MOVA,L7D    MOVA,M7E    MOVA,A7F'/
      DATA ((CODE(I,J),I=1,2),J=105,156)/
     *'ADD B 80    ADD C 81    ADD D 82    ADD E 83',
     *'ADD H 84    ADD L 85    ADD M 86    ADD A 87',
     *'ADC B 88    ADC C 89    ADC D 8A    ADC E 8B',
     *'ADC H 8C    ADC L 8D    ADC M 8E    ADC A 8F',
     *'SUB B 90    SUB C 91    SUB D 92    SUB E 93',
     *'SUB H 94    SUB L 95    SUB M 96    SUB A 97',
     *'SBB B 98    SBB C 99    SBB D 9A    SBB E 9B',
     *'SBB H 9C    SBB L 9D    SBB M 9E    SBB A 9F',
     *'ANA B A0    ANA C A1    ANA D A2    ANA E A3',
     *'ANA H A4    ANA L A5    ANA M A6    ANA A A7',
     *'XRA B A8    XRA C A9    XRA D AA    XRA E AB',
     *'XRA H AC    XRA L AD    XRA M AE    XRA A AF',
     *'ORA B B0    ORA C B1    ORA D B2    ORA E B3'/
      DATA ((CODE(I,J),I=1,2),J= 157,200)/
     *'ORA H B4    ORA L B5    ORA M B6    ORA A B7',
     *'CMP B B8    CMP C B9    CMP D BA    CMP E BB',
     *'CMP H BC    CMP L BD    CMP M BE    CMP A BF',
     *'HLT   76    RNZ   C0    POP B C1    PUSHB C5',
     *'RST 0 C7    RZ    C8    RET   C9    RST 1 CF',
     *'RNC   D0    POP D D1    PUSHD D5    RST 2 D7',
     *'RC    D8    RST 3 DF    RPO   E0    POP H E1',
     *'XTHL  E3    PUSHH E5    RST 4 E7    RPE   E8',
     *'PCHL  E9    XCHG  EB    RST 5 EF    RP    F0',
     *'POP PSF1    DI    F3    PUSHPSF5    RST 6 F7',
     *'RM    F8    SPHL  F9    EI    FB    RST 7 FF'/
C  2-BYTE-BEFEHLE
      DATA ((CODE(I,J),I=1,2),J=201,218)/

     *'MVI B,06    MVI C,0E    MVI D,16    MVI E,1E',
     *'MVI H,26    MVI L,2E    MVI M,36    MVI A,3E',
     *'ADI   C6    ACI   CE    OUT   D3    SUI   D6',
     *'IN    DB    SBI   DE    ANI   E6    XRI   EE',
     *'ORI   F6    CPI   FE    '/
C  3-BYTE-BEFEHLE
      DATA  ((CODE(I,J),I=1,2),J=219,244)/
     *'LXI B,01    LXI D,11    LXI H,21    LXI S,31',
     *'SHLD  22    LHLD  2A    STA   32    LDA   3A',
     *'JNZ   C2    JMP   C3    CNZ   C4    JZ    CA',
     *'CZ    CC    CALL  CD    JNC   D2    CNC   D4',
     *'JC    DA    CC    DC    JPO   E2    CPO   E4',
     *'JPE   EA    CPE   EC    JP    F2    CP    F4',
     *'JM    FA    CM    FC    '/
      NA=1
      N=0
      WRITE(6,1000)
1000  FORMAT('1LISTEN-ASSEMBLER INTEL 8080 '/)
10    DO 20 I=1,5
      Z(I) = ' '
20    CONTINUE
      READ(5,2000) (Z(I),I=6,17)
2000  FORMAT(12A6)
      N=N+1
      NAME=Z(6)
      FUNK=Z(7)
      OPER=Z(8)
C EINTRAGEN DER NAMEN IN DIE NAMENSLISTE
      IF(NAME.EQ.' ') GOTO 25
      NALI(1,NA) = NAME
      NALI(2,NA) = DUAHEX(BZ)
      NA =NA + 1
C  ASSEMBLER-ANWEISUNGEN
25    IF(FUNK.EQ.' ') GOTO 120
      IF(FUNK.EQ.'END') GOTO 150
      IF(FUNK.NE.'ORG') GOTO 30
      BZ = UMWA(OPER,16)
      GOTO 120
C  DATEN-DEFINITIONEN
30    IF(FUNK.NE.'DB') GOTO 40
      Z(2) = BZ
      BZ = BZ + 1
      Z(3) = OPER
      GOTO 120
40    IF(FUNK.NE.'DW') GOTO 50
      Z(1) = 'U12'
      Z(2) = BZ
      BZ = BZ + 2
      Z(4) = OPER
      GOTO 120
50    IF(FUNK.NE.'DS') GOTO 60
      Z(2) = BZ
      BZ = BZ + UMWA(OPER,10)
      GOTO 120
60    IF(FUNK.NE.'EQU') GOTO 65
      NALI(2,NA-1) = OPER
      GOTO 120
C  OPERAND ENTHAELT CODE
65    DO 70 I=1,21
      IF(FUNK.NE.VORB(I)) GOTO 70
      IF(I.EQ.1) CALL SCHIE(FUNK,OPER,3)
      IF(I.NE.1) CALL SCHIE(FUNK,OPER,2)
      GOTO 80
70    CONTINUE
C  DURCHSUCHEN DER CODE-LISTE
80    DO 110 I=1,244
      IF(FUNK.NE.CODE(1,I)) GOTO 110
      IF(I.GT.200) GOTO 90
C  1-BYTE-BEFEHLE
      Z(2) = BZ
      BZ = BZ + 1
      Z(3) = CODE(2,I)
      GOTO 120
90    IF(I.GT.218) GOTO 100
C  2-BYTE-BEFEHLE
      Z(2) = BZ
      BZ = BZ + 2
      Z(3) = CODE(2,I)
      Z(4) = OPER
      GOTO 120
C  3-BYTE-BEFEHLE
100   Z(1) = 'U23'
      Z(2) = BZ
      BZ = BZ + 3
      Z(3) = CODE(2,I)
      Z(4) = OPER
      GOTO 120
110   CONTINUE
      Z(1) = 'CODE'
C  UEBERTRAGUNG AUF EXTERNEN SPEICHER
120   WRITE(10'N) Z
      GOTO 10
C  NAMEN EINSETZEN UND LISTE AUSGEBEN
150   WRITE(10'N) Z
      DO 180 I=1,N
      READ(10'I) Z
      IF(Z(1).NE.'U12'.AND.Z(1).NE.'U23') GOTO 170
      DO 160 J=1,NA
      IF(Z(4).NE.NALI(1,J)) GOTO 160
      IF(Z(1).EQ.'U12') CALL VERT(NALI(2,J),Z(3),Z(4))
      IF(Z(1).EQ.'U23') CALL VERT(NALI(2,J),Z(4),Z(5))
      Z(1) = ' '
      GOTO 170
160   CONTINUE
170   IF(Z(2).NE.' ')Z(2) = DUAHEX(Z(2))
      WRITE(6,5000)I,Z
5000  FORMAT(1X,I4,'.',2A6,2X,A2,1X,A2,1X,A2,2X,12A6)
180   CONTINUE
      STOP
      END
```

Bild 5. FORTRAN-Hauptprogramm

4 Unterprogramme

Die Unterprogramme (*Bild 6*), die die in FORTRAN schwierige Zeichenverarbeitung übernehmen, wurden ursprünglich in der Assembler-Sprache der Anlage UNIVAC 1108 geschrieben; hier erscheinen sie jedoch zum besseren Verständnis in FORTRAN.

Das Unterprogramm DUAHEX verwandelt eine Dualzahl in sedezimale Zeichen. Es übernimmt die Dualzahl in der Speicherstelle W und übergibt die Sedezimalzahl rechtsbündig als Zeichen.

Das Unterprogramm SCHIE verschiebt Teile des Operanden in den Befehlsteil. Der Inhalt der Speicherstelle N gibt dabei an, um wieviel Zeichen verschoben werden soll.

Das Unterprogramm VERT verteilt die sedezimalen vierstelligen Adressen der Speicherstelle AB auf die beiden Speicherstellen A und B zu je zwei Zeichen.

Das Unterprogramm UMWA verwandelt dezimale bzw. sedezimale Zeichen in eine Dualzahl. Dabei enthält die Speicherstelle W die umzuwandelnden Zeichen und die Speicherstelle B die Basis des Zahlensystems.

Die Unterprogramme benötigen für die Ausführung der in FORTRAN nicht verfügbaren Schiebeoperationen die beiden FUNCTION-Unterprogramme SHILI und SHIRE, die den Inhalt einer Speicherstelle um eine bestimmte Zahl von Zeichen nach links bzw. nach rechts verschieben. Sie arbeiten in den hier gezeigten FORTRAN-Versionen mit der Multiplikation bzw. Division von Zweierpotenzen. Sie werden jedoch besser in der Assembler-Sprache der betreffenden Rechenanlage geschrieben.

5 Übertragung auf eine andere Anlage

Die hier vorgestellte reine FORTRAN-Version läuft auf einer Anlage UNIVAC 1108 mit FORTRAN-V-Compiler. Diese Sprache dient hauptsächlich zur Verarbeitung von Zahlen. Die Zeichenbehandlung wie im vorliegenden Fall ist stark vom Code und vom Aufbau der verwendeten Rechenanlage abhängig. Die UNIVAC 1108 verarbeitet die Zeichen in einem 6-bit-Code mit sechs Zeichen in einem Wort. Die Nachbildung der Schiebebefehle in den Unterprogrammen SHILI und SHIRE bereitete Schwierigkeiten. Vor den sedezimalen Adressen der ORG-Anweisung muß der Buchstabe H und vor den Dezimalzahlen in der DS-Anweisung der Buchstabe D stehen.

Bei der Übertragung auf eine andere Anlage sollte diese bei der Zeicheneingabe die Angabe A6 im FORMAT zulassen. Bei A5 müssen die Eingabevorschriften vereinfacht werden. Wenn Schwierigkeiten auftauchen, sollte man versuchen, die Unterprogramme in Assembler-Sprache zu schreiben.

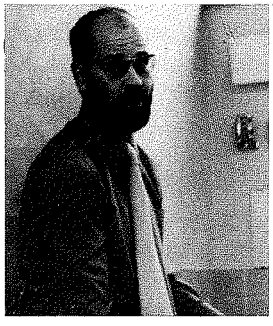
Literatur

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

[2] *Barron, D. W.*: Assembler und Lader, Carl Hanser Verlag München 1970.


```
      INTEGER FUNCTION DUAHEX(W)
      IMPLICIT INTEGER (A-Z)
      DIMENSION ZEI(16),ZIF(4)
      DATA ZEI/'ааааа0','ааааа1','ааааа2','ааааа3','ааааа4',
     *'ааааа5','ааааа6','ааааа7','ааааа8','ааааа9',
     *'аааааA','аааааB','аааааC','аааааD','аааааE',
     *'аааааF'/
      DATA LEER/'  ааа'/
      D = IABS(W)
      DO 10 I = 1,4
      IND = MOD(D,16)
      IF(W.GE.0) ZIF(5-I) = ZEI(IND+1)
      IF(W.LT.0) ZIF(5-I) = ZEI(16-IND)
      D=D/16
10    CONTINUE
      H=0
      DO 20 I=1,4
      H =SHILI(H,1)
      H = H+ ZIF(I)
20    CONTINUE
      DUAHEX = H + LEER
      RETURN
      END

      SUBROUTINE SCHIE(F,O,N)
      IMPLICIT INTEGER(A-Z)
      DATA L3/'ааа   '/,L2/'ааа
      FU = SHIRE(F,N)
      FU = SHILI(FU,N)
      OP = SHIRE(O,6-N)
      F = FU + OP
      O = SHILI(O,N)
      IF(N.EQ.3) O = O + L3
      IF(N.EQ.2) O = O + L2
      RETURN
      END

      SUBROUTINE VERT(AB,B,A)
      IMPLICIT INTEGER (A-Z)
      DATA L/'аа    '/
      HA = SHIRE(AB,2)
      A = SHILI(HA,4) + L
      B = SHILI(AB,4) + L
      RETURN
      END

      INTEGER FUNCTION UMWA(W,B)
      IMPLICIT INTEGER(A-Z)
      DIMENSION Z(6),ZEI(16)
      DATA ZEI/'ааааа0','ааааа1','ааааа2','ааааа3','ааааа4'
     *,'ааааа5','ааааа6','ааааа7','ааааа8','ааааа9'
     *,'аааааA','аааааB','аааааC','аааааD','аааааE'
     *,'аааааF'/
      X = W
      DIF = 0
      DO 20 I=1,6
      Z(I) = SHIRE(X,6-I) - DIF
      DIF = DIF + Z(I)
      DIF = SHILI(DIF,1)
20    CONTINUE
      FAKT = 1
      DUA = 0
      DO 40 I= 6,1,-1
      DO 30 J=1,B
      IF(Z(I).NE.ZEI(J)) GOTO 30
      DUA = DUA + (J-1)*FAKT
      FAKT = FAKT*B
30    CONTINUE
40    CONTINUE
      UMWA  = DUA
      RETURN
      END

      INTEGER FUNCTION SHILI(X,N)
      INTEGER X,BIT,FAKT
      DATA BIT/6/
      FAKT=2**(N*BIT)
      SHILI = X*FAKT
      RETURN
      END

      INTEGER FUNCTION SHIRE(X,N)
      INTEGER X,BIT,FAKT
      DATA BIT/6/
      FAKT = 2**(N*BIT)
      SHIRE = X/FAKT
      RETURN
      END
```

Bild 6. FORTRAN-Unterprogramme



Dipl.-Ing. Günter Schmitt ist waschechter Berliner. Er studierte an der TU Berlin und entwickelte von 1963 bis 1969 bei Siemens in Berlin Meßwandler. Dann zog es ihn nach Dieburg zur Fachhochschule der Deutschen Bundespost. Dort beglückt er seine Studenten mit Programmieren und Datenverarbeitung; seine Frau und seine beiden Töchter hat er leider für diese edle Kunst noch nicht begeistern können. Hobbys: Wandern und Häusle bauen
Privattelefon: (0 60 78) 42 10
ELEKTRONIK-Leser seit 1975

Die Lösung selbst sehr komplizierter Probleme schien mit dem Auftauchen der Mikroprozessoren in greifbare Nähe gerückt zu sein. Nach der ersten Begeisterung wurde aber deutlich, daß das Programmieren in Maschinensprache eine sehr zeitraubende, fehlerträchtige und über weite Strecken stupide Tätigkeit sein kann. Auf der Suche nach effektiveren Verfahren beschritt man den Weg, den die Datenverarbeitung vorgezeichnet hatte, und ist im Augenblick dabei, sich zunehmend höherer Programmiersprachen wie BASIC und FORTRAN zu bedienen. In zahlreichen Fällen ist dieser Weg jedoch nicht sinnvoll. Im folgenden Beitrag wird eine Alternative vorgestellt, die die wesentlichen Erkenntnisse der Datenverarbeitung verwertet, die aber dennoch auf die speziellen Belange des Mikroprozessoreinsatzes eingeht.

Dr. Karl Meinzer

IPS – eine neue Programmiertechnik für Mikrocomputer

Man hört oft den Hinweis, daß die üblichen Programmiersprachen (wie z. B. FORTRAN) das Softwareproblem lösen würden, weil man dann auf eine große „Bibliothek" von Programmen zurückgreifen kann. Bei näherem Hinsehen erweist sich dies Argument als wenig stichhaltig. Der Ingenieur, der einen digitalen PID-Regler realisieren möchte, kann mit einer FORTRAN-Bibliothek, die nur Programme zur Rentenberechnung oder für statistische Regressionen enthält, kaum etwas anfangen. Für Mikrocomputeranwendungen existiert praktisch keine brauchbare Software, da Minicomputer nur in den seltensten Fällen dort eingesetzt wurden, wo man heute Microcomputer vorsieht. Man ist daher durchaus frei, neue Wege zu beschreiten, zumal die konventionellen Lösungen auch beim Minicomputereinsatz keinesfalls alle Probleme optimal gelöst haben.

1 Forderung: eine unkonventionelle Programmiertechnik

1.1 Einsatzbereiche und Aufgaben von Programmiersprachen

Die allgemeine Verwirrung bezüglich der Programmiersprachen hat mehrere Ursachen. Zunächst ist es äußerst schwierig, das Problem überhaupt in rationalen Begriffen zu beschreiben, da die Schaffung von Programmiersprachen nur teilweise ein Ingenieurproblem ist. Eine Programmiersprache soll schließlich ein Vermittlungswerkzeug sein zwischen der „menschlichen" Betrachtungsweise eines Problems und der eindeutigen Form, in der ein Computer instruiert werden muß. Die Definition der „menschlichen" Betrachtungsweise ist eher ein psychologisches und künstlerisches Problem. Hinzu kommt, daß der Mensch sich schnell an eine gegebene Sprache gewöhnt und dann ihre Nachteile als „normal" empfindet und nicht mehr motiviert ist, etwas Neues zu lernen. In diesem Bereich ist daher nur mit sehr langsamen Fortschritten zu rechnen.

Zum besseren Verständnis des Problems ist es nützlich, den Einsatz von Computern in drei große Gruppen einzuteilen:

1. Mathematische Problemlösungen: Hier werden in der Regel nur kleine Datenmengen verarbeitet, aber die Rechen-Operationen können sehr kompliziert sein. Eine hohe Genauigkeit ist erforderlich.
2. Kommerzielle Datenverarbeitung: Hier werden große Datenmengen relativ einfachen Operationen unterworfen. Die Problematik steckt in der Menge der Daten und den damit verbundenen Dateiverfahren.
3. Ingenieurprobleme im weiteren Sinn: Hierzu gehören Maschinensteuerungen, Spiele, Probleme der künstlichen Intelligenz und andere Probleme, die Wechselwirkungen mit der Außenwelt erfordern. Programme dieser Kategorie verarbeiten meist nur geringe Datenmengen, und auch die Genauigkeitsforderungen sind mäßig. Der Aufwand steckt in komplizierten Abläufen.

Mikroprozessoren werden primär für diesen letzten Bereich eingesetzt. Die üblichen Programmiersprachen sind jedoch für die ersten beiden Aufgabenbereiche geschaffen worden. Insbesondere hat die Lösung mathematischer Probleme nachhaltig ihren Charakter geprägt.

1.2 Der Unterschied zwischen mathematischen und technischen Problemen

Für mathematische Aufgabenstellungen ist es charakteristisch, daß es Formeln zur Beschreibung des Problems gibt. Dies hat zur Folge, daß Programmier-

sprachen die Formeln fast in ihrer ursprünglichen Form als „Spezifikationen" der gewünschten Ergebnisse auffassen können. Die Entwicklung ging daher dahin, die Syntax solcher Formelausdrücke möglichst gut zu verstehen und programmtechnisch zu beherrschen.

Leider gibt es für den ingenieurmäßigen Einsatz von Computern keinen der Mathematik vergleichbaren Formelapparat. Vielmehr werden solche Probleme durch Abläufe und Entscheidungen beschrieben (z. B.: Wenn der Druck zu hoch ist, schließe das Ventil!). Zwingt man solche Problemstellungen in den syntaktischen Rahmen einer Programmiersprache, die für mathematische Probleme geschaffen worden ist, werden zwei Übersetzungen erforderlich:

1. Das Problem muß zunächst durch den Ingenieur in der Syntax der Sprache formuliert werden. Dies ist schon ein Übersetzungsvorgang, da der Ingenieur vor seinem geistigen Auge Abläufe sieht, die er in Spezifikationen (die Syntax der Sprache) umformen muß.
2. Anschließend muß ein geeignetes Programm die Syntax wieder in ausführbare Befehle umformen, da der Computer nur Ablaufanweisungen durchführen kann.

Unter diesen Umständen stellt sich die Frage, ob man nicht eine Sprache konstruieren kann, die auf eine syntaktische Ergebnisspezifikation verzichten kann und statt dessen rein prozedural aufgebaut ist (d. h. eine Aktion nach der anderen in strenger Reihenfolge ausführt). Dadurch ist nicht nur die Sprache technischen Problemen viel besser angepaßt, sondern auch die Implementation eines entsprechenden Systems wird erheblich einfacher.

Dies soll nicht so verstanden werden, daß man aus der bisherigen Entwicklung für Mikrocomputer nichts lernen kann. In der Tat sind einige äußerst leistungsfähige Prinzipien entwickelt worden, die sich am besten mit Schlagworten wie „strukturierte Programmierung", „Entwicklung von oben nach unten" (*top down design*), Modularisierung und gute Eigendokumentation beschreiben lassen. Diese Techniken sind besonders leistungsfähig in Programmen, deren Intelligenz in Ablaufstrukturen liegt, was ja gerade bei technischen Problemstellungen der Fall ist.

1.3 RPN und Erweiterbarkeit als Sprachbasis

Diese Überlegungen lassen sich ohne Schwierigkeiten als Basis eines Sprachentwurfs verwenden. Man kann rein prozedurale Programmiersprachen durch die Einführung von sogenannten Stapeln schaffen. Ein Stapel ist eine Wertablage, bei der nur die obersten Eintragungen zugänglich sind; die Werte, die zuletzt abgelegt wurden, müssen auch als erste wieder entfernt werden. Die meisten Mikrocomputer benutzen solche Stapel zur Zwischenspeicherung von Rückkehradressen bei Unterprogrammen und externen Programmunterbrechungen. Weniger bekannt ist, daß man auch alle Rechenoperationen auf solch einen Stapel beziehen kann. Die Taschenrechner der Firma Hewlett-Packard arbeiten z. B. nach diesem Prinzip. Diese Technik ist unter dem Namen umgekehrte polnische Notation (RPN) bekannt geworden. Mathematische Ausdrücke sind in dieser Form nicht ganz so leicht zu lesen wie in der üblichen algebraischen Form. Für den technischen Einsatz von Mikrocomputern spielt dies jedoch keine Rolle, da wie erwähnt die Intelligenz der Programme in den Abläufen steckt und Ausdrücke nur eine untergeordnete Bedeutung haben.

Hat man sich einmal für RPN entschieden, ergibt sich sofort eine Reihe von unerwarteten Vorteilen. Eine Reihe von Problemen verschwindet einfach, die in anderen Sprachen nicht oder nur unzulänglich zu lösen sind:

1. Die Parameterübergabe zwischen Programmmoduln wird durch den Stapel extrem einfach; formale Verfahren sind überflüssig, und Programme können vielmehr vertikal strukturiert werden.
2. Das Problem der Variableninitialisierung und ihr Gültigkeitsbereich entfällt. Lokale Variable befinden sich ohne Namen auf dem Stapel, nur globale Variable haben Namen.
3. Bei Variablen wird sauber zwischen „Topf" und seinem „Inhalt" unterschieden. Die Verfügbarkeit der Adressen erlaubt sehr leistungsfähige Adreßberechnungen (*Pointer*).
4. Das Testen der Programme wird sehr einfach. Moduln können „per Hand" über den Stapel mit Parametern versorgt werden. Indem man so die Resultate verfolgt, können Fehler leicht lokalisiert werden.
5. Da die Befehle genau in der Reihenfolge ausgeführt werden, in der sie hingeschrieben wurden, gibt es keine Zweifel, was der Compiler mit dem Programm anstellt.

Der vielleicht wichtigste Vorteil ergibt sich durch das Zusammenwirken mit später beschriebenen Eigenschaften: Da ein starrer syntaktischer Rahmen fehlt, wird die Sprache auf ganz natürliche Weise erweiterbar. Auf diese Weise kann das Grundsystem sehr kompakt gehalten werden; man kann alle Probleme durch geeignete Sprachergänzungen lösen. Gerade für kleine Systeme ist dieser Weg viel vernünftiger als die Sprache durch eine Anhäufung von „features" aufzublähen, die Speicherplatz belegen und nur selten benötigt werden. In der Tat läßt sich eine RPN-Sprache so konstruieren, daß man alle Anwendungsprogramme als problemspezifische Sprachergänzungen auffassen kann. Dadurch entfällt der Unterschied zwischen der eigentlichen Sprache, den Kommandos und der Anwenderprogrammierung.

Da sich mit einer höheren Programmiersprache nicht alle denkbaren Peripheriesituationen vorwegnehmen lassen, ist es zweckmäßig, die Sprache mit einem Assembler für den zugrunde liegenden Prozessor zu versehen. Dadurch lassen sich beliebige Hardwarekonfigurationen mit der höheren Sprache „verbinden". Auch gestattet ein integraler Assembler, extrem zeitkritische Programmabschnitte schnellstmöglich laufen zu lassen. Da in einem Programm in der Regel nur sehr wenige Assemblermoduln benötigt

werden und diese meist sehr kurz und einfach sind, wird hierdurch die Portabilität der Programme kaum beeinträchtigt.

1.4 Dialog- und Inkrementaltechnik und Benutzerfreundlichkeit

Neben der eigentlichen Sprache ist auch die Bequemlichkeit beim Umgang mit dem System von entscheidender Bedeutung. Ein Compiler, der fünf Durchläufe braucht, um ein ausführbares Programm zu erzeugen, kann kaum als bequem gelten. Eine Grundforderung ist, daß das System interaktiv ist, d. h. daß sich Probleme im Dialog lösen lassen. Das bedeutet, daß das Übersetzungsprogramm mit einem Durchlauf auskommt und inkremental arbeitet, so daß stückweise „Reparaturen" möglich sind. Zusätzlich soll ein interpretativer Modus vorhanden sein, um die Programmentwicklung zu erleichtern.

Wahrscheinlich beruht die Beliebtheit von BASIC und APL – trotz ihrer Unzulänglichkeit für technische Problemstellungen – auf dem bequemen interaktiven Betrieb.

Der Dialogbetrieb hat üblicherweise zur Folge, daß mit Namen versehene Objekte definiert werden müssen, bevor man sich auf sie beziehen kann. In Hinblick auf die anderen Vorteile ist dies ein kleiner Preis.

Ein schnelles System, das mit einem Durchlauf auskommt, beseitigt auch die Notwendigkeit von verschiebbarem Code.

Statt eines getrennten Ladevorgangs übersetzt man einfach das Programm neu an die Stelle, an der man den Code haben möchte.

Wenn man diese Überlegungen zur Basis einer Programmiersprache macht, stellt sich heraus, daß nur noch wenige Freiheitsgrade bestehen. Die Hauptschwierigkeit besteht darin, einen geeigneten Satz von Grundbefehlen zu schaffen. Natürlich kann dies nicht willkürlich erfolgen; vielmehr ist eine enge Rückkopplung von Anwendungsproblemen erforderlich, die die Sprache verwenden. Nur so kann man sicherstellen, daß der Befehlssatz leistungsfähig ist und die Befehle eine gute Eigendokumentation in RPN-Strukturen gewährleisten. Das wichtigste Prinzip dabei muß sein, daß nur so wenig Regeln wie möglich verwendet werden und diese ohne Ausnahme befolgt werden.

2 Die verschiedenen Implementierungsverfahren

Zur Schaffung von Softwaresystemen stehen grundsätzlich drei Techniken zur Verfügung. Die Sprachen BASIC und APL werden in der Regel als sogenannte Interpreter realisiert. Zur Ausführungszeit des Programms wird der Programmtext analysiert, und die erkannten Befehle werden zur Ausführung gebracht. Wegen dieser Analyse sind Interpreter meist für Echtzeitaufgaben zu langsam.

Auf der anderen Seite sind interpretierbare Programme sehr änderungsfreundlich, da keine Übersetzung erfolgt und der Quelltext immer vorliegt.

Eine andere Technik besteht darin, das Programm vor der Ausführung zu analysieren und in Maschinenbefehle zu übersetzen. Solche Übersetzungsprogramme werden Compiler genannt. Die solchermaßen erzeugten Programme belegen generell sehr viel Speicherplatz und enthalten häufige Wiederholungen von typischen Befehlsfolgen.

Bei einer dritten Technik versucht man die Vorteile beider Verfahren zu kombinieren und die Nachteile zu vermeiden. Das Prinzip dabei ist, daß man den Grundsatz von Befehlen einer höheren Sprache durch entsprechende Maschinenbefehlsfolgen nur einmal definiert. Ein Compiler erzeugt nun Pseudocode, der diese Grundroutinen als Befehlssatz für einen auf diese Weise definierten virtuellen höheren Computer auffaßt. Die Interpretation dieser Pseudebefehle (in der Regel Adressen, die zu ausführbarem Code zeigen) benötigt nur sehr wenig Zeit. Ein solches System ist nur etwa 2- bis 3mal langsamer als optimaler Maschinencode. Unter der Voraussetzung, daß dies akzeptabel ist, liefert diese Technik extrem kompakte Programme, die wesentlich weniger Speicherplatz als bei den anderen Verfahren belegen.

Außerdem ist der Pseudocode unabhängig vom verwendeten Mikroprocessor; nur die Grundbefehle stellen die Verbindung zum Prozessor her. Dadurch kann das System sehr schnell auf einen neuen Prozessor angepaßt werden, und Anwendungsprogramme können ohne Änderung sofort laufen. Da die ganze Technik noch stark im Fluß ist, vermeidet man auf diese Weise eine zu starke Abhängigkeit von bestimmten Produkten oder Herstellern. Der Übergang auf einen neuen Prozessor bedeutet, daß etwa 1 KByte Maschinencode neu geschrieben werden muß.

3 IPS – die Realisierung des Konzepts

Der Verfasser hat ein System unter Verwendung der skizzierten Prinzipien unter dem Namen IPS geschaffen, um innerhalb der Universität Marburg eine schnelle Programmerstellung bei Forschungsprojekten zu ermöglichen, die Mikrocomputer für Datenerfassungs- und Steuerungsaufgaben verwenden. Da solche Programme meist nur an einer Stelle eingesetzt werden, ist eine ökonomische Programmierung extrem wichtig. Ein zweites Einsatzgebiet ergab sich aus der Mitarbeit des Verfassers bei einem Raumfahrtprojekt. Sowohl in einem Satelliten als auch bei den Bodenstationen werden Mikrocomputer eingesetzt. Wegen der verschiedenen Prozessoren, die hierbei eingesetzt werden, war eine höhere Programmiersprache erforderlich, um einen Programmaustausch zwischen den Anlagen zu ermöglichen.

Die Entwicklung von IPS wurde durch die Tatsache erleichtert, daß in den USA ein System mit einer ähnlichen Zielsetzung für Minicomputer unter dem Namen Forth beschrieben worden ist [1]. Einige Ideen zur Implementation konnten für IPS direkt übernommen werden. Die Interaktionstechniken und die Gestalt der Sprache, wie sie sich dem Benutzer darstellt, mußten jedoch neu gestaltet werden, um die im Microcomputerbereich andersartige Hardware optimal einsetzen zu können.

4 Hardware: Entwicklungssystem und Anwenderanlage

Der Ausgangspunkt der Entwicklung von IPS war, daß ein Teil der Programme in minicomputerähnlichen Anlagen laufen soll. Diese Anlagen haben neben einem Massenspeicher (Kassetten oder Floppy Disk) eine schreibmaschinenähnliche ASCII-Tastatur und einen Fernsehschirm zur Kommunikation mit dem Benutzer.

Darüber hinaus soll für Prozessoren ohne diese Einrichtungen die Programmerstellung auf den erstgenannten Anlagen möglich sein. Dadurch wird z. B. eine einfache Programmerstellung für spezielle Steuerungen möglich, deren Programme in ROM abgelegt sind und die später anderen Geräten ihren Charakter geben, ohne selbst als Mikrocomputer in Erscheinung zu treten.

Die Entwicklung von IPS-Programmen setzt einen Computer voraus, der folgende Einheiten hat:

- einen 8-bit-Mikroprozessor: Es existieren Versionen für die Typen COSMAC (RCA), 8080 (und damit für den Z80), 6502 und 6800;
- 16-KByte-Speicher (RAM). IPS selbst belegt etwas unter 6 KByte, so daß genügend Platz für Anwendungsprogramme vorhanden ist;
- eine ASCII-Tastatur und einen Bildspeicher mit 16 Zeilen zu 64 Zeichen (dieser Speicher nimmt 1 KByte ein und liegt wie andere Speicher im Adreßraum des Prozessors; sein Inhalt wird gleichzeitig durch eine entsprechende Elektronik als Text auf einem Bildschirm dargestellt; der Prozessor hat entweder beim Zugriff Priorität, oder das Auslesen des Bildes erfolgt durch eine Verkämmung mit dem Prozessortakt).

  Diese Art der Anzeige gestattet einen wesentlich transparenteren Verkehr mit dem Computer als der übliche Fernschreibmodus;
- eine 20-ms-Taktquelle: Dieser Takt dient zur Führung von Uhr und Stoppuhren und zur Koordination verschiedener Aufgaben;
- kein oder nur so viel ROM, um das System vom Massenspeicher laden zu können;
- einen Massenspeicher zur Daten- und Programmspeicherung: In der Regel sind Kassettensysteme ausreichend. Floppy-Disk-Speicher können auf Grund der Struktur von IPS ohne Extraaufwand als virtueller Hauptspeicher betrieben werden.

  Es werden ausschließlich Blöcke konstanter Länge (512 Byte, ein halber Fernsehschirm) abgespeichert. Diese Daten können sowohl ASCII-Zeichen als auch Binärdaten sein.

5 Einführung in die Sprache

Im folgenden soll IPS nur so weit dargestellt werden, daß ein typisches Programm diskutiert werden kann und auf diese Weise ein orientierender Eindruck vom Charakter der IPS-Programmiertechnik entsteht. Zunächst einige Worte zur Kommunikation mit dem System: Der Fernsehschirm ist wie erwähnt in 16 Zeilen eingeteilt. Die unteren acht Zeilen werden für Eingaben (manuell oder vom Massenspeicher) benutzt. Die acht oberen Zeilen dienen dem System für Antworten an den Benutzer.

5.1 Die Grundbefehle

IPS verwendet als Grunddatentyp 16-bit-Ganzzahlen mit Vorzeichen. Wenn man 125 schreibt, wird diese Zahl auf den Stapel gelegt und in der ersten Zeile auf dem Fernseher als 125 angezeigt. Schreibt man nun –20, erscheint 125 –20 auf dem Fernseher. Gibt man jetzt * ein (Multiplikation), wird die Anzeige –2500. Auf dem Stapel kann sich eine beliebige Zahl von Zahlen befinden; die Rechenoperationen beziehen sich immer auf die obersten Werte (nach rechts dargestellt).

Neben den normalen Rechenoperationen stehen logische Operatoren zur Verfügung, um Bitmanipulationen zu erlauben. Dazu werden einfach die entsprechenden Worte hingeschrieben (UND, ODER, EXO und NICHT). Alle Aktionen haben den Charakter von Befehlen, die Syntax ist extrem einfach: Alle Namen werden in den Eingaben durch mindestens ein Leerzeichen getrennt. Namen sind beliebige Zeichenfolgen (die ersten 63 signifikant), z. B. * oder /MOD.

Eine weitere Klasse von Befehlen gestattet, die Reihenfolge von Zahlen auf dem Stapel zu verändern; z. B. DUP dupliziert den obersten Wert und legt ihn noch einmal auf den Stapel oder VERT vertauscht die beiden oberen Werte.

Zahlen können unter einem Namen dauerhaft gespeichert werden. Man kann sowohl Konstante (KON), Variable (VAR) oder Felder (FELD) definieren. Der Aufruf eines Konstantennamens liefert den Wert der Konstanten auf den Stapel. Bei Variablen oder Feldern wird statt dessen die Adresse auf den Stapel gelegt; um ihren Wert zu erhalten steht das Wort @ zur Verfügung. Es ersetzt die Adresse auf dem Stapel durch den Inhalt der Adresse. Eine Zahl kann in eine Adresse mit dem Befehl ! gespeichert werden. Dieser Befehl erwartet eine Zahl und darüber eine Adresse auf dem Stapel, die beide entfernt werden.

Die Befehle@B und !B haben die gleiche Funktion, beziehen sich jedoch auf Bytes statt auf 16-bit-Zahlen.

5.2 Programmerzeugung und Strukturierung

Programmoduln werden durch einen Doppelpunkt, auf den ein Name folgt, erzeugt. Es werden dann die Tätigkeiten hingeschrieben, als ob der Computer die Befehle direkt ausführen würde. Ein Modul wird mit einem Strichpunkt abgeschlossen. Schreibt man nun den Namen des Moduls, werden alle Tätigkeiten, die im Modul stehen, ausgeführt. Natürlich kann der Name des Moduls auch in weiteren Moduln eingebaut werden; man kommt so zu beliebigen Hierarchien von Programmen.

Für Entscheidungsprozesse stehen einige Befehle zur Verfügung, die dem Prinzip der strukturierten Programmierung angepaßt sind. Die Bedeutung von JA?, NEIN:, DANN dürfte ziemlich klar sein; diese Namen entsprechen der IF THEN ELSE Konstruktion

anderer Sprachen. Um mit RPN konsistent zu sein, geht der Test dem JA? voraus.

Es stehen Konstruktionen für wiederholte Programmdurchläufe zur Verfügung. JE ... NUN erwartet zwei Parameter: den Laufbeginn und das Laufende. Die Befehle zwischen dem JE ... NUN werden für alle aufeinanderfolgenden Werte vom Laufbeginn bis zum Laufende einschließlich ausgeführt. Wenn das Laufende kleiner als der Laufbeginn sein sollte, werden die Aktionen zwischen dem IE ... NUN überhaupt nicht ausgeführt. Diese Laufanweisung kann auch mit anderen Inkrementen als mit eins verwendet werden.

Zwei weitere Laufanweisungen stehen für Fälle zur Verfügung, bei denen die Anzahl der Durchläufe nicht von vornherein anzugeben ist. ANFANG ... ENDE? entspricht etwa dem DO UNTIL von PASCAL; der Test ist am Ende der Laufanweisung. Die Befehle ANFANG ... JA? ... DANN/NOCHMAL entsprechen dem DO WHILE mit dem Test am Laufanfang.

Die Strukturierungsworte von IPS haben die Befehle GO TO und die sogenannten Label anderer Sprachen überflüssig gemacht. Man gewöhnt sich sehr schnell an diese Strukturen, denn sie gestatten, die meisten Probleme viel natürlicher und übersichtlicher zu formulieren als mit Sprungbefehlen. In der Tat werden die Programme so übersichtlich, daß man auch nur in den seltensten Fällen Flußdiagramme benötigt.

5.3 Spracherweiterungsmechanismen

Die Spracherweiterung nach „oben" ergibt sich automatisch aus der Art, wie Programmoduln geschaffen werden; in der Tat kann man jeden Modul als anwendungsorientierte Spracherweiterung auffassen, da bei der Benutzung dieser Moduln kein Unterschied zu den Grundroutinen der Sprache besteht.

Die Erweiterungsmöglichkeit nach „unten" wird durch den Assembler sichergestellt. Mit dem Assembler lassen sich Moduln schaffen, die sich in ihrer Anwendung durch nichts von IPS-Moduln unterscheiden. Man kann auf diese Weise beliebige Hardware versorgen oder zeitkritische Probleme programmieren.

Der Assembler verwendet im Gegensatz zur Sprache IPS die englische Sprache, um weitgehend die Originalkürzel für die Maschinenbefehle verwenden zu können. Allerdings bestehen doch einige Unterschiede zur Original-Assemblersprache. Einmal ist der Assembler formatfrei wie alle Eingaben bei IPS. Weiterhin wird auch beim Assembler die strukturierte Programmierung verwendet. Statt der Sprungbefehle werden die Worte Y? N: TH und BEGIN END verwendet. Wie beim IPS-Compiler wird der Stapel zur Verwaltung der Sprungadressen verwendet. Die Reihenfolge der Assemblerworte entspricht der IPS-Konvention: erst Quelle, dann Ziel, dann eventuelle Zusatzangaben und schließlich die Operation.

5.4 Mehrprogrammbetrieb

Technische Problemstellungen erfordern gewöhnlich mehrere Programme, die nahezu unabhängig voneinander quasi-parallel laufen müssen. Der Autor hat dieses Problem durch eine „Kette" gelöst. Diese stellt ein Feld von mehreren Programmen dar, die zyklisch ausgeführt werden. Beim Systemstart ist diese Kette nur durch den Compiler besetzt, die anderen Positionen enthalten NOPs (*no operation*). Man kann diese Kettenposition durch Programme besetzen, die dann auch periodisch ausgeführt werden. Diese Vorgänge lassen sich auch von Programmen ausführen, und Programme können sich selbst aushängen. Es stellte sich heraus, daß auf diese Weise ein sehr allgemeines „Multiprogramming" möglich ist, ohne daß man für jede Kettenposition einen eigenen Stapel benötigt.

Neben den Einhänge- und Aushänge-Operatoren sind noch zwei weitere Funktionen erforderlich. Diese Befehle ‚Warten' und ‚Fortsetzen') gestatten, auf innere oder äußere Ereignisse zu warten, bevor weitere Aktivitäten ausgelöst werden. Die vier Kettenoperatoren haben überraschenderweise ein zentrales Betriebssystem überflüssig gemacht, ohne nennenswerte Einschränkungen zu bedingen.

Die Kette ist nicht in allen Fällen schnell genug. Deshalb ist zusätzlich ein Unterbrechungsbetrieb vorgesehen. Neben der bekannten Maschinenunterbrechungstechnik (Interrupt), die mit dem Assembler programmiert werden muß und sehr kurze Reaktionszeiten ermöglicht, erlaubt IPS außerdem noch sogenannte Pseudounterbrechungen. Ein Stapelcomputer besitzt ideale Voraussetzungen für einen Unterbrechungsbetrieb, da die Programme wiedereintrittsfähig und rekursiv sind und bei Unterbrechungen keine „Rettvorgänge" erforderlich sind. Man kann daher den Emulator so gestalten, daß er Unterbrechungen zwischen den einzelnen Code-Routinen akzeptiert. Diese Pseudo-Unterbrechungen haben eine größere Latenz als Maschinenunterbrechungen, benötigen aber keinen Verwaltungsaufwand.

Der 20-ms-Eingang löst eine solche Pseudounterbrechung aus, um die Uhr und die vier Stoppuhren zu führen und andere Verwaltungsaufgaben wahrzunehmen.

In der Praxis steht somit für jede Dringlichkeitsstufe ein optimales Verfahren zur Verfügung; die folgende Faustregel hat sich bei den 8-bit-Prozessoren gut bewährt:

Bedienung erforderlich in (bzw. höchste Folgefrequenz):

< 200 µs (> 5 kHz): DMA bzw. spezielle Hardware

200 µs...20 ms
(50 Hz...5 kHz): Prozessorunterbrechung (in Assembler)

20 ms...200 ms
(5...50 Hz): Pseudounterbrechung (in IPS)

> 200 ms (< 5 Hz): Kettenbetrieb (in IPS)

6 Ein Programmbeispiel

Eine IBM-Selectric-Ausgabeschreibmaschine soll gesteuert werden: Das Programm liest unformatierte ASCII-Kassetten, fügt die entsprechenden Steuerzei-

chen ein und erzeugt die Ansteuersignale der Schreibmaschine im Selectric-Spezialcode. Dieser Beitrag wurde z. B. mit dem IPS-Texteditor auf einem Bildschirm geschrieben und dann mit diesem Programm ausgedruckt.

Ein Bandblock enthält 512 Zeichen (8 Zeilen). Das Programm verwendet einen zyklischen Puffer für 2 Blöcke; S ist eine Variable, die die Anzahl der zum Drucken vom Band gelesenen Blöcke enthält. Wenn S gleich null ist, sind keine Blöcke zum Druck vorhanden; S gleich zwei bedeutet einen vollen Puffer, und das Bandgerät muß angehalten werden.

6.1 *Das Hauptprogramm*

Am Ende des Programms (*Bild 1*) befindet sich die Routine SCHREIB, das Hauptprogramm (wegen der erwähnten Notwendigkeit, Objekte zunächst zu definieren, bevor man sich auf sie beziehen kann, liest man IPS-Programme in der Regel von hinten nach vorne).

```
( IBM SCHREIBMASCHINENSTEUERUNG VOM 2 . 11. 77 )
0 KON CBA
HIER 2 + ? CBA ! HIER 68 + $H !
CBA    VAR CHARP
0      VAR LE
#1000 VAR SBP
#1000 VAR IBP
0      VAR BL
CODE REMOVE 14 LD IM #C PHI 0 LD IM #C PLO.8 LD IM #A PLO
       BEGIN  #D I/O LD MX #20 AND IM ( BELEGT? )
       D=0 Y? #C DEC #C GHI            ( 140 US LOOP )
      VERT D=0 END TH #FF LD IM #A STR 3 I/O #A DEC 4 I/O NEXT

CODE H/TRANSMIT 1 LD IM #D PLO BEGIN 8 LD IM #A PLO PS INC PS LD
A #FF XOR IM  #A STR 100 LD IM #C PHI 0 LD IM #C PLO #A INC
        BEGIN #D I/O LD MX #30 AND IM  ( FREI? )
        D=0 NOT Y? #C DEC  #C GHI
       VERT D=0 END NEXT
                TH #A DEC #D GLO D=0 Y? 3 I/O
                                     N: 4 I/O
                                     TH NEXT
CODE L/TRANSMIT 0 LD IM #D PLO 0 END
CODE MSTATUS PS ->X PS DEC #D I/O #A ->X
             PS DEC 0 LD IM PS STR NEXT
0      VAR LV
0      VAR S
0      VAR EINFL
4    FELD SC         #1020 #400C SC 2 !FK
18   FELD 3ZEICHEN
128 FELD T1

: INCR DUP @ 1 + VERT ! ;
: ADJ DUP @ #3FF UND #1000 ODER VERT ! ;
: LIES LADEFLAGGE @B NICHT DUP LV @ UND
       JA? IBP @ 512 + IBP ! IBP ADJ    S INCR 0 LV !
       DANN
       EINFL @ S @ 2 < UND UND ( LADEFLAGGE NICHT )
       JA? IBP @ DUP #200 + $LOAD 1 LV !
       DANN ;

: NORMALZEICHEN DUP MSTATUS EXO #80 UND =0
                JA? DUP #80 UND >0 ( DREHE KOPF ) JA?    1
                                                  NEIN:  2
                                                  DANN
                    H/TRANSMIT 0 4 JE NUN REMOVE 0 6 JE NUN
                DANN L/TRANSMIT CHARP INCR REMOVE ;

: SONDERBEH #20 - DUP 4 < JA? DUP 2 = JA? 1 LE !
                                      DANN
                              SC + @B H/TRANSMIT CHARP INCR
                              REMOVE
                          NEIN: CHARP @ 2 - CHARP !
                                4 - DUP DUP + + 3ZEICHEN +
                                CHARP @ 3 >>>
                          DANN ;
: TYPLINE #0D ( CR/LF )  CHARP @ 1 + !B CBA CHARP ! 0 LE !
          ANFANG CHARP @ @B #7F UND T1 + @B
                 DUP DUP #1F > VERT #30 < UND
                 JA?   SONDERBEH
                 NEIN: NORMALZEICHEN
                 DANN LE @
            ENDE? ;
: SZEILE SBP @ CBA 64 >>> CBA 1 - CHARP !
         CBA DUP 63 + JE I @B #20 - >0
                         JA? I CHARP !
                         DANN
                       NUN TYPLINE ;

: SBLOCK 0 7 JE SZEILE   SBP @ 64 + SBP ! LIES
                MSTATUS 2 UND >0 ( STOP? )
                JA? 0 EINFL !
                DANN
             NUN SBP ADJ S @ 1 - S ! BL INCR ;

: SCHREIB EINFL @ JA? S @ >0
                       JA? SBLOCK BL @ 4 >
                           JA? 0 BL ! MSTATUS    ( START? )
                                    JA? 0 7 JE CBA 1 - CHARP !
                                            TYPLINE
                                       NUN
                                    NEIN: 0 EINFL !
                                    DANN
                               DANN
                          DANN
                     NEIN: 0 BL ! MSTATUS #80 UND =0
                                JA?  ( TIEFSTELLUNG BEI AUS )
                                     2 H/TRANSMIT REMOVE DANN
                           MSTATUS ( START? ) JA? 1 EINFL !
                                              DANN
                     DANN
       MSTATUS 2 UND >0
       JA? 0 EINFL !
       DANN LIES ;

( ENDE SCHREIBMASCHINE; SCHREIB IST EINGEHAENGT )

( CODETABELLE FUER IBM KOPF CORRESPONDENCE COURIER )
#2020   #2020   #2020   #2020   #2321   #2020   #2220   #2020
#2020   #2020   #2020   #2020   #2020   #2020   #2020   #2020
#B420   #8787   #CF26   #58D7   #DE96   #4625   #0119   #C634
#7FCC   #3E37   #574F   #5E16   #071F   #FFC9   #BE27   #9928
#9C24   #9A82   #D2DB   #F9B8   #95C3   #93F0   #FCCA   #CCB2
#90D1   #C5DD   #BBF3   #84BD   #C0FA   #54F6   #7531   #8129
#1CD8   #1A02   #525B   #7938   #1543   #1370   #7C4A   #4C32
#1051   #455D   #3B73   #043D   #407A   #D476   #F5B1   #2049

   T1      64 !FK

#084F   #582B   #2B08   #0853   #2D2F   #2708   #082D   #6060
#2708
   3ZEICHEN 9 !FK     ( ENDE TABELLE )
( CODETABELLE FUER IBM KOPF 963 )
#2020   #2020   #2020   #2020   #2321   #2020   #2220   #2020
#2020   #2020   #2020   #2020   #2020   #2020   #2020   #2020
#8420   #0787   #DE04   #9670   #F6C6   #F0CF   #7502   #7A01
#7F76   #3E37   #5E57   #1F16   #464F   #BED7   #FFB7   #FA9F
#F973   #B8B1   #D8D1   #9990   #C0C9   #B4FC   #D4BD   #35DD
#CC9C   #B2C5   #D2BB   #93DB   #CA9A   #24C3   #2526   #F527
#7981   #3831   #5851   #1910   #4049   #347C   #543D   #155D
#4C1C   #3245   #523B   #135B   #4A1A   #2943   #2882   #20F3

   T1      64 !FK

#0828   #295F   #5F08   #083D   #272F   #2D08   #0829   #282D
#2D08
   3ZEICHEN 9 !FK     ( ENDE TABELLE )
```

Bild 1. Programmbeispiel mit IPS

Da SCHREIB in der Kette eingehängt ist, wird diese Routine periodisch ausgeführt. SCHREIB prüft zunächst, ob EINFLAG gesetzt ist; nur dann wird die Druckaktivität gestartet. Wenn S größer als null ist, ist ein Block im Puffer vorhanden, und es wird der Modul SBLOCK (Schreibe Block) ausgeführt.

Ein Blockzähler BL zählt die Anzahl der gedruckten Blöcke; wenn er vier überschreitet, wird der Druckbetrieb angehalten (Seite voll). Wenn jedoch der Startschalter auf „ein" steht, werden acht Zeilenvorschübe ausgeführt, und es wird weitergedruckt (Endlospapier).

Die anderen Teile des Hauptprogramms sind weniger wichtig, sie stellen vor allem sicher, daß am Ende des Druckens der Druckkopf auf „klein" steht, da sich sonst die Maschine nicht mehr manuell auf „klein" schalten läßt.

6.2 Die Hilfsroutinen (Unterprogramme)

Das Hauptprogramm ruft SBLOCK auf. SBLOCK schreibt acht Zeilen (0 7 JE ... NUN), indem es den Modul SZEILE ausführt; anschließend wird der Zeiger des Puffers (SBP) entsprechend einer Zeile um 64 erhöht. Daneben wird noch geprüft, ob ein Stoppbefehl vorliegt.

Der Modul SZEILE nimmt 64 Zeichen aus dem zyklischen Puffer bei SBP und legt sie in den Hilfspuffer CBA (>>> ist ein Feldtransportbefehl). Dann wird CBA Zeichen für Zeichen so abgesucht, daß der Zeiger CHARP auf das letzte zu druckende Zeichen der Zeile zeigt. Der Laufindex wird durch I bereitgestellt. Mit dem so vorbereiteten Puffer CBA wird TYPLINE gerufen. Dieser Modul fügt Zeilenvorschub/Wagenrücklauf nach dem letzten zu druckenden Zeichen ein. Dann werden die Zeichen in CBA ohne das höchste Bit (#7F UND) als Index zur Tabelle T 1 zum Aufsuchen des Selectric-Codes verwendet. Wenn der gefundene Wert zwischen #1F und #30 liegt, ist eine Sonderbehandlung durch den Modul SONDERBEH erforderlich. Andernfalls liegt ein normales Zeichen vor, und NORMALZEICHEN wird ausgeführt.

Zunächst zu SONDERBEH: Einige Tätigkeiten der Maschine werden nicht mit dem Druckkopf ausgeführt, sondern über andere Steuermagnete (z. B. Leerzeichen). Diese Steuersignale müssen an einen anderen Ausgang gegeben werden als die Kopfsteuersignale.

Einige ASCII-Zeichen sind auf dem verwendeten Kugelkopf nicht vorhanden. Sie werden durch zwei andere Zeichen, die übereinander geschrieben werden, simuliert. Dazu wird das Hilfsfeld entsprechend manipuliert. SONDERBEH führt diese Tätigkeiten aus.

NORMALZEICHEN ist einfacher. Es wird geprüft, ob der Kopf in der richtigen Grundstellung für das zu druckende Zeichen ist. Wenn nicht, wird zunächst eine Kopfdrehung ausgeführt.

Jede Maschinentätigkeit beginnt damit, daß der entsprechende Magnetcode den Routinen L/TRANSMIT oder H/TRANSMIT übergeben wird. Diese Moduln warten, bis die Maschine bereit ist, den Code zu empfangen und geben dann den Code an den entsprechenden Ausgang. Der Modul REMOVE macht das Gegenteil: Wenn die Maschine den Erhalt des Codes quittiert hat, wird die Magnetansteuerung entfernt. Auf diese Weise wird die Ausgabe mit der Maschine synchronisiert.

Die letzten drei Moduln sind besonders bemerkenswert, da sie im Assemblercode des Prozessors geschrieben sind. Diese Routinen zeigen, wie sich spezielle Hardware ohne Einschränkungen im Rahmen von IPS bedienen läßt.

Der Modul LIES im Schreibprogramm schließlich ist für das Einlesen der Kassettenblöcke verantwortlich und soll hier nicht weiter diskutiert werden.

6.3 Testverfahren und Entwicklungsaufwand

Das Programm wurde zunächst in einer Anlage mit Bildschirm und Tastatur getestet; dann wurde es für den Steuercomputer der Schreibmaschine (ohne Bildschirm und Tastatur) „transcompiliert". Dort wird es nur noch durch einen Schalter bedient. Es belegt in dieser Form unter 4 KByte Speicher einschließlich der Puffer und Tabellen. Die Entwicklung dauerte 1 1/2 Wochen; die meiste Zeit davon wurde zur Lösung von Hardwareproblemen benötigt.

7 Einige Details der IPS-Implementierung

Zum Schluß sollen einige spezielle Punkte der IPS-Implementierung dargestellt werden, die auch bei der Entwicklung anderer Systeme nützlich sein könnten. Wie erwähnt werden auf dem Stapel 16-bit-Zahlen verwaltet. Wenn man einen zweiten Stapel vorsieht, der überwiegend vom System zur Ablage von Rückkehradressen verwendet wird, gibt es keine Konfliktsituation bei der Parameterübergabe von Moduln; die Benutzung der Stapel wird wesentlich transparenter. Der Rückkehrstapel ist auch dem Programmierer zugänglich, es sind lediglich einige Einschränkungen zu beachten.

Ein sehr leistungsfähiges Konzept bei der Emulation eines „High-Level"-Computers ist die indirekte Pseudobefehlsausführung. Zunächst scheint es naheliegend, als Pseudocode Adressen zu ausführbaren Code-Routinen zu verwenden (*Bild 2*). Man kann jedoch auch die Emulation eine Stufe indirekter machen. d. h. am Anfang jeder Routine wird ein Zeiger vorgesehen, der zum ausführbaren Code der Routine zeigt (*Bild 3*). Dadurch erhalten alle Routinen das gleiche Format, unabhängig von ihrer Art. Insbesondere benötigen Modulaufrufe (die etwa den Unterprogrammen entsprechen) nicht zusätzlich eine Adresse.

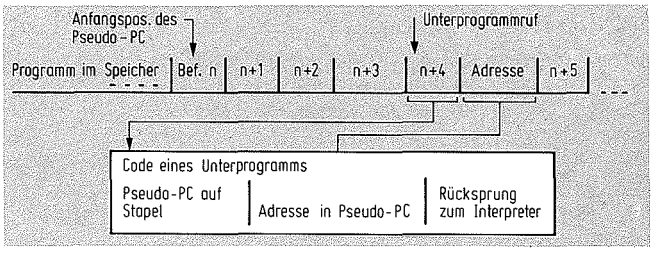


Bild 2. Direkte Ausführung eines Unterprogramms

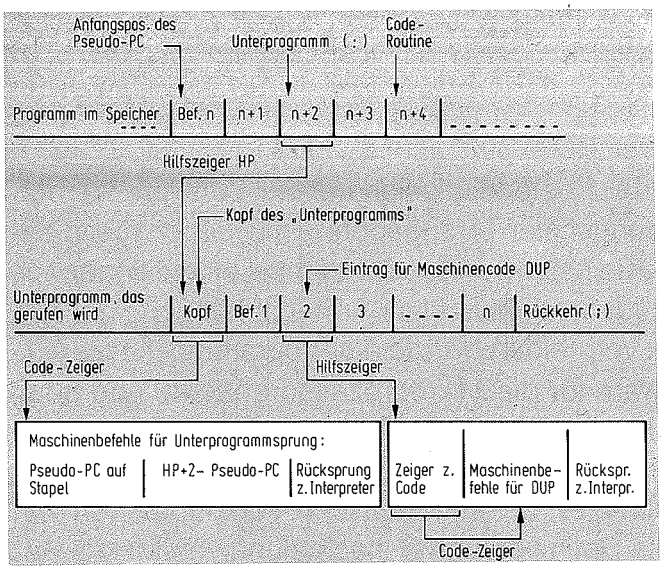


Bild 3. Indirekte Ausführung eines Unterprogramms

Weil der Zeiger zum ausführbaren Code sich ja am Anfang des Moduls befindet, ist die Adresse implizit vorhanden. Dadurch wird nicht nur Speicher gespart, sondern der Aufbau des Compilers wird auch einfacher, da semantische Inhalte der Befehle keine Bedeutung für die Compilation haben. Auch folgt aus diesem Sachverhalt die natürliche Erweiterbarkeit der Sprache – die Unterscheidung zwischen Anwenderroutinen und solchen, die die Sprache darstellen, entfällt.

In IPS werden die Namen (entsprechend codiert) und die Zeiger zu dem entsprechenden Pseudocode in einer Streuspeichertabelle für 512 Namen aufbewahrt. Die Erfahrung zeigt, daß bei dieser Größe in einem 16-K-System Tabelle und Speicher etwa gleichzeitig voll werden. Der Autor zog die Streutabelle einer verketteten Liste vor, da der Zugriff wesentlich schneller ist und die Transcompilation für Anlagen wie z. B. die Ausgabeschreibmaschine keine besondere Maßnahmen erfordert (die Namen und Zeiger sind vom eigentlichen Programm getrennt).

Da RPN-Sprachen weitgehend syntaxfrei sein können und der Compiler durch die indirekte Emulation praktisch blind, d. h. ohne Kenntnis dessen, was compiliert wird, arbeiten kann, ist das ganze System relativ einfach. Allerdings sind bei der Compilation doch vier verschiedene Arten von Namen zu berücksichtigen.

Tabelle. Die vier verschiedenen Eintragungstypen

Eintragungstyp	Compiler-Modus: Direkt (interpretierend)	Compilierend (:)
Normal (:)	ausführen	compilieren
INT	ausführen	Fehlermeldung
PRIOR	Fehlermeldung	ausführen
HPRI	ausführen	ausführen

Die Fehlermeldung kennzeichnet das betreffende Eingabewort durch ein vorgesetztes Fragezeichen und unterbricht die weitere Verarbeitung der Eingabe

Diese 2-bit-Information wird mit dem Namencode in der Streutabelle aufbewahrt. Die meisten Namen sind von der Art „normal" und werden außerhalb des Definitionsmodus ausgeführt und innerhalb compiliert. Die *Tabelle* beschreibt die Namensarten. Typ INT wird z. B. für Variableneintragungen oder den : verwendet, um Eintragungen innerhalb von Eintragungen zu verhindern. Typ PRIOR wird für JA? NEIN: DANN und ; verwendet. Diese Moduln werden zur Compilationszeit ausgeführt, z. B. um Sprungadressen zu verwalten bzw. zu prüfen, ob kein Strukturierungsfehler begangen worden ist.

Es ist beachtenswert, daß die gute Lesbarkeit strukturierter Programme hauptsächlich von der eingerückten Schreibweise zusammengehöriger oder bedingter Aktionen herrührt. Der Computer analysiert jedoch nicht diese Geometrie. Er nutzt vielmehr die Tatsache aus, daß bei einer konsequenten Anwendung der Strukturierungsregeln diese Geometrie zu einer LIFO-Struktur (Stapel) isomorph ist. Dies beleuchtet den engen Zusammenhang zwischen Stapelcomputer und strukturierter Programmierung.

8 Entwicklungsstand und Ausblick

Zur Zeit existieren IPS-Versionen für die Typen COSMAC (CDP 1801/2) 8080, 6502 und 6800. Daneben existiert eine Sonderversion für den COSMAC, bei der alle Interaktionen über Funklinien abgewickelt werden können; diese Version ist für die AMSAT Phase III Satelliten vorgesehen. Außer der letzten Version, die kleiner ist, belegen die Systeme 6 KByte Speicher.

Die ersten IPS-Versionen liefen Mitte 1976; seit dieser Zeit wurde das System bei einer großen Zahl von Anwendungen innerhalb der Universität Marburg und bei dem erwähnten Raumfahrtprojekt vervollkommnet. Speziell beim technischen Einsatz von Mikroprozessoren konnte durch IPS der Zeitaufwand bei der Programmerstellung und der Inbetriebnahme gegenüber konventionellen Programmierhilfen erheblich gesenkt werden.

Der Autor schrieb einen Artikel über IPS auch für die amerikanische Zeitschrift BYTE.

Literatur

[1] *Moore, C.-H.:* Forth, a new way to program a mini-computer. Astron. Astrophys. Suppl. (1974) 15, S. 497...511.

Dr. rer. nat. Karl Meinzer stammt aus Iserlohn (NRW) und studierte in Münster und Marburg theoretische Physik. Promotion in angewandter Physik. Seit 1971 plant er im „Zentralen Entwicklungslabor für Elektronik" der Universität Marburg den Bau und Einsatz von Elektronik bei Forschungsprojekten.
Hobbys: Reisen, Radeln (mit Familie), Fliegen, Amateurfunk, Satellitenbau und Jazz, nicht unbedingt in dieser Reihenfolge.
Privattelefon: (0 64 21) 2 26 05
ELEKTRONIK-Leser seit 1970

Dr.-Ing. Klaus-Dieter Brinkmann

Frank Kreißel

Zur Eindämmung der Kostenexplosion bei der Software werden vielfältige Entwicklungshilfen [4] angeboten, die jedoch meist beachtliche, für den Kleinstanwender untragbare Investitionen erfordern und die zudem oftmals stark firmenabhängig sind. Um diese Nachteile zu vermeiden, wurde ein Programmpaket zur Erstellung von Software für die Mikroprozessoren 8080 und 8085 entwickelt, das ausschließlich in FORTRAN IV geschrieben und auf 16-bit-Maschinen einsetzbar ist. Über den ersten Teil – einen komfortablen Assembler, der arithmetische Verknüpfungen von Operanden und die Definition und Schachtelung von Makro-Befehlen erlaubt – wird im folgenden kurz berichtet.

Ein universell einsetzbarer Assembler

Die wichtigste Forderung bei der Entwicklung dieses Programms bestand in der Kompatibilität mit den Assemblern der Firmen Intel [1], NEC [2] und Siemens [3]. Es wurden deshalb die von diesen Programmsystemen bekannten Pseudo-Assembler-Befehle in der Art übernommen, daß mit ihrer Hilfe in jedem Falle ein gültiger Maschinencode erzeugt wird. Diese Befehle und der Befehlsvorrat der Prozessoren 8080 und 8085 werden im folgenden als bekannt vorausgesetzt und nicht näher beschrieben.

Die zweite wichtige Forderung bestand darin, daß das Entwicklungssystem auf Kleinrechnern lauffähig und leicht auf andere Anlagen transferierbar sein soll. Es wurde deshalb – unter bewußter Inkaufnahme hoher Rechenzeiten – vollständig in FORTRAN IV für 16-bit-Maschinen geschrieben, wodurch leichte Transferierbarkeit garantiert ist.

1 Neue Pseudo-Assembler-Befehle

Die Wirkungsweise neu eingeführter Befehle wird kurz erläutert und an den *Bildern 1* und *2* demonstriert.

1.1 ONBYTE/OFFBYTE

Hiermit kann ein Teilbytezähler zur Bestimmung des erforderlichen Speicherplatzes für einzelne Programmteile ein- bzw. ausgeschaltet werden.

ONBYTE setzt den Zähler auf Null und veranlaßt die Zählung des benötigten Speicherplatzes (in Byte) bis zu der durch OFFBYTE gekennzeichneten Programmstelle.

1.2 ONZYKLUS/OFFZYKLUS

Dieser Befehl gestattet eine Abschätzung des Zeitverhaltens einzelner Programmteile. ONZYKLUS setzt einen Zyklenzähler auf Null, der bis zum Befehl OFFZYKLUS entsprechend der Bearbeitungsdauer der Assembler-Befehle inkrementiert wird (selbstverständlich kann hierdurch nur ein grober Überblick über die Dauer einzelner Programmteile geliefert werden, da zum Zeitpunkt der Assemblierung keine Aussagen über bedingte Sprünge und die Zahl der Schleifendurchläufe möglich sind).

1.3 RAM <ADRESSE>

Bei der Assemblierung behandelt das Programm ohne diesen Befehl den gesamten durch den 8080/85 adressierbaren Speicherbereich als ROM-Speicher. Ein Schreiben in diesem Bereich durch z. B. STA ADRS führt deshalb zu einer Speicherwarnung.

Durch RAM <ADRESSE> wird der Speicherbereich von ADRESSE bis $FFFF_{16}$ (≙ 64 KByte) als RAM-Speicher erklärt, in den geschrieben werden kann.

1.4 ORGWRITE <ADRESSE>, n_1/VAR1,...,n_N/VARN

Die Wirkung dieses neuartigen Befehls, der hier nur in seiner einfachsten Form vorgestellt wird, entspricht einer Kombination der bekannten Befehle ORG, SET und EQU. Mit seiner Hilfe wird den Variablen VAR1...VARN in der angegebenen Reihenfolge beginnend bei ADRESSE ein Speicherbereich von jeweils n_1...n_N Byte zugewiesen. Zusätzlich wird automatisch für alle Variablen, die durch „Schreibbefehle in den Speicher transferiert" werden (STA VARX, SHLD VARY) hinter dem für VARN reservierten Platz der erforderliche Speicherbereich zur Verfügung gestellt. Während im *Bild 2* die bekannten Assembler [1, 2, 3] zur Adreßvereinbarung für KARTE und ANZAHL zwei SET/EQU-Befehle erfordern, kann mit dem hier

beschriebenen Programm das Problem durch eine ORGWRITE-Anweisung gelöst werden. Zusätzlich wird hiermit der 80-Byte-Bereich für KARTE vor unbeabsichtigtem Überschreiben geschützt.

Der kombinierte Einsatz der Befehle RAM und ORGWRITE erleichtert somit dem Benutzer die Aufteilung von Programm und Daten auf ROM- und RAM-Speicher, indem eine vorgegebene Einteilung überprüft und automatisch verbessert wird.

```
CROSS-ASSEMBLER MUEC 8080 AUF MUTTERCOMPUTER PDP11    DATUM: 22. 1.78

VERSION: MUEC 01/78/07::RT-11 V2C

SEITE: 1   MUEC 8080

ADR.    H-CODE     MARKE      BEFEHL   OPERAND  KOMMENTAR                          T-ZY.    T-BY.


                                ORG      8200H    ;ANFANGSADRESSE FESTLEGEN
                   MARKE1:      SET      3        ;WERT FUER MARKE1 ZUWEISEN
                   PORTA:       EQU      0        ;WERT FUER PORTA ZUWEISEN
8200    3E 82      AMPEL:       MVI      A,82H    ;BEFEHLSWORT FUER E/A-BAUSTEIN LADEN
8207    D3 03                   OUT      MARKE1
8204    3E 21      PHAS1:       MVI      A,21H    ;BYTE FUER PHASE 1 LADEN
8206    D3 00                   OUT      PORTA    ; ACC AUSGEBEN
8208    3E 0E                   MVI      A,0EH    ;ZEITFAKTOR LADEN
820A    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
                   MARKE1:      SET      0        ;MARKE1 ERHAELT NEUEN WERT
820D    3E 11      PHAS2:       MVI      A,11H    ; BYTE FUER PHASE2 LADEN
820F    D3 00                   OUT      MARKE1   ;ACC AUSGEBEN
8211    3E 0A                   MVI      A,0AH    ;ZEITFAKTOR LADEN
8213    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
8216    3E 0A      PHAS3:       MVI      A,0AH    ;BYTE FUER PHASE 3 LADEN
8218    D3 00                   OUT      PORTA    ;ACC AUSGEBEN
821A    3E 0A                   MVI      A,0AH    ;ZEITFAKTOR LADEN
821C    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
821F    3E 0C      PHAS4:       MVI      A,0CH    ;BYTE FUER PHASE 4 LADEN
8221    D3 00                   OUT      PORTA    ;ACC AUSGEBEN
8223    3E 14                   MVI      A,14H    ;ZEITFAKTOR LADEN
8225    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
8228    3E 0A      PHAS5:       MVI      A,0AH    ;BYTE FUER PHASE5 LADEN
822A    D3 00                   OUT      PORTA    ;ACC AUSGEBEN
822C    3E 0A                   MVI      A,0AH    ;ZEITFAKTOR LADEN
822E    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
8231    3E 19      PHAS6:       MVI      A,19H    ;BYTE FUER PHASE 6 LADEN
8233    D3 00                   OUT      PORTA    ;ACC AUSGEBEN
8235    3E 0A                   MVI      A,0AH    ;ZEITFAKTOR LADEN
8237    CD 00 83                CALL     DELAY    ;UNTERPROGRAMM AUFRUFEN
823A    C3 00 82   REPT:        JMP      AMPEL    ;SPRUNG ZUM ANFANG
                                ORG      8300H    ;NEUE ADRESSE GLEICH 8300H
                   DELAY:       TOTZ     0DAH,32H ;MACRO-AUFRUF MIT ZWEI PARAMETERN
                   TOTZ:        MACRO    PAR1,PAR2
                                ONBYTE
                                ONZYKLUS
8300    06 32      LSW:         MVI      B,PAR2   ;KONSTANTE IN REGISTER B LADEN     00007    00002
8302    0E D6      LOOPB:       MVI      C,PAR1   ;KONSTANTE IN REGISTER C LADEN     00014    00004
8304    0D         LOOPC:       DCR      C        ;REGISTER C ERNIEDRIGEN            00019    00005
8305    C2 04 83                JNZ      LOOPC    ;WENN C#0 --> LOOPC                00029    00008
8308    05                      DCR      B        ;REGISTER B ERNIEDRIGEN            00034    00009
```

Bild 1. Programmauswertung mit Byte- und Zyklenzähler

```
CROSS-ASSEMBLER MUEC 8080 AUF MUTTERCOMPUTER PDP11    DATUM: 22. 1.78

VERSION: MUEC 01/78/07::RT-11 V2C

SEITE: 1   MUEC 8080

ADR.    H-CODE     MARKE      BEFEHL   OPERAND          KOMMENTAR


                                                        ;DIESES PROGRAMM ILLUSTRIERT DIE
                                                        ;WIRKUNG DES ORGWRITE

                                ORG      8000H          ;ANFANGSADRESSE GLEICH 8000H
                                RAM      8240H          ;ANFANG DES RAM-SPEICHERS
                                ORGWRITE 8240H,80/KARTE ;ES WERDEN IM RAM-SPEICHER
                                                        ;80 BYTE RESERVIERT UNTER
                                                        ;DEM NAMEN KARTE
8000    FB         START:       EI                      ;INTERRUPTFLIP-FLOP SETZEN
8001    DB 11                   IN       17             ;KANAL 17D EMPFANGEN
8003    32 90 82                STA      ANZAHL         ;GELESENES BYTE SPEICHERN IN
                                                        ;ADRESSE, DIE DURCH ORGWRITE DEFINIERT WIRD
8006    4F                      MOV      C,A            ;AKKUKULATOR --> REGISTER C
8007    21 40 82                LXI      HL,KARTE       ;ADRESSE VON KARTE IN HL LADEN
800A    06 00                   MVI      B,0            ;REGISTER B=0
800C    09                      DAD      BC             ;HL=HL+BC
800D    2B         NEXSIG:      DCX      HL             ;HL=HL-1
800E    DB 12                   IN       18             ;AKKUKULATOR VON KANAL 18 LADEN
8010    77                      MOV      M,A            ;AKKUKULATOR IN DURCH HL
                                                        ;ADRESSIERTES BYTE LADEN
8011    79                      MOV      A,C            ;C ---> A
8012    3D                      DCR      A              ;A=A-1
8013    4F                      MOV      C,A            ;A ---> C
8014    C2 0D 80                JNZ      NEXSIG         ;WENN C#0 SPRINGE ZU NEXSIG
8017    76                      HLT                     ;HALTZUSTAND EINNEHMEN
8018    00                      NOP                     ;LEERBEFEHL
                                END                     ;ENDE DES PROGRAMMS


LISTE DER MARKEN:
-----------------

NAME    WERT     ART      SCHREIBEN ZUGRIFF DEFINITION SPEICHER-WARNUNG


A       0007 SET             REG.     JA       JA
ANZAHL  8290 ORGWRITE-DEF    JA       JA       INT.
B       0000 SET             REG.     JA       JA
BC      0000 SET             REG.     JA       JA
C       0001 SET             REG.     JA       JA
D       0002 SET             REG.     NEIN     JA
DE      0002 SET             REG.     NEIN     JA
E       0003 SET             REG.     NEIN     JA
H       0004 SET             REG.     NEIN     JA
HL      0004 SET             REG.     JA       JA
KARTE   8240 ORGWRITE-DEF    NEIN     JA       INT.
L       0005 SET             REG.     NEIN     JA
M       0006 SET             REG.     JA       JA
NEXSIG  800D ADRESSE         NEIN     JA       JA
PSW     0006 SET             REG.     NEIN     JA
SP      0006 SET             REG.     NEIN     JA
START   8000 ADRESSE         NEIN     NEIN     JA

KEINE FEHLER FESTGESTELLT

STATISTIK:
----------

SPEICHERBEDARF :  106 BYTE

DURCH ORGW DEF. BEREICH :   81 BYTE
```

Bild 2. Speicherzuweisung durch ORGWRITE-Befehl

1.5 $ <N>

Zur Erleichterung der Aufteilung seiner Programme im Speicher stehen dem Anwender neun verschiedene Adreßzähler zur Verfügung. Durch $ <N> wird der Zähler mit der Nummer N zum aktuellen Adreßzähler erklärt. Die Adressen für die folgenden Programmteile werden entsprechend dem Stand dieses Zählers berechnet.

2 Aufbau, Wirkungsweise und Umfang des Programms

In zwei Durchgängen wird versucht, aus dem vom Benutzer im freien Format geschriebenen Programm und einer eventuell angewählten permanenten Makrodatei den Maschinencode zu erstellen.

Im ersten Durchgang wird eine Syntaxprüfung vorgenommen, eine Markentabelle erstellt, und es wird untersucht, ob alle angesprochenen Makros vorhanden sind.

Zu Beginn des zweiten Durchganges wird intern den Registern und Registerpaaren ein Wert gemäß *Tabelle 1* zugewiesen. Mit Hilfe der Markentabelle werden die absoluten Adressen berechnet und die Assembler-Befehle in Maschinencode übersetzt. Dieses wird durch eine Wertzuweisung für die einzelnen Operanden und die Befehle vorgenommen (Einzelheiten siehe z. B. [2], C 7...C 9). Der Vorteil dieses Verfahrens besteht darin, daß eine große Vielfalt in der Adressierung von Registern, Registerpaaren, Stackpointer, Speicher und Prozessorstatuswort gegeben ist. So entsprechen beispielsweise folgende Befehle einander:

MOV B + 2, D * E + 1

MOV 2, 7

MOV D, A

Das gesamte Programm umfaßt etwa 1500 FORTRAN-Statements und besteht aus einem Hauptprogramm und vierzehn Unterprogrammen. Ein Großteil dieser Unterprogramme dient der Textverarbeitung und kann auf anderen Anlagen eventuell durch Systemroutinen ersetzt werden, während ein Unterprogramm der Zuordnung der Dateien und der Abfrage der Uhrzeit dient und somit etwas maschinenabhängig ist.

3 Arbeiten mit dem Programm

Das Programm in der beschriebenen Version ist auf einer Rechenanlage PDP 11/10

unter dem Betriebssystem RT11-Foreground/Background implementiert. Zur Demonstration der Maschinenunabhängigkeit ist außerdem eine Version auf eine UNIVAC 1108 transferiert und dort eingesetzt worden.

Der Steuerung des Programmablaufes dienen die folgenden mit dem Zeichen „." beginnenden Befehle, die in beliebiger Reihenfolge eingegeben werden können. Ist ein Befehl, der an den ersten beiden Buchstaben erkannt wird, gültig, so antwortet die PDP 11 durch Schreiben des Zeichens „*" auf die Konsole. Hieran anschließend hat der Benutzer seine Daten zur Ablaufsteuerung einzugeben. In *Bild 3* ist für einen möglichen Ablauf ein vollständiges Beispiel gegeben.

3.1 .END

Dieser Befehl, der keine weiteren Daten benötigt, beendet die Eingabe von Steuerbefehlen und startet die Assemblierung.

3.2 .FORMAT

Mit Hilfe dieses Befehls kann das Druckformat vom Anwender beliebig verändert werden. Die benötigten Daten werden in der Form eingegeben:

<FELD> = <SPALTE>

Beim FORMAT-Befehl sendet der Rechner kein „*"-Zeichen. SPALTE bezeichnet hierbei die Position im 132 Spalten breiten Ausgabeprotokoll, an der eine durch FELD bestimmte Information gedruckt werden soll. *Tabelle 2* gibt einen Überblick über die verschiedenen Möglichkeiten der Textausgabe.

Durch Angabe einer Spaltennummer Null wird die Ausgabe von FELD unterdrückt. Ohne den FORMAT-Befehl wird ein Protokoll gemäß Bild 2 erstellt.

3.3 .IN

Dieser Befehl steuert die Eingabe des zu assemblierenden Codes. Der Benutzer muß hinter dem Zeichen „*" angeben, wo sein Programm steht.

3.4 .MAC

Entsprechend dem vorherigen Befehl hat der Benutzer anzugeben, wo seine Makros gespeichert sind,

```
Befehl                  Bedeutung

.R MU8080               Start des Programmes MU8080
.IN                     Eingabesteuerung
*TEST.DAT               Der zu assemblierende Text steht in der
                        Datei TEST.DAT
.MAC                    Anwahl einer Macro-Datei
*MACRO.DAT              Die Macros sind in der Datei MACRO.DAT
                        gespeichert
.OUT                    Ausgabesteuerung
*LP:                    Druckausgabe über den Line-Printer
.VERSION                Auswahl eines Processortyps
*INTEL85                Intel 8085 wird benutzt
.OBJ                    Steuerung für den Maschinencode
*ASS.DAT                Speicherung des Maschinencodes in ASS.DAT
.FORMAT                 Neugestaltung des Druckformates
OA = 001                Adressen in oktaler Form ab Spalte 1
OC = 007                Maschinencode in oktaler Form ab Spalte 7
MA = 013                Marken ab Spalte 13
BE = 018                Befehle ab Spalte 18
OP = 025                Operanden ab Spalte 25
KO = 040                Kommentar ab Spalte 40
TB = 090                Teilbytezähler ab Spalte 90
TZ = 098                Teilzyklenzähler ab Spalte 98
.END                    Eingabe Ende; Assemblierung starten.
```

Bild 3. Anwendung der Steuerbefehle auf der Anlage PDP 11

d. h. der Rechner erwartet den Namen der Makro-Datei.

3.5 .OBJ

Erscheint dieser Befehl, so soll der erstellte Maschinencode in eine Datei abgespeichert werden. Der Rechner benötigt den Namen dieser Datei.

3.6 .OUT

Es ist der Name des Gerätes anzugeben, auf dem der Druck des Ausgabeprotokolls erfolgen soll.

3.7 .VERSION

Die Mikroprozessoren 8080 und 8085 unterscheiden sich hinsichtlich des Befehlsvorrates und der Ausführungszeiten einzelner Befehle voneinander. Für die Prozessoren des Typs 8080 geben die verschiedenen Hersteller unterschiedliche Ausführungszeiten für einzelne Befehle an [1, 2, 3, 5]. Durch .VERSION wird festgelegt, für welchen Prozessor as-

Tabelle 1. Interne Wertzuweisung für Register, Registerpaare, Speicher, Stackpointer und Prozessorstatuswort

Operand	Wert
A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
BC	0
DE	2
HL	4
SP	6
PSW	6

Tabelle 2. Daten für die Formatsteuerung

Daten	Bedeutung
BE	Befehlsfeld
HA	Adresse sedezimal
HC	Maschinencode sedezimal
KA	Kartenzähler
KO	Kommentar
MA	Markenfeld
NO	Fehlermeldungen
OA	Adresse oktal
OC	Maschinencode oktal
OP	Operandenfeld
OR	Original Datenkarte
TB	Teilbytezähler
TZ	Teilzyklenzähler
ZY	Zyklenzähler

Tabelle 3. Rechenzeiten für die Assemblierung auf PDP 11 und UNIVAC 1108

Programm	Rechenzeiten [s] (absolut; Zeit/Karte)	
	PDP 11	UNIVAC 1108
BLINK [6]	22;1.57	10;0.71
AMPEL [6]	36;1.02	14;0.60
Eigenes Programm aus 329 Befehlen	225;0.68	126;0.38

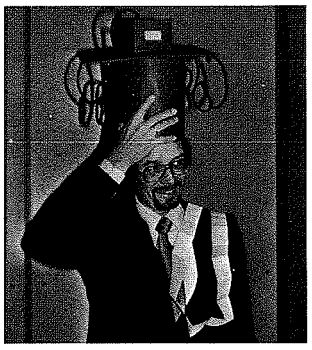
sembliert werden soll. Fehlt dieser Steuerbefehl, wird der Typ 8080 von NEC zugrunde gelegt. Weitere gültige Eingabedaten neben NEC sind SIEMENS (SIEMENS ≙ 8080 von Siemens oder von Intel, da beide gleiches Zeitverhalten aufweisen und den gleichen Befehlsvorrat besitzen) und INTEL85.

4 Speicherbedarf und Rechenzeiten

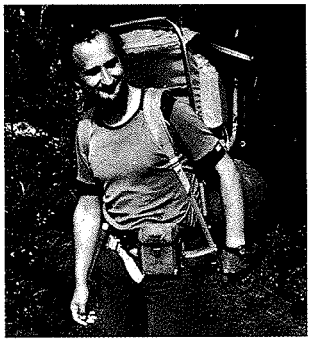
Unabhängig von der maximalen Anzahl von Befehlen, Marken und Makros benötigt das Programm mindestens 20 KWorte zu je 16 bit. Für jede zu speichernde Marke werden zusätzlich sechs Worte benötigt, so daß bei maximal 200 Marken 21 KWorte erforderlich sind.

Die Rechenzeiten, die selbstverständlich von der benutzten Maschine und dem Betriebssystem abhängen, betragen auf der PDP 11 für Programme mittlerer Größe einige Minuten, wie aus der *Tabelle 3* ersichtlich ist.

Die Arbeit stammt aus dem Institut für Theoretische Elektrotechnik und Meßtechnik der Universität Karlsruhe.



Dr.-Ing. Klaus-Dieter Brinkmann stammt aus Quedlinburg, ist verheiratet und Vater eines $1^1/_2$ Jahre alten Sohnes. Nach dem Studium der Elektrotechnik an der TH Darmstadt wechselte er 1973 als Wissenschaftlicher Assistent an das Institut für Theoretische Elektrotechnik und Meßtechnik der Universität Karlsruhe, wo er im Dezember 1977 über Schaltungslayout promovierte.
Zu seinen Hobbys zählen: Skilaufen, Windsurfen und Tennis.
Diensttelefon: (07 21) 6 08 26 29
ELEKTRONIK-Leser seit 1972



Frank Kreißel, zwar in Northeim geboren, aber vom Gemüt echter Pfälzer, studiert nach längerem Bundeswehr-Intermezzo Elektrotechnik an der TU Karlsruhe, wo er im Moment seine Diplomarbeit anfertigt. Aus langer Nebentätigkeit als Programmierer entwickelte sich die Neigung zur Software, die sich auch im Thema der Studienarbeit (s. o.) niederschlug.
Hobbys: (selbstgespielte) Musik, (Berg-)Wandern und Reisen in allen Formen und natürlich Elektronik.
Telefon: (06 34 22) 65 86
ELEKTRONIK-Leser seit 1975

Literatur

[1] 8080 Microcomputer Systems User's Manual. Firma Intel.
[2] The μCOM-8 Software Manual. Firma NEC.
[3] Microprocessorbausteine System SAB 8080. Firma Siemens.
[4] *Gößler, R.*: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43.
[5] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (IV). ELEKTRONIK 1977, H. 8, S. 67...74.
[6] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (V). ELEKTRONIK 1977, H. 9, S. 71...78.

Professionelles Computersystem mit niedrigem Preis

Zusammen mit passenden Peripheriegeräten stellt Commodore unter der Bezeichnung CBM 3001 ein vollwertiges Computersystem mit Zentraleinheit, Floppy-Disk und Drucker für professionelle Anwendung vor. Dieses System ist bereits für weniger als 10 000 DM erhältlich. Dadurch wird ein Einsatz direkt am Arbeitsplatz in Forschung und Entwicklung, in Schulen und Betrieben, bei Medizinern, Juristen und Architekten möglich. Die Programmierung erfolgt in BASIC, als Zentraleinheit findet der bewährte Mikroprozessor 6502 Verwendung, das Betriebssystem umfaßt 13 KByte, der Anwenderspeicher ist von 16 auf 32 KByte ausbaubar. Der 9-Zoll-Monitor kann 1000 Zeichen auf 25 Zeilen zu 40 Zeichen anzeigen. Zusätzlich zum ASCII-64-Zeichensatz sind 64 grafische Symbole abbildbar. Der Drucker ist mit eigener Mikroprozessorsteuerung ausgestattet, es können alle Zeichen des CBM 3001 auf Normalpapier ausgegeben werden. Die Floppy-Disk verfügt ebenfalls über einen eigenen Mikroprozessor der mit Hilfe des Betriebssystems (DOS) Lesen und Schreiben von Daten bei gleichzeitiger Datenübertragung zum Zentralsystem durchführt. Verwendet werden Minidisketten, die Formatierung erfolgt im Laufwerk.

□ Vertrieb: Commodore Büromaschinen GmbH, Frankfurter Str. 171–175, 6078 Neu-Isenburg, Tel. (0 61 02) 80 03.

„Personal Computer" PC 100 für vielseitige Anwendungen

Unter der Bezeichnung PC 100 KIT liefert Siemens die komplett bestückte und geprüfte Baugruppe eines Mikrocomputersystems mit vollständiger ASCII-Tastatur, alphanumerischer LED-Anzeige mit 20 Zeichen, Thermodrucker mit 20 Zeichen (120 Zeichen/min) sowie einem 1-KByte-RAM als Arbeitsspeicher. Weitere Sockel für zusätzliche Speicher sind vorgesehen. Gehäuse und Netzteil sind nicht im PC 100 enthalten. Das 8-K-Monitorprogramm bietet umfangreiche Möglichkeiten.
Der Systembus ist auf eine Anschlußleiste geführt. Für periphere Zusatzgeräte wie Fernschreiber und Kassettenrecorder sowie für die E/A-Kanäle sind weitere Anschlüsse vorhanden. Ein Benutzerhandbuch in englischer Sprache wird mitgeliefert. Als Komplettgerät ist der PC 100 ebenfalls lieferbar, dann ist er in ein Gehäuse mit Netzteil eingebaut und sofort betriebsbereit. Gegenüber dem Kit sind außerdem zusätzlich noch ein 8-KByte-BASIC-Interpreter und ein erweiterter Arbeitsspeicher mit 4-KByte-RAM enthalten. Bei dieser Ausführung wird ein Handbuch in deutscher Sprache mitgeliefert.

□ Hersteller: Siemens AG, ZVW 104, Postfach 103, 8000 München 1.

Dipl.-Inform.
Willi Haas

Im folgenden Artikel wird ein Programm zur Steuerung eines Fischer-Technik-Modells vorgestellt, mit dessen Hilfe man Werkstücke berührungslos messen und sortieren kann. Dabei wird weniger Wert auf die Darstellung des Meßverfahrens gelegt als auf die Erläuterung des Zusammenspiels zwischen Rechner und Software. Das Prozeßmodell hat den Charakter eines technisch sehr hochstehenden „Spielzeugs" und ist deshalb auch hervorragend geeignet, den Neuling in die Arbeitsweise eines solchen Systems einzuführen.

Programm für die mikrorechnergesteuerte Längenmessung

Vor zwei Jahren wurde von *Dipl.-Ing. Karl Armbruster* am Institut für Planungs- und Programmiertechniken von Prozeßrechnern der Universität Karlsruhe ein Versuchsaufbau zur rechnergesteuerten Längenmessung entwickelt. Die Prozeßsteuerung läuft über ein Kleinrechnersystem der Firma PCS, München. Dieser Rechner hat keinen parallelen E/A-Baustein und keine programmierbare Uhr. Die Signale von und zum Prozeßmodell laufen über eine digitale E/A-Karte, die über die Bus-Adresse eines E/A-Kanals angesprochen werden kann. Die auftretenden Interrupts müssen softwaremäßig analysiert werden, um sie zuordnen zu können. Die rechnergesteuerte Längenmessung wird seither an diesem Institut im Rahmen des Praktikums „Kopplung von Prozeßrechnern mit technischen Prozessen" vorgeführt.

Das diesem Beitrag zugrundeliegende Prozeßmodell ist eine Nachbildung dieser ersten Version, wurde jedoch an einigen Stellen, insbesondere im Bereich der Auswerfervorrichtung, mit einer stabileren Konstruktion versehen. Außerdem gestattete der Einsatz eines Z80-Mikrocomputersystems ein weiterentwickeltes Software-Konzept, da es über eine programmierbare Parallelschnittstelle und eine programmierbare Uhr verfügt.

1 Das Modell

Das Gesamtmodell besteht im wesentlichen aus drei Funktionseinheiten:

- der mechanischen, in Fischer-Technik aufgebauten eigentlichen Längenmeßanlage,
- dem Mikrorechner als Steuereinheit,
- einer Interface-Schaltung zur Anpassung und Erzeugung von Signalen für den Mikrorechner und umgekehrt.

Der mechanische Teil besteht aus einem Förderband, auf welchem die Werkstücke an zwei Lichtschranken vorbeitransportiert werden, und einer Auswerfervorrichtung zur Sortierung der gemessenen Werkstücke. Aus Gründen einer übersichtlichen funktionalen Trennung ist die Interface-Platine beim Prozeßmodell untergebracht. Dadurch kann der Lernende einfacher verstehen, welche Daten nun das Prozeßmodell und welche der Mikroprozessor liefert.

Das Interface generiert aus den durch Verdunkelung bzw. Freigabe der Lichtschranken entstehenden Signalen Interrupts, die vom Mikrorechner weiterverarbeitet werden können. Umgekehrt paßt es die vom Rechner ankommenden Signale an den jeweiligen Pegel der Schalteinheit, z. B. Auswerfermotor, an. Sämtliche Interrupts werden durch das kurzzeitige Aufleuchten farbiger Lämpchen am Ort des Interrupt-Verursachers, z. B. Lichtschranke, angezeigt.

Zur Steuerung und zur Verarbeitung der Signale vom Prozeßmodell wird ein Mikrocomputersystem vom Typ Z80-MCZ eingesetzt. Es enthält alle zur Längenmessung benötigten Bausteine (*Bild 1*). Der Vollständigkeit halber sei hier angemerkt, daß auch die Minimalkonfiguration, aus den beiden Baugruppen Z80-MCB (Micro-Computer-Board) und Z80-MDC (Memory-Disk-Controller) bestehend, für die Durchführung von Messungen ausreichen würde. Will man sich jedoch die Möglichkeit einer abgeänderten Versuchsanordnung oder Programmverbesserung vorbehalten, so benötigt man ein Floppy-Disk-Laufwerk und ein Datenterminal. Diese Konfiguration mit Z80-MCB, Z80-MDC und einem Floppy-Disk-Lauf-

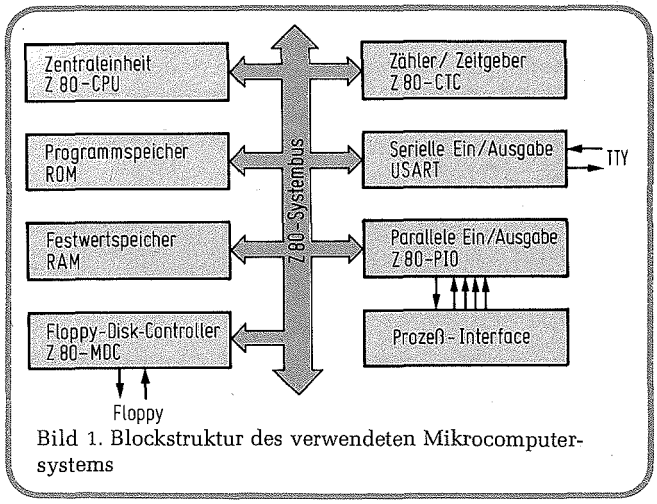


Bild 1. Blockstruktur des verwendeten Mikrocomputersystems

werk ist unter der Bezeichnung Z80-PDS (*Program Development Station*) erhältlich und genügt vollkommen den hier gestellten Anforderungen.

Auf die speziell in dieser Längenmessung benötigten Bausteine sei hier kurz näher eingegangen:

Z80-PIO:

Eine programmierbare parallele Ein/Ausgabe-Einheit mit zwei TTL-kompatiblen Datenports. Port A wird im Bit-Ein/Ausgabe-Modus (Mode 3) initialisiert, was den Vorteil hat, daß die 8 Leitungen einzeln als Ein/Ausgabe-Leitungen spezifiziert werden können. Im Versuch haben die Leitungen folgende Bedeutung: Die Bits 0...3 von Port A sind zur Eingabe, die restlichen zur Ausgabe vorgesehen. Dies wird der PIO durch Setzen des Steuerwortes 0CFH (0000 1111) angezeigt.

Bit 0: Signal von Lichtschranke 1, ansteigende Flanke;
Bit 1: Signal von Lichtschranke 2, ansteigende Flanke;
Bit 2: Signal von Lichtschranke 2, fallende Flanke;
Bit 3: Signal vom Auswerferendschalter, steigende Flanke;
Bit 4...7: ungenutzt.

Port B arbeitet im Byte-Ausgabe-Modus (Mode 0). Die Ausgabe von 01H stößt den Auswerfermotor an, entsprechend wird der Auswerfer angehalten, wenn 00H am Port B anliegt. Jedem möglichen PIO-Interrupt wird eine andere Interrupt-Adresse zugeordnet. Dadurch werden gleich die zugehörigen Interrupt-Service-Routinen angesprungen. Eine Analyse der eingehenden Interrupts wie beim PCS-Rechner entfällt somit.

Z80-CTC (Counter/Timer Circuit):

Ein programmierbarer Zähler/Zeitgeber-Baustein, der über vier voneinander unabhängige Kanäle verfügt. Dem Anwender steht jedoch nur der Kanal 2 zur Verfügung, da die drei anderen Kanäle für Floppy-Disk-Steuerung, Baudratengenerierung und Breakpoint-Zählfunktion belegt sind.

Um ein normiertes Zeitmaß von 10 ms zu erhalten, wird der CTC mit dem Steuerwort 0A7H geladen, wodurch die Betriebsart „Zeitgeber" eingestellt und der Systemtakt um den Faktor 256 geteilt wird. Die Periode C des Zeitgebers berechnet sich wie folgt (t_c = Systemtaktperiode, P = Vorteiler, TC = programmierbare Zeitkonstante):

$$C = t_c \cdot P \cdot TC = 400\ ns \cdot 256 \cdot 95 \approx 10\ ms.$$

Da im Steuerwort Bit 2 gesetzt ist, wird der nächste Befehl an den CTC, nämlich das Steuerwort 5FH = 95D, als Zeitkonstante interpretiert und in das Zeitkonstantenregister des Kanals 2 geladen. Der Baustein generiert nun genau alle 10 ms einen Interrupt. Zählt man diese Interrupts, so hat man eine normierte Zeitbasis mit einer Auflösung von $^1/_{100}$ s.

2 Meßprinzip

Als Datenaufnehmer dienen zwei Lichtschranken, die aus je einer Glühbirne und einem Fotowiderstand

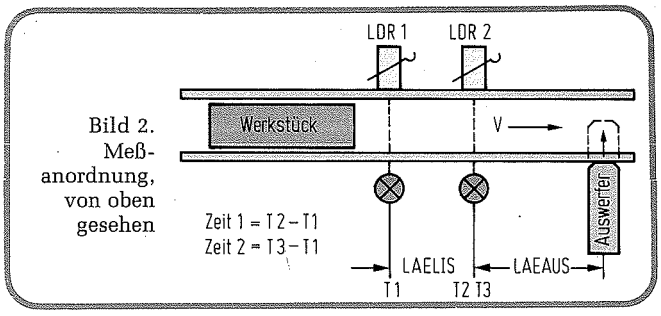


Bild 2. Meßanordnung, von oben gesehen

(LDR) aufgebaut sind (*Bild 2*). Dazwischen werden die zu messenden Werkstücke auf einem Transportband hindurch befördert. Zum Zeitpunkt T1 erreicht die Vorderkante des Werkstücks die erste Lichtschranke, zum Zeitpunkt T2 ist die Vorderkante des Werkstücks an der zweiten Lichtschranke angekommen. Die Rückkante gibt schließlich zum Zeitpunkt T3 den Lichteinfall an der zweiten Fotodiode wieder frei. Bezeichnet man T2...T1 mit ZEIT 1 und T3...T1 mit ZEIT 2, den Abstand zwischen den beiden Lichtschranken mit LAELIS und die Länge des zu messenden Werkstückes mit LAENGE, so lassen sich folgende einfachen Gleichungen für die Transportgeschwindigkeit angeben:

$$V = \frac{LAELIS}{T2 - T1} = \frac{LAELIS}{ZEIT\ 1} \quad (1)$$

$$V = \frac{LAELIS + LAENGE}{T3 - T1} = \frac{LAELIS + LAENGE}{ZEIT\ 2} \quad (2)$$

Unter der Voraussetzung einer während der Messung konstanten Transportgeschwindigkeit kann man durch Gleichsetzen von (1) und (2) die Länge des Prüflings erhalten:

$$LAENGE = \frac{LAELIS \cdot ZEIT\ 2}{ZEIT\ 1} - LAELIS =$$

$$\frac{(ZEIT\ 2 - ZEIT\ 1) \cdot LAELIS}{ZEIT\ 1} \quad (3)$$

Damit hat man das Problem der Längenmessung auf zwei Zeitmessungen reduziert, die sich durch Programmierung des CTC als normierter Zeitgeber auf einfache Art und Weise durchführen lassen. Jeder Interrupt von der Uhr bewirkt, daß eine Interrupt-Service-Routine angesprungen wird, in der ein Zähler dekrementiert wird. Dieser Zähler wird mit einem Wert von 3000, das entspricht 30 s, initialisiert. Nach Ablauf dieser 30 s ist der Zähler bei Null angekommen und bewirkt die Ausgabe der Zeit.

3 Programmsystem zur Längenmessung

Durch Eingabe des Kommandos %-MESSUNG wird die Prozedur-Datei MESSUNG von der Diskette geladen und ausgeführt (*Bild 3*). Das Programm meldet sich an der Konsole mit einer Ausgabe und einem kurzen Dialog, in dem der Benutzer zur Eingabe einiger Parameter aufgefordert wird. Das Meßprogramm benötigt vier Parameter (*Bild 4*), die alle zweistellig an der Konsoltastatur eingegeben werden müssen. Eine Fehlbedienung ist nicht möglich, da die Parametereingabe so lange wiederholt wird, bis zwei Ziffern angekommen sind. Mit den Parametern „obere (GRO)

und untere Toleranzgrenze (GRU)" wird das Meßintervall der Prüflinge festgelegt, d. h. alle außerhalb dieses Meßintervalls liegenden Werkstücke werden von einem Auswerfer vom Förderband gestoßen. Dadurch ist eine Selektion in „gute" und „schlechte" Prüflinge möglich. Mit Parameter 3 (LAELIS) wird dem Programm mitgeteilt, wie weit die beiden Lichtschranken voneinander entfernt sind, und Parameter 4 (LAEAUS) ist gleich dem Abstand zwischen der Lichtschranke LDR2 und dem Arm des Auswerfers. Diese beiden Parameter machen das Prozeßmodell flexibel und unabhängig von der mechanischen Anordnung von Lichtschranken und Auswerfer.

Nach Eingabe aller Parameter ist das Programm zur eigentlichen Längenmessung bereit; es wird nach jeder Messung zyklisch wiederholt (Bild 3). Jede einzelne Messung wird an der Konsole protokolliert, das Format ist in *Bild 5* dargestellt. Alle am Rechner ein-

NR.	IR1	IR2	ZEIT1	IR3	ZEIT2	VZZEIT	LAENGE	QUAL.KRIT.	IR3
00	*	*	01.23	*	03.90		60	GUT	
01	*	*	01.22	*	03.91		61	GUT	
02	*	*	01.24	*	03.96		61	GUT	
03	*	*	01.03	*	03.29		61	GUT	
04	*	*	00.88	*	02.85		62	GUT	
05	*	*	01.47	*	03.75	00.93	43	KURZ	*
06	*	*	01.25	*	03.19	00.74	43	KURZ	*
07	*	*	03.01	*	07.42	02.17	44	KURZ	*
08	*	*	02.97	*	06.12	01.71	32	KURZ	*
09	*	*	01.50	*	03.19	01.28	31	KURZ	*
10	*	*	01.06	*	04.46	00.15	89	LANG	*
11	*	*	01.25	*	05.31	00.17	90	LANG	*
12	*	*	01.24	*	05.46	00.17	95	LANG	*
13	*	*	01.24	*	05.66	00.17	99	LANG	*
14	*	*	01.24	*	05.84	00.17	NICHT MESSBAR		*
15	*	*	01.22	*	01.75	01.39	12	KURZ	*
16	*	*	01.19	*	03.75		60	GUT	
17	*	*	01.20	*	03.78		60	GUT	
18	*		TIMEOUT						
19	*	*	05.72		TIMEOUT				

Bild 5. Ausgabeformat der Messungen an der Konsole

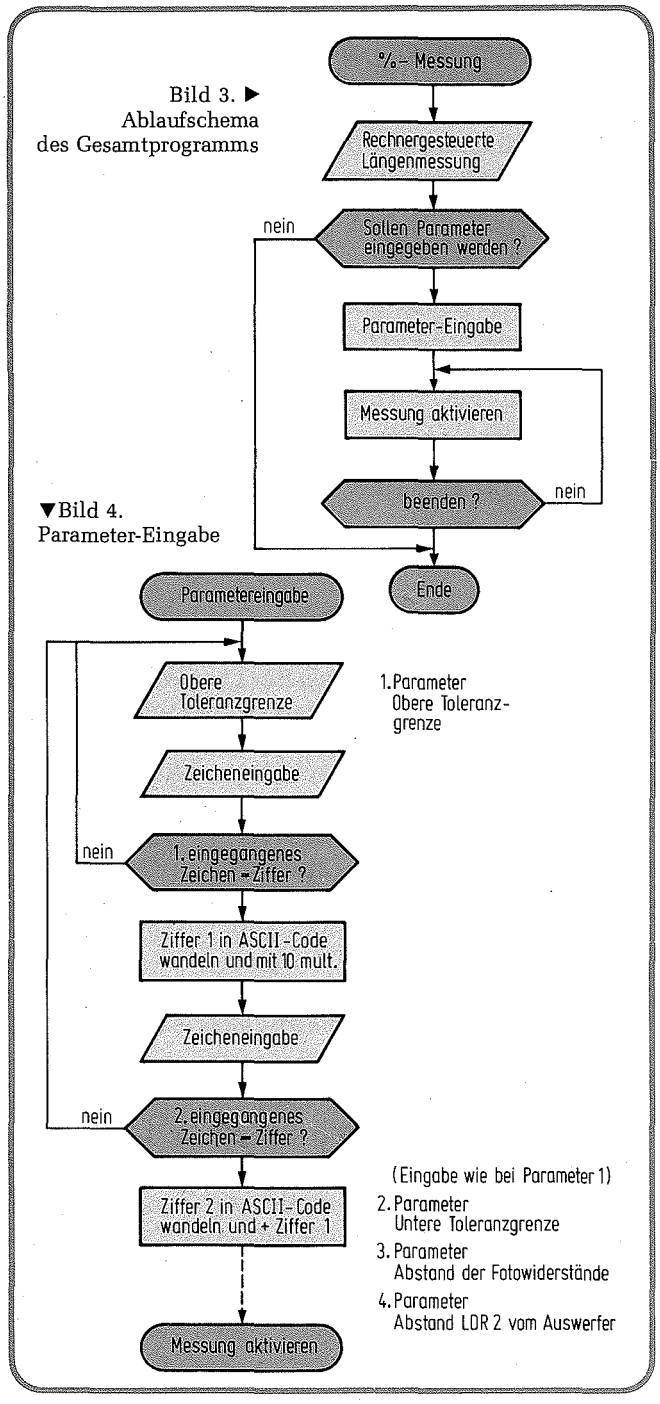


Bild 3. ▶ Ablaufschema des Gesamtprogramms

▼Bild 4. Parameter-Eingabe

gehenden Interrupts werden zusätzlich durch ein akustisches Signal (ASCII 07H) angezeigt.

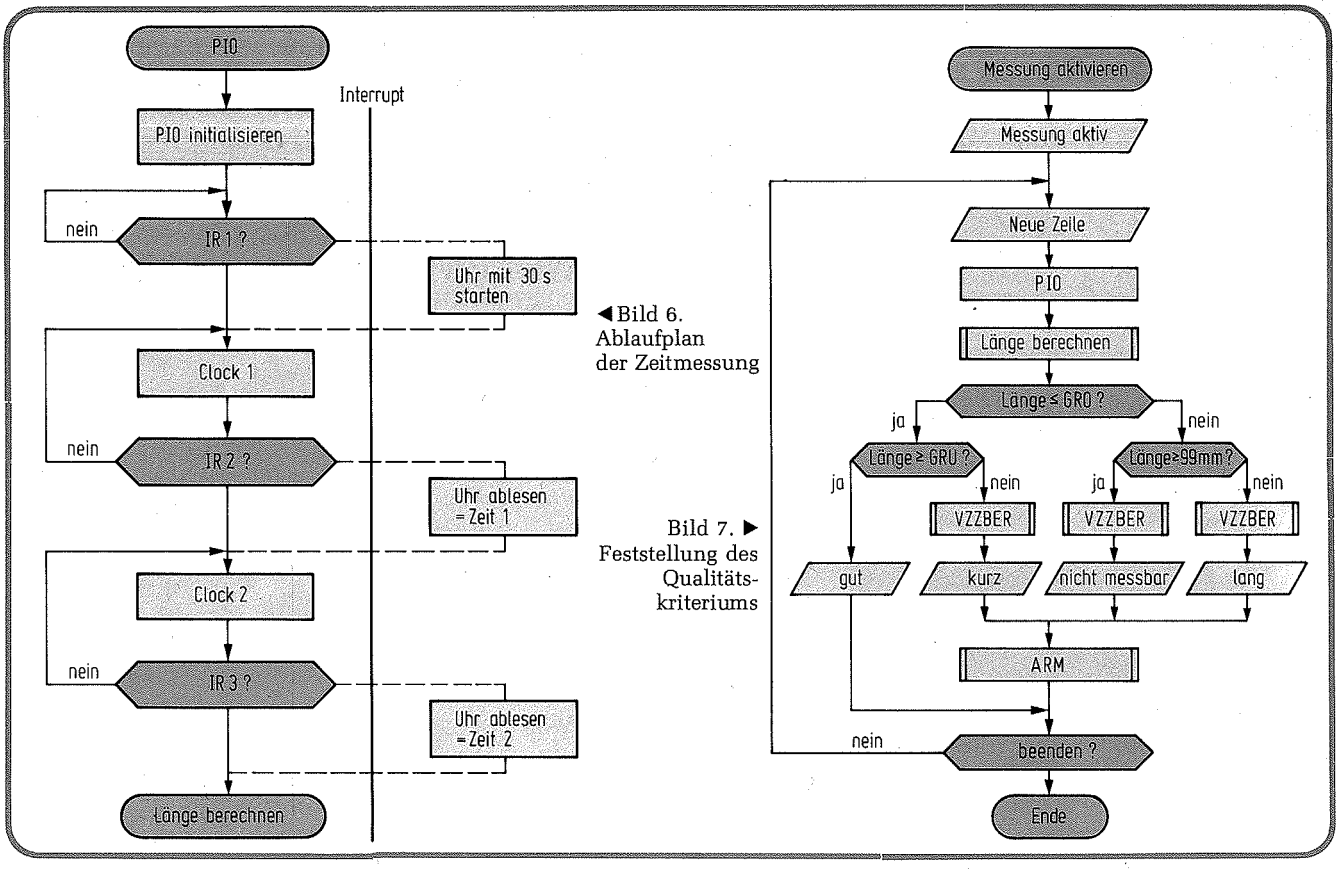
Der Ablauf einer Messung geht so vor sich: Das Werkstück liegt auf dem Förderband und erreicht zum Zeitpunkt T 1 die erste Lichtschranke (Bild 2). Das im Ein/Ausgabe-Interface erzeugte Signal durch Verdunkelung der Lichtschranke löst in der Z80-PIO einen Interrupt IR1 aus und bewirkt das Starten der Uhr. Als Anfangswert sind 30 s vorgesehen; das bedeutet, daß ein neuer Meßversuch unternommen wird, wenn nach Ablauf dieser Zeit das Werkstück nicht bei der zweiten Lichtschranke angekommen ist (siehe Versuch 18 in Bild 5). Zum Zeitpunkt T2 erreicht die Vorderkante des Werkstücks LDR2, wodurch der zweite Interrupt (IR2) ausgelöst wird. In der zugehörigen Interrupt-Service-Routine wird vom Initialwert 3000 der aktuelle Zählerstand der Uhr abgezogen, und man hat ZEIT 1, mit einem Fehler von $^1/_{100}$ s. Genau derselbe Vorgang wird wiederholt, wenn das Werkstück den Lichteinfall an der zweiten Lichtschranke wieder freigibt. Durch Subtraktion des jetzt aktuellen Zählerstandes vom Initialwert hat man auch den Wert für ZEIT 2 (*Bild 6*).

Damit sind alle Größen bekannt, die zur Berechnung der Länge nach Formel (3) notwendig sind. An die Längenberechnung schließt sich eine Plausibilitätskontrolle an. Vier verschiedene Qualitätskriterien sind vom Programm unterscheidbar (*Bild 7*). Befindet sich der Prüfling innerhalb der Toleranzgrenzen, so ist nach Ausgabe einer Meldung „GUT" das Programm sofort in der Lage, das nächste Werkstück zu messen. In allen anderen Fällen wird eine Verzögerungszeit berechnet (VZZBER), nach deren Ablauf der Auswerfer gestartet werden soll, um das Werkstück vom Band herunterzuschieben. Die Verzögerungszeit wird so berechnet, daß der Auswerferarm das Werkstück genau in der Mitte trifft. Es wäre sonst nicht möglich, sowohl sehr große als auch sehr kleine Werkstücke korrekt vom Band zu stoßen. Ist ein Prüfling länger als 99 mm, so gilt er als nicht mehr meßbar, da erstens die Ausgabe der Länge zweistellig erfolgt und zweitens das Messen längerer Stücke aufgrund der Abmessungen des Systems gar nicht mehr sinnvoll ist. Die Verzögerungszeit berechnet sich nach Formel (4).

$$\text{ZEIT} = \frac{\text{LAEAUS} - 0{,}5 \cdot \text{LAENGE}}{\text{V}} - \text{KONST} \qquad (4)$$

$$= \frac{(2 \cdot \text{LAEAUS} - \text{LAENGE})\text{ZEIT 1}}{2 \cdot \text{LAELIS}} - \text{KONST}$$

In der zweiten Form ist die Formel günstiger, da die Geschwindigkeit V nicht explizit als Variable zur Ver-



◀ Bild 6. Ablaufplan der Zeitmessung

Bild 7. ▶ Feststellung des Qualitätskriteriums

fügung steht und keine Gleitpunkt-Arithmetik verwendet wird. Außerdem ist in dieser Formel zu berücksichtigen, daß im Falle einer Werkstücklänge von mehr als dem zweifachen Abstand von Auswerfer zu LDR2 sich eine negative Verzögerungszeit ergäbe. Dies hätte zur Folge, daß wegen der Zweierkomplementdarstellung negativer Zahlen im Rechner ein völlig unsinniger Wert zustandekäme.

Nachdem die Verzögerungszeit berechnet wurde, wird die Uhr mit diesem Wert initialisiert, und sobald der Zähler den Wert 0 erreicht hat, wird die Ausgabe von 01H an Port B der PIO veranlaßt, welche über das Interface den Auswerfermotor einschaltet. Nach einer Vorwärts- und einer Rückwärtsbewegung wird ein Endschalter betätigt, der den Interrupt 4 erzeugt. Interrupt 4 bewirkt die Ausgabe von 00H an Port B und damit das Abschalten des Auswerfermotors.

Der Auswerfermechanismus wird beim vorliegenden Prozeßmodell also lediglich vom Rechner ein- und ausgeschaltet, alle anderen Funktionen, wie Polwenden an der richtigen Stelle, erfolgen automatisch und ohne Rechnersteuerung. Danach kann der nächste Meßversuch unternommen werden.

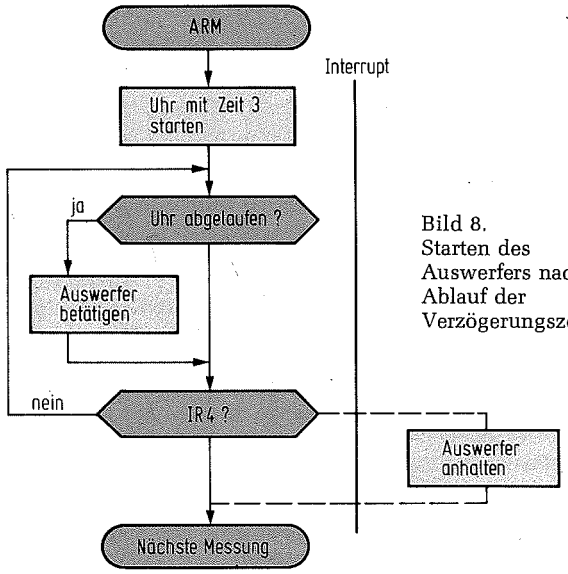


Bild 8. Starten des Auswerfers nach Ablauf der Verzögerungszeit

4 Ergebnisse

Aufgrund der komfortablen Protokollierung der Meßergebnisse und der dialogorientierten Inbetriebnahme des Programms, wird ein Speicherplatz von 2,5 KByte benötigt. Ohne Ein/Ausgabe wäre es nur 1 KByte lang. Die Genauigkeit des Meßverfahrens ist in erster Linie von den Gleichlaufeigenschaften des Förderbandantriebs abhängig. Bei dem verwendeten Fischer-Motor und -Getriebe darf natürlich nicht zuviel erwartet werden. Trotzdem sind Abweichungen vom Sollwert von mehr als 5 % äußerst selten. Eine weitere Fehlerquelle entsteht durch die Verwendung einfacher Ganzzahl-Arithmetik und Division ohne

Dipl.-Inform. Willi Haas stammt aus Oberkirch im Schwarzwald und studierte seit 1972 Informatik an der Universität Karlsruhe. Seit Anfang 1977 beschäftigt er sich mit Mikrocomputern. Jetzt ist er bei der Firma Siemens in der Zentralen Forschung und Entwicklung tätig.
Hobbys: Sport, Musik, Fotografie
Diensttelefon: (0 89) 67 82-35 32
ELEKTRONIK-Leser seit 1976

Rest. Außerdem könnte die Meßgenauigkeit durch einen kürzeren Zeitraster und einen geringeren Abstand der beiden Lichtschranken noch verfeinert werden. Schließlich könnte man das Programm noch in einem PROM unterbringen, wodurch Floppy Disk und RAM eingespart werden können. Das gesamte Rechnersystem bestünde dann nur noch aus einer einzigen Karte (Z80-MCB) – eine mit Sicherheit preislich sehr interessante Lösung für eine automatische Längenmessung mit Steuerung durch einen Mikrocomputer.

Literatur

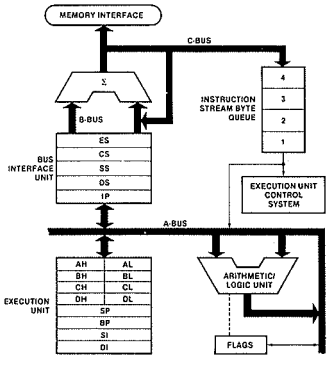
[1] *Armbruster, K., Ülzmann, W.*: Rechnergesteuerte Längenmessung. Praktikumsunterlagen Universität Karlsruhe.

[2] *Blomeyer, H.-P.*: Mikrocomputer-Technik. Hofacker Verlag, München.

[3] *Koch, G., Rembold, U.*: Einführung in die Informatik für Ingenieure. Hanser-Verlag, München.

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

8-bit-Mikroprozessor mit 16-bit-Architektur



Als neuesten Mikroprozessor stellte die Firma Intel auf dem Pariser Bauelementesalon erstmals in Europa den Typ 8088 vor: Dieser Mikroprozessor ist von der 16-bit-CPU 8086 abgeleitet, das heißt er verfügt intern über eine 16-bit-Architektur und arbeitet nach außen (über einen 8-bit-Datenbus) wie ein 8-bit-Mikroprozessor. Aus diesem Grund ist er vollständig softwarekompatibel mit dem 8086 und darüber hinaus – bezogen auf die Interface-Bausteine – hardwarekompatibel mit dem 8085 A. Nach Firmenangaben soll er der zur Zeit leistungsfähigste 8-bit-Mikroprozessor auf dem Markt sein, das heißt er ist fünfmal so leistungsfähig wie der 8080 A bzw. um den Faktor 2 besser als der 8085 A. Der neue Mikroprozessor, der in HMOS-Technik (verbesserte N-Kanal-MOS-Technik) hergestellt wird und in einem 40poligen DIL-Keramikgehäuse angeboten wird, kann bis zu 1 MByte Speicher direkt adressieren. Die Ausführung von Multiplikationen und Divisionen ist hardwaremäßig realisiert. Seine interne Taktrate beträgt 5 MHz. Anwendungsbeispiele für diesen Mikroprozessor sind Bildschirmgeräte, Terminals, Druckersteuerungen usw.

□ Vertrieb: Intel Semiconductor GmbH, Seidlstr. 20, 8000 München 2, Tel. (0 89) 55 81 41.

16-bit-Mikroprozessor für 4 MHz

Unter der Bezeichnung TMS 9900-40 bietet Texas Instruments einen schnellen 16-bit-Mikroprozessor an. Mit einer Zykluszeit von 250 ns ergibt sich gegenüber den früheren Typen eine Erhöhung des Datendurchsatzes um 33 %. Momentan sind aus dieser 4-MHz-Reihe die CPU TMS 9900-40, der Taktgenerator TIM 9904-40, der Peripherie-Interfacebaustein TMS 9901-40 und der asynchrone Übertragungs-Controller TMS 9902-40 erhältlich. Der Rechenbaustein ist in der Lage, eine Seitenumschaltung mit einem einzigen Befehl durchzuführen, um einen Speicherbereich über 64 KByte hinaus ansprechen zu können. Der Daten- und Adreßbus werden bei diesem System gemultiplext. Für die Prototypen-Entwicklung in den Entwicklungslabors existiert ein Universal-Emulator.

□ Vertrieb: Texas Instruments Deutschland GmbH, Haggertystr. 1, 8050 Freising, Tel. (0 81 61) 80-1.

Herkömmliche industrielle Klimaanlagen gehen für gewöhnlich ziemlich verschwenderisch mit der Energie um. Das im folgenden beschriebene System wurde mit einem Mikroprozessor M6800 ausgestattet, der für die richtige Raumtemperatur bei optimalem Wirkungsgrad sorgt. Die praktisch erzielbare Energieeinsparung hängt von den klimatischen Verhältnissen außerhalb des Gebäudes und den Aktivitäten im Inneren ab. Man erwartet pro Jahr eine Einsparung von durchschnittlich 40%.

R. Hodgson

Mikroprozessoreinsatz bei der Temperaturregelung mit Klimaanlagen

1 Einführung

Industrielle Klimaanlagen benutzen fast ausschließlich pneumatische Steuerungen, wofür es zahlreiche gute Gründe gibt. Obwohl vor Erscheinen des Mikroprozessors eine elektronische Steuerung möglicherweise wirtschaftlicher gewesen wäre, entschloß man sich erst heute in verstärktem Maße zu dieser Lösung. Der Hauptvorteil liegt darin, daß man die Computerleistung des Mikroprozessors „praktisch umsonst" zur Verfügung hat. Damit ist die Steuerung zur optimalen Energieausnutzung realisierbar. Wie gezeigt wird, sind die Einsparungen beachtlich. Zusätzlich liefert ein Mikroprozessorsystem statistische Daten, die eine bessere Planung und damit wiederum eine bessere Ausnutzung der Einrichtungen ermöglichen. Auch die Unterhaltungskosten werden reduziert, da das System Eigentests durchführen und sogar den Service automatisieren kann.

2 Die industrielle Klimaanlage

2.1 Prinzipielle Ausführung

Das Prinzip der Klimatisierung besteht darin, Luft einer gewissen Temperatur (und Feuchtigkeit) in einen Raum zu leiten, so daß die gewünschten Bedingungen darin aufrechterhalten werden. In der einfachsten Form wird Luft von außen durch eine Reihe von Heiz- bzw. Kühlelementen geblasen, die von Thermostaten im betreffenden Raum gesteuert werden. Die verbrauchte Luft aus dem Raum kann – gemischt mit etwas frischer Luft – wiederverwendet werden, vorausgesetzt, es fallen im klimatisierten Raum keine giftigen Gase an. Die Bestimmungen über den Grad der Lufterneuerung, den CO_2-Gehalt usw. zusammen mit den thermischen Anforderungen bilden die Kriterien für die Schaffung von Klimazonen innerhalb des Gebäudes, von denen jede von einer Klimatisiereinheit (air handling unit = AHU) versorgt wird. Die Kapazität der AHUs sowie der maximal erlaubte Grad der Luft-Wiederverwendung für jede Zone werden nach wirtschaftlichen Gesichtspunkten im Rahmen der Bestimmungen gewählt.

Die schematische Darstellung einer typischen AHU mit Luft-Wiederverwendung zeigt *Bild 1*. Darin zieht ein Absaugventilator die verbrauchte Luft aus der Klimazone. Sie wird in veränderlicher Menge (abhängig von der Stellung der Dämpfungsklappen D 1...D 3) mit frischer Luft gemischt. Die gemischte Luft passiert mit konstanter Strömungsgeschwindigkeit das Kühlaggregat, deren Kühlmittel (Wasser von 7 °C) so reguliert wird, daß sich eine feste Lufttemperatur (gewöhnlich 10 °C) einstellt. Vom Versorgungsventilator kann die Luft jetzt durch eine Reihe von Auslaßschächten geblasen werden. Vorher wird sie von einem Heizaggregat HA, das von den Raum-Thermostaten gesteuert wird, auf die gewünschte Temperatur (z. B. 22 °C) gebracht. Eine komplette AHU verfügt zusätzlich über einen Frostschutz und einen Abschaltmechanismus bei Fehlern in der Versorgung oder der Ventilatoren. Der Einfachheit halber wurde das in Bild 1 weggelassen. Außerdem wurde die Luftfeuchtigkeitsregulierung außer acht gelassen, da sie in Systemen mit Luft-Wiederverwendung meistens nicht benötigt wird.

Das Regelsystem dieser AHU kann sehr einfach sein. Es besteht beispielsweise aus der Regeleinheit für das Kühlaggregat (10 °C, fest) und einer weiteren für das Heizaggregat (22 °C). Allerdings wird dabei keine Rücksicht auf den Energieverbrauch für die Kühlung, die anschließende Aufheizung und die Umwälzung der Luft vorgenommen. Es ist z. B. eine völlig unsinnige Energieverschwendung, wenn die

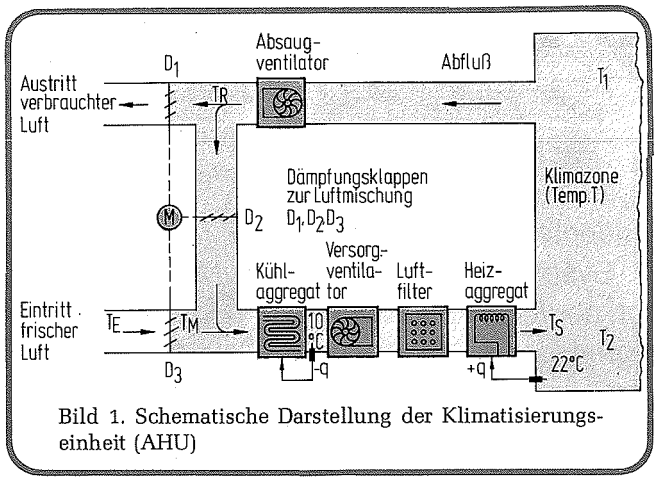


Bild 1. Schematische Darstellung der Klimatisierungseinheit (AHU)

Außentemperatur 22 °C beträgt, die Luft aber auf 10 °C gekühlt wird, bevor sie wieder auf 22 °C erwärmt und in ein Warenhaus gepumpt wird. Theoretisch müßte die Regelung so modifiziert werden, daß die Kühlung der Luft mit der Aufheizung koordiniert wird. Damit könnte das Energieaufkommen sowohl für das Heizen als auch für die Kühlung minimiert werden.

In der Praxis werden – bei konventionellen Systemen – drei Parameter nicht berücksichtigt, die in diesem Zusammenhang wichtig sind:

- die Wärmequellen und -senken in der Umgebung des Gebäudes;
- die Charakteristik der Wärmeabgabe und -aufnahme in der Klimazone;
- die Anwesenheit bzw. Abwesenheit von Personal in der Klimazone.

Zwar wäre es möglich, die herkömmlichen elektromechanisch-pneumatischen Regelsysteme daraufhin zu erweitern, aber in bezug auf die Komplexität und die Anpassungsfähigkeit ergibt sich mit elektromechanisch-elektronischen Systemen eine wesentlich brauchbarere Lösung.

2.2 Minimierung des Energieverbrauchs

Aus Bild 1 ergibt sich, daß man offensichtlich am besten damit beginnt, nicht zugleich zu heizen und zu kühlen. Die Ventilstellungen q werden von pneumatischen Antrieben eingestellt, die in einem elektronischen System von Strom/Pneumatik-Druckumsetzern angesteuert werden. Positive Werte von q werden dabei dem Heizaggregat mitgeteilt (+100 ≙ maximal Heizen), negative dem Kühlaggregat (–100 ≙ maximal Kühlen). Damit basiert die Temperaturregelung nicht länger auf der „Konkurrenz" zwischen Heizen und Kühlen. Der Durchfluß durch die Aggregate ist konstant, so daß die Temperaturregelung auch dadurch erreicht werden kann, indem man das Verhältnis M von wiederverwendeter zu frischer Luft variiert. Der Energieverbrauch ist dabei null (q = 0). Die Antriebe für die miteinander gekoppelten Dämpfungsklappen D 1...D 3 müssen in diesem Fall gesteuert werden. Die Heiz/Kühlaggregate werden erst dann eingeschaltet, wenn diese Maßnahme für die gewünschte Temperatur nicht mehr ausreicht.

Noch mehr Energie kann man sparen, wenn man zu bestimmten Zeiten die Anlage ganz ausschaltet, z. B. zwischen 22.00 und 6.00 Uhr, wenn der Raum nicht genutzt wird. Die Temperatur darf dann ruhig aus dem zulässigen Bereich wandern.

2.3 Die digitale Regelung

Die Zeitkonstanten einer Klimaanlagen-Regelung liegen in einem Bereich (Minuten bis Stunden), der für herkömmliche Analog-Regler höchst ungünstig ist. Das liegt vor allem an der zeitlichen Drift der verwendeten Bauelemente. Die digitale Regelung hat diesen Nachteil nicht – die Daten werden völlig driftfrei abgespeichert. Bei dieser Art der Regelung wird eine Reihe von Regelgrößen (Temperaturen) in festen Intervallen abgetastet, digitalisiert und nach dem festgelegten Regelgesetz numerisch zur Berechnung der Stellgrößen herangezogen. Die jeweiligen arithmetischen Operationen (Addition, Skalierung) und ihre Reihenfolge nach dem im Programm festgelegten Algorithmus werden mit Hilfe der sogenannten Z-Transformation ermittelt: im Gegensatz zur Laplace-Transformation mit der man die komplexe Übertragungsfunktion von Analog-Regelsystemen bestimmt und die – zumindest den Elektronikern – geläufiger ist. Das Ziel ist aber dasselbe: Die Regelung soll genau und stabil sein.

Um ein Regelsystem für ein gewünschtes Übertragungsverhalten auszulegen, müssen die dynamischen Eigenschaften aller Elemente im Regelkreis bekannt sein. Da sich diese Eigenschaften in den meisten Klimaanlagen nicht ändern, werden sie ermittelt, indem man die Sprungantwort am offenen Regelkreis mißt, wobei das Sprungsignal an geeigneten Punkten eingespeist wird.

Jetzt kann eine praktikable Abtastrate gewählt werden – gewöhnlich eine Größenordnung höher als die Zeitkonstante der schnellsten Komponente. Dann wird die Art der Z-Transformation anhand des gewünschten Übertragungsverhaltens des geschlossenen Kreises festgelegt. Häufig wird verlangt, daß eine Sprung- oder Rampenfunktion ohne Überschwingen ausgeregelt werden soll oder daß auf eine Rampenfunktion das Integral der Fehlerquadrate minimiert wird.

In realen Systemen können die tatsächlich benötigten Koeffizienten für die arithmetischen Operationen gewöhnlich nicht exakt vorhergesagt werden. Das liegt daran, daß nicht alle Komponenten, die den Regelkreis beeinflussen, gut genug beschrieben sind, um ein genaues mathematisches Modell zu liefern. Um trotzdem die gewünschte Charakteristik präzise zu erreichen, sollte dafür gesorgt werden, daß die Verstärkungsfaktoren und die Koeffizienten der Z-Transformation vor Ort einzustellen sind. Das kann entweder von Hand geschehen oder – in komplexeren Systemen – sogar von einem zusätzlichen Programm übernommen werden. Da die Anforderungen von Klimaanlagen im Hinblick auf die Antwortfunktion aber normalerweise nicht sehr hoch sind, nimmt man diese Einstellungen – falls sie überhaupt nötig sind – manuell vor und vermeidet zusätzlichen Programmaufwand.

3 Ein praktisches System

3.1 Regelungstechnische Gesetzmäßigkeit

Das im folgenden beschriebene System wird von einem Mikroprozessor gesteuert und ist seit etwa zwei Jahren bei der Firma Motorola in Toulouse für eine Halbleiter-Fabrikationsstätte im Einsatz.

Die ursprüngliche Anlage war ähnlich aufgebaut, wie es Bild 1 zeigt, mit einem Heizaggregat für die gesamte Zone. Ein eingehendes Studium der thermischen Gegebenheiten zeigte, daß die schnellste Komponente des Systems eine Zeitkonstante von 3 min hatte. Man entschied sich deshalb für eine Abtastrate von 20 s. Als Entwurfskriterium wurde festgelegt, daß die Sprungantwort (Änderung der Raumtemperatur) aperiodisch (ohne Überschwingen) sein sollte. *Bild 2* zeigt das Blockdiagramm eines Modells für das geplante Regelsystem. Die digitalen Regelglieder D 1 (Z) und D 2 (Z) legen die arithmetischen Operationen fest, die nach dem Regelalgorithmus erforderlich sind. Das Abtast- und Halteglied muß die Ventilpositionen q zwischen den Abtastzeitpunkten aufrecht erhalten. Die Soll-Temperatur T_C wird mit der Raumtemperatur T verglichen. Die Ergebnisse ΔT_I werden von D 1 (Z) benutzt, um die erforderliche Temperatur T_S (der einströmenden Luft) so zu berechnen, daß ΔT_I null wird. Damit kann das Verhältnis M von Frischluft zu wiederverwendeter Luft bestimmt werden. Es gilt für T_S der formale Zusammenhang

$$T_S = T_R\,(1\text{–}M) + T_E \cdot M + K$$

wobei K eine Konstante ist, die für die zusätzliche Heizwirkung des Versorgungsventilators steht.

Wenn M nicht mehr unmittelbar eingestellt werden kann, werden die Dämpfungsklappen auf den geeigneten Endanschlag eingestellt, und es wird der Wert q für zusätzliche Heizung bzw. Kühlung errechnet. Der durch die Auswertung von D 1 (Z) ermittelte Wert von T_S wird mit dem tatsächlichen Wert von T_S verglichen. Die Differenz ΔT_S wird dann von D 2 (Z) dazu benutzt, um die erforderlichen Ventilstellungen q des Heiz- bzw. Kühlaggregats zu berechnen, damit ΔT_S wieder zu null wird.

In dieser kurzen Beschreibung blieben die praktischen Probleme wie Hysterese, Abschalten bei Grenzwertüberschreitung, Initialisierung des Systems, Genauigkeit der Berechnungen usw. unberücksichtigt. Obwohl diese Dinge trival erscheinen, für die Stabilität des Systems sind sie von Bedeutung und müssen dementsprechend bei der Programmierung korrekt behandelt werden.

3.2 Das Mikroprozessorsystem

Die Regelelektronik basiert auf dem System MES 6800 mit Tastatur/Drucker-Interface, drei EPROM-Karten (6 KByte) für das Programm und zwei RAM-Karten (4 KByte) für die veränderlichen Daten. Alle Ein/Ausgaben laufen über Karten, die mit PIAs (Peripherie-Interface-Adapter) bestückt sind und von Buserweiterungs-Bausteinen getrieben werden. Grundsätzlich werden vier Arten von E/A-Operationen unterschieden, für die man sieben PIAs benötigt:

1. Informationsausgabe: drei 7-Segment-Anzeigeeinheiten und LED-„Signallampen";
2. Informationseingabe: Solltemperatureinstellung (Daumenradschalter), externe Uhr (Tag, Nacht, Wochenende), 50-Hz-Frequenz (als Abtast-Zeitbasis), Steuerschalter;
3. Ansteuerung der Dämpfungsklappen- und Ventilantriebe (Strom/Druck-Umsetzer);
4. Eingabe der Temperaturwerte von sechs Meßpunkten.

Zusätzlich läuft ständig ein Schreiber mit und dokumentiert die Aktivitäten des Systems. Damit werden die Außentemperatur T_E, die Innentemperatur T sowie die Einstellung der Dämpfungsklappen und der Ventile kontinuierlich aufgezeichnet.

Das Interface für die Ein/Ausgabe ist konventionell aufgebaut – die PIAs werden wie üblich mit Initialisierungsroutinen programmiert. Im Hinblick auf die Tatsache, daß Justierungsarbeiten und Fehlersuche an einem in Betrieb befindlichen System durchzuführen waren, wurde die bestehende pneumatische Steuerung weiterverwendet. Der Mikroprozessor steuert dieses System über eine Parallelschnittstelle (PIA) an, dem ein D/A-Umsetzer mit Stromausgang (MC 1408) nachgeschaltet ist. Dieser wiederum treibt einen Strom/Luftdruck-Umsetzer (Samson 6440), der zum pneumatischen System gehört (*Bild 3*).

Die Temperatur wird an sechs Punkten gemessen: T_E = Außentemperatur, T_R = Temp. der wiederverwendeten Luft, T_M = Temp. der gemischten Luft, T_S = Temp. der von der AHU gelieferten Luft, T_1 = Raumtemperatur 1, T_2 = Raumtemperatur 2.

T_M wird für den Regelalgorithmus nicht gebraucht. Dieser Wert dient dazu, vor einer drohenden Beschädigung des Kühlaggregats (Frost) zu warnen. Die Raumtemperatur wird an zwei weit entfernten Punkten gemessen. Der Mittelwert T_I wird dann berechnet und für den Regelalgorithmus verwendet. Als Temperaturfühler dienen Transistoren vom Typ BC585, deren Basis-Emitter-Spannung linear von der Temperatur abhängig ist. Dieser Wert wird in einen proportionalen Strom umgesetzt. Die Information wird dann über eine verdrillte 2adrige Leitung zur Regelelektronik übertragen. Dort wird sie wieder in eine Spannung und danach mit den Bausteinen MC1405/MC14435 in einen digitalen Wert umgesetzt, der von einem PIA entgegengenommen wird (*Bild 4*).

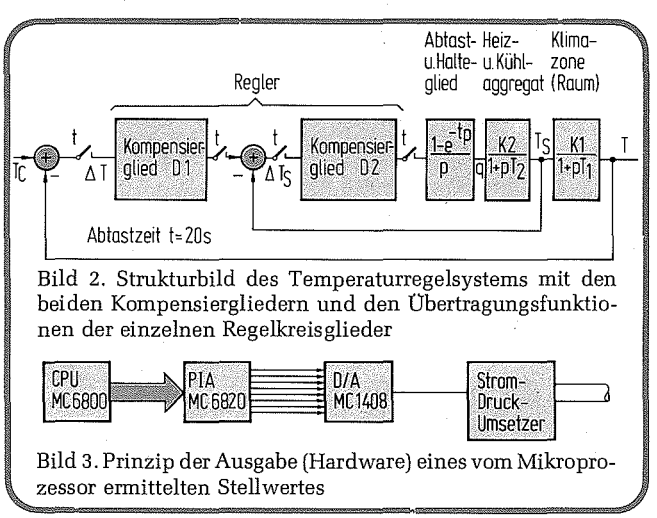


Bild 2. Strukturbild des Temperaturregelsystems mit den beiden Kompensiergliedern und den Übertragungsfunktionen der einzelnen Regelkreisglieder

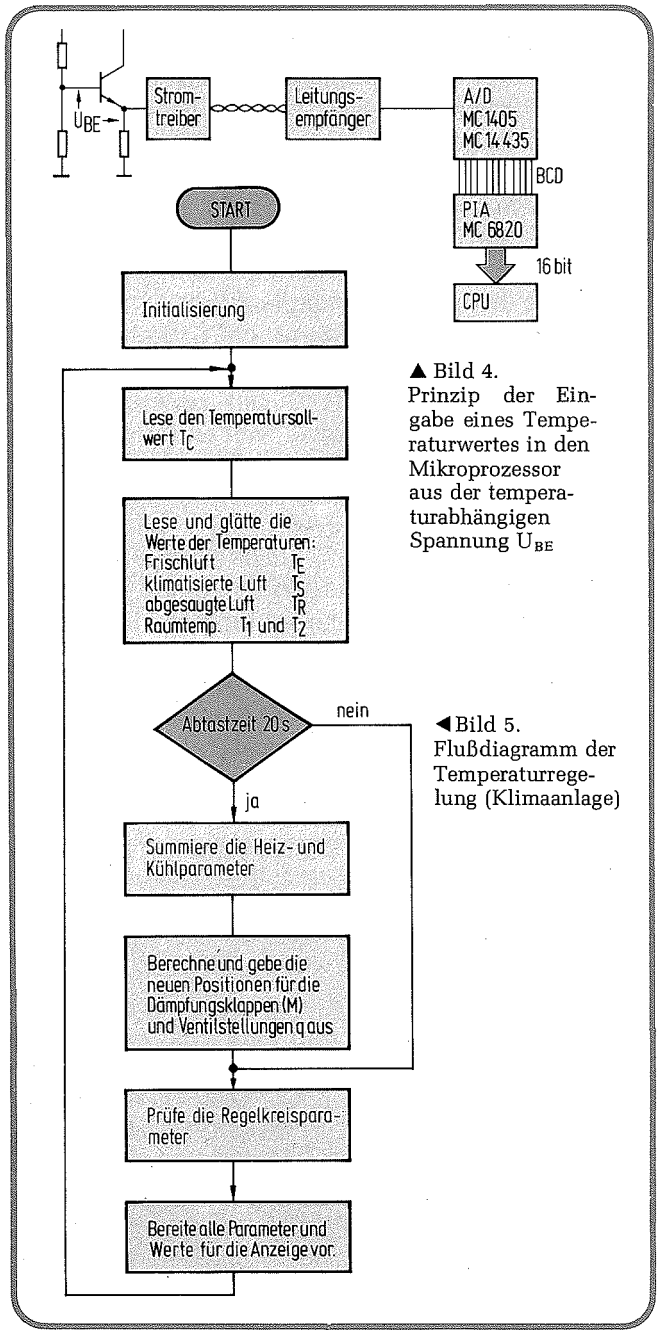
Bild 3. Prinzip der Ausgabe (Hardware) eines vom Mikroprozessor ermittelten Stellwertes

Der Mikroprozessor macht aus der 3stelligen BCD-Information (die im Multiplex vorliegt) eine 16-bit-Dualzahl, die er entsprechend rundet, damit für den Regelalgorithmus ein rauschfreies Signal zur Verfügung steht.

Bild 5 stellt das Flußdiagramm des Gesamtsystems dar.

Das Betriebspersonal muß sowohl die aktuellen thermischen Erfordernisse während des Tages als auch die durchschnittlichen täglichen Anforderungen kennen.

Der Mikroprozessor liefert z. B. den integrierten Wert von $T_E - 18$ °C für den gegenwärtigen Tag und den aufsummierten Gesamtwert. Eine andere Berechnung liefert die durchschnittlich „addierte bzw. subtrahierte Temperatur", die nötig war, um die Klimazone innerhalb der geforderten Grenzen zu temperieren.



▲ Bild 4. Prinzip der Eingabe eines Temperaturwertes in den Mikroprozessor aus der temperaturabhängigen Spannung U_{BE}

◀ Bild 5. Flußdiagramm der Temperaturregelung (Klimaanlage)

Tabelle. Energieverbrauch der Klimatisiereinheiten

	Mikroprozessor gesteuert		pneumatisch		Energie-einsparung
Woche	Heizen (kWh)	Kühlen (kWh)	Heizen (kWh)	Kühlen (kWh)	
11	500	1300	0	5 600	68 %
12	360	1200	0	5 100	69 %
13	20	600	0	3 700	82 %
14	0	1700	0	6 200	72 %
Gesamt	880	4800	0	20 600	72 %

4 Praktische Ergebnisse der Mikroprozessor-Regelung

Um das System zu überprüfen, wurde ein Raum von 3200 m^3 gewählt, der je zur Hälfte von einer AHU klimatisiert wurde. Fabrikationsprozeß und -einrichtungen waren in beiden Hälften die gleichen. Die zwei AHUs wurden mit Energiezählern im Heiz- und Kühlwasserkreis versehen. Das Integral über die Durchflußrate und der Temperaturaustausch der aktiven Flüssigkeit in jedem Aggregat ergeben die verbrauchte Energie. Eine AHU wurde dann mit dem Mikroprozessor-Regler verbunden, während die andere so belassen wurde, wie in Bild 1 dargestellt. Die *Tabelle* zeigt die während eines Zeitraums von vier Wochen (Frühjahr 1977) ermittelten Werte.

Es ist interessant, daß – obwohl beide Systeme überwiegend Kühlenergie verbrauchten (wärmeabgebender Prozeß im Raum) – beim Mikroprozessorsystem ein Heizanteil vorhanden ist. Das ist aber allein damit zu erklären, daß beim Wiedereinschalten die Temperatur unter den Sollwert gesunken war.

Nach Schätzungen des Betriebspersonals können jährliche Einsparungen von 40 % erreicht werden. Dieser Wert liegt deshalb niedriger als der in der Tabelle angegebene, weil bei höherer Außentemperatur (gegenüber der Solltemperatur), z. B. im Sommer, die natürliche Kühlung mit Frischluft nicht möglich ist. Beide Systeme brauchen dann gleich viel Energie.

Literatur

[1] *Jury, E. I.:* Sampled-data Control Systems. John Wiley 1958.

[2] *Aldridge, D.:* Analog-to-Digital Conversion Techniques with the M6800 Microprocessor System. Motorola Application Note AN-757.

[3] *Birk, H.:* Digitale Regelung durch Mikroprozessoren, ELEKTRONIK 1976, H. 3, S. 87...89.

[4] *Birk, H.:* Mikrorechner-Software für die digitale Regelung, ELEKTRONIK 1976, H. 5, S. 63...66.

Robin Hodgson, gebürtiger Engländer, studierte Elektrotechnik an der Universität Oxford bis 1958 und am Cranfield Institute of Technology bis 1960. Danach arbeitete er als Entwickler für Radar- und Mikrowellenanlagen und für verschiedene wissenschaftliche Meßsysteme. Seit 1972 ist er bei der Firma Motorola in Genf tätig, wo er z. Zt. Manager für Industrie-Applikationen in Europa ist.

Ursula
und Lutz May

Übersteigen Mikroprozessorprogramme in Maschinensprache eine gewisse Größe, dann ist es sinnvoll, einen Assembler zu verwenden. Er übersetzt die mnemonischen Befehlskürzel in den Maschinen-Code. Gewöhnlich erstellt man den Programmtext mit Hilfe eines Editors und korrigiert (editiert) ihn so lange, bis er formal richtig ist. Auf den nächsten Seiten wird demonstriert, wie Editieren und Assemblieren in der Praxis vor sich gehen. In ihrer Bedienung unterscheiden sich die entsprechenden Hilfsprogramme der verschiedenen Hersteller nur geringfügig.

Editieren und Assemblieren mit einem Mikrocomputer

1 Arbeitsmittel und Vorkenntnisse

Der verwendete Computer (CPU: 6802) setzt sich aus einem Bildschirmterminal und einer Rechnerkarte mit einem 16-KByte-RAM-Zusatz zusammen. Das Monitor-Programm stammt von Motorola. Es handelt sich um den ROM-Satz Minibug-3 und IOS. Zusammen benötigen sie einen Adreßbereich von 2 KByte. *Bild 1* zeigt den Adreßbereich der benötigten Baugruppen sowie den Anschluß der Tastatur und des Bildschirmterminals. Assembler und Editor [1] sind auf Kassetten erhältlich. Sie wurden von der Firma TSC (Technical Systems Consultants) entwickelt.

Da die Software für das Monitor-Programm Mikbug-7 ausgelegt ist, muß man nach dem Laden des Assemblers und des Editors an den vom Hersteller beschriebenen Stellen einige Sprungadressen ändern. In *Tabelle 1* sind die notwendigen Änderungen aufgeführt.

Um mit einem Assembler arbeiten zu können, muß man den Mnemonik-Code des verwendeten Mikroprozessors gut kennen. Er ist leicht erlernbar, da er aus Abkürzungen der kompletten Befehlsbezeichnungen besteht. Dazu einige Beispiele:

Load Accumulator A: LDA A
Logical Shift Right: LSR
Branch if Minus: BMI

Als zweites müssen die verschiedenen Adressierungsarten des Mikroprozessors bekannt sein [2]: immediate, direct, index und extended. Sind diese Begriffe noch fremd, muß man sie in der Praxis studieren und üben.

2 Bedienung des Assemblers

Alle Anweisungen, die der „TSC 6800 Mnemonic Assembler" zu bearbeiten hat, müssen ihm im 7-bit-ASCII-Code mitgeteilt werden, wobei das Paritätsbit (höchstwertiges Bit) nicht gesetzt werden darf. Jede Befehlszeile, die der Assembler bearbeiten soll, wird aus vier Informationsblöcken gebildet: Label, Operator *(mnemonic)*, Operand und Kommentar. Die Zeile muß mit einem Wagenrücklauf *(carriage return, $0D_{16}$)* abgeschlossen werden.

Label:

a) Wird ein Label verwendet, so muß er in der ersten Spalte der Befehlszeile beginnen (nicht erst nach einigen Leerzeichen).

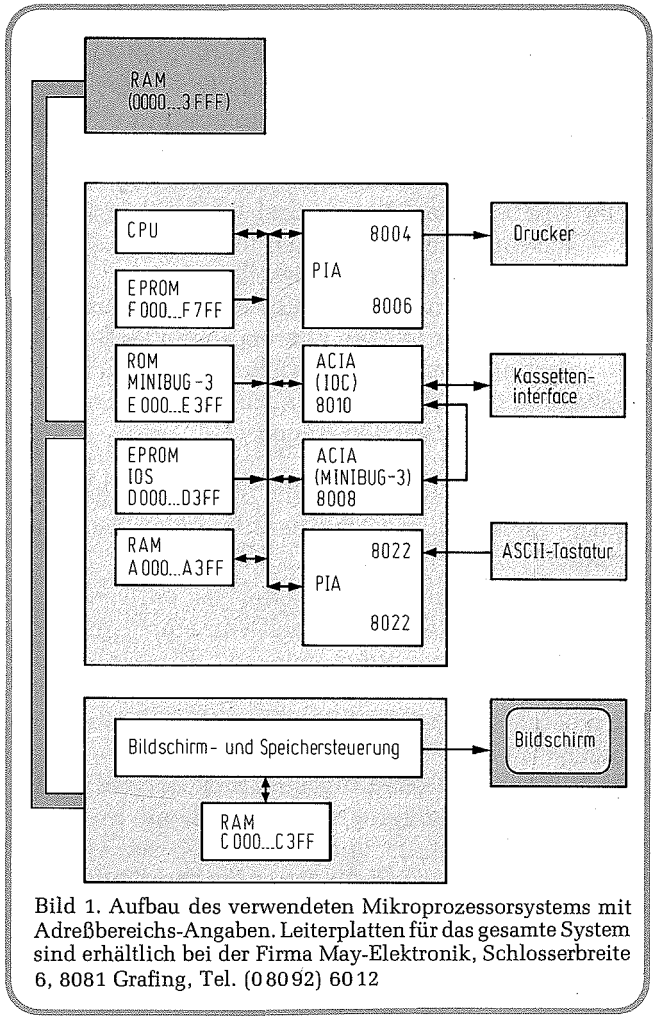


Bild 1. Aufbau des verwendeten Mikroprozessorsystems mit Adreßbereichs-Angaben. Leiterplatten für das gesamte System sind erhältlich bei der Firma May-Elektronik, Schlosserbreite 6, 8081 Grafing, Tel. (08092) 6012

Tabelle 1. Adressenänderungen im Assembler- und Editor-Programm

Assembler:	031C:	E0D0 → DC00
	0321:	E1D1 → E108
	0324:	E1D1 → E108
Bei Verwendung eines Anadex-Druckers müssen zusätzlich diese Befehle eingesetzt werden:		
	11D1:	00
	11D2:	0A
	11D30	0C
	11D4:	04
	1143:	04
	07C5:	38
Editor:	0207:	E1AC → E11F
	020A:	E1D1 → E 108
	020D:	E1AC → E11F
	0210:	E1D1 → E 108

Tabelle 2. Definition der Basis für einen Operanden-Ausdruck

Basis	Vorsilbe	Nachsilbe	Bemerkung
Dezimal	keine	keine	Dezimalzeichen
Binär	%	B	0,1 erlaubt
Oktal	@	O oder Q	0...7 erlaubt
Sedezimal	$	H	0...9, A...F
ASCII	'	keine erlaubt	ASCII-Code

b) Ein Label besteht aus einer Kombination von Buchstaben (A...Z) und Zahlen (0...9).
c) Jeder Label beginnt mit einem Buchstaben (A...7)
d) Nur die ersten sechs Zeichen eines Labels werden vom Assembler berücksichtigt, alle weiteren werden ignoriert.
e) Ein Label darf im Quellenprogramm nur ein einziges Mal im Label-Block stehen.

Operator:

a) Der Operator besteht aus drei Buchstaben (A...Z). Danach muß ein Leerzeichen folgen (Space). Ausnahme: Wird in Punkt b) beschrieben.
b) Definiert der Mnemonik-Befehl einen Akkumulator wie LDA A oder LDA B, können auch vier Buchstaben hintereinander geschrieben werden: LDAA, LDAB. In diesem Fall muß nach dem vierten Buchstaben ein Leerzeichen folgen.

Operand:

a) Im Operanden-Feld kann ein Adressierungsart-Indikator oder ein, noch zu definierender, Ausdruck stehen.
b) Der Adressierungsart-Indikator ist entweder ein „Pound"-Zeichen (#), gefolgt von einer „immediate" Adresse oder eine „indexed" Adresse, gefolgt von einem Komma und einem X (,X). Wie die Adressen auszusehen haben, wird noch beschrieben.
c) Wenn kein Adressierungsausdruck notwendig ist, kann das Operanden-Feld leer bleiben.

Kommentar:

a) Das Kommentar-Feld dient zur Beschreibung eines Arbeitsvorganges, um sich beim späteren Durchlesen der Programmliste besser orientieren zu können. Dieses Feld wird vom Assembler nicht bearbeitet.
b) Zum Schreiben des Kommentars dürfen alle Zeichen, angefangen vom Leerzeichen (20_{16}) bis zum DEL-Zeichen ($7\ F_{16}$) verwendet werden.

Adressen und Ausdrücke:

Sie bestehen aus einer Kombination von Zeichen und Symbolen, ausgenommen die vier arithmetischen Operatoren +, –, x, /. Plus- und Minuszeichen dürfen aber als Vorzeichen verwendet werden. Adressen im Operandenfeld können auch Labels sein, die dann aber auch in irgendeiner Befehlszeile im Label-Block einmal erscheinen müssen. Leerzeichen müssen in einem Ausdruck nicht enthalten sein. Man kann die Basis eines Operanden-Ausdrucks bestimmen, indem man bestimmte Vor- oder Nachzeichen im Ausdruck setzt *(Tabelle 2)*.

2.1 Spezielle Assembler-Befehle

Zusätzlich zu den 72 Mnemonik-Befehlen des Mikroprozessors kennt der Assembler elf sogenannte Pseudo-Operatoren. Es handelt sich dabei um organisatorische Anweisungen, die den Assembler betreffen. Die elf Operatoren mit ihren englischen Definitionen lauten:

FCC *form constant character*
FCB *form constant byte*
FDB *form double byte*
SPC *insert spaces in output listing*
OPT *activate or deactivate assembler options*
PAG *skip to next page of output*
ORG *define new origin* (PC)
EQU *assign value to symbol*
END, MON *signal end of source program*
NAM, TTL *specify name or title*
RMB *reserve memory bytes*

Im einzelnen haben sie folgende Funktionen:

FCC: Dieser Befehl fordert den Assembler auf, einen Zeichenausdruck im ASCII-Code in das Programm mit zu übernehmen, z. B. um eine Tabelle oder einen Satz in das Programm einzubauen. Eine Befehlszeile sähe dann so aus:

Label FCC 'Text' Kommentar

Die Apostrophe sind notwendig, um den ASCII-Textbereich zu charakterisieren. Ein Label muß nicht verwendet werden. Das gilt auch für die zwei folgenden Operator-Anwendungen. Der Text darf höchstens 255 Zeichen lang sein.

FCB: Mit diesem Pseudo-Operator wird aus maximal zwei Zahlen, die im Ausdruck näher bezeichnet werden, ein 8-bit-Wert (Byte) gebildet und in das Programm eingearbeitet. Folgt nach dem Zahlenausdruck ein weiterer, der durch ein Komma abgetrennt ist, wird der folgende Zahlenausdruck ebenfalls bearbeitet.

Format:

Label FCB Ausdruck 1, Ausdruck 2, ...Ausdruck N
Ausdrücke: $ 0A oder 00 oder $ 20 usw. N darf nicht größer als 255 sein.

FDB: Dieser Operator hat die gleiche Funktion wie FCB. Der Unterschied ist, daß nicht ein 8-bit-, sondern ein 16-bit-Wert gebildet wird (zwei Byte).

Format:
Label FDB Ausdruck 1, Ausdruck 2, ...Ausdruck N
Hierbei dürfen maximal 127 Ausdrücke geschrieben werden.

SPC: Es wird damit dem Assembler mitgeteilt, wieviele Leerzeichen beim Ausdrucken der Programmliste an dieser Stelle gesetzt werden sollen.
Format:

SPC Ausdruck

Es darf hier kein Label verwendet werden. Ist der Ausdruck eine Null, dann wird nur ein Leerzeichen eingefügt. Der SPC-Befehl ist nur auf einer Druckerseite wirksam. Sind mehr Leerzeichen einzufügen, als auf der momentan angewählten Seite noch Platz finden, werden die überzähligen ignoriert.

OPT: Hiermit werden organisatorische Assembler-Zusatzbefehle aktiviert oder desaktiviert.
Format:

OPT Zusatz 1, Zusatz 2, ...Zusatz N

In *Tabelle 3* werden die möglichen Zusätze mit ihrer englischen Definition aufgeführt. Ihre Funktion wird in den Beispielprogrammen erklärt. Die Angabe „default" bedeutet, daß die Funktion auch gewählt wird, wenn der Assembler keine gegenteilige Anweisung bekommt, der Zusatz selbst aber fehlt.
Diese Assembler-Zusätze werden erst im zweiten Teil des Assembler-Durchlaufs bearbeitet (der Assembler besteht aus drei Arbeitsblöcken, die er nacheinander durchläuft)

PAG: Soll auf jeder ausgedruckten Seite eine fortlaufende Seitenzahl mit erneutem Titelausdruck erfolgen, dann wird der Operator PAG eingesetzt. In der gleichen Befehlszeile darf kein Label geschrieben werden. Allgemein sei bemerkt, daß es sinnvoll ist, den Titel eines Quellenprogramms sowie organisatorische Assembler-Befehlszusätze an den Anfang des Programms zu schreiben.

ORG: Hiermit wird dem Assembler die Startadresse bekanntgegeben, an der das zu übersetzende Programm beginnen soll.
Format:

ORG Ausdruck

Es ist hier kein Label erlaubt. Der Ausdruck ist eine vierstellige Sedezimaladresse. Wird der Operator ORG nicht verwendet, dann nimmt der Assembler an, daß das Quellprogramm später mit der Speicheradresse 0000 beginnen soll.

EQU: Dieser Operator gibt an, welcher reelle Wert für ein verwendetes Symbol, das man für die Benennung einer Adresse oder einer Konstanten benutzt hat, eingesetzt werden soll. Ein Symbol darf nur einmal definiert werden.
Format:

Label EQU Ausdruck

Hier muß ein Label zur Symbolgleichsetzung geschrieben werden.

END oder MON: Dieser Befehl muß am Schluß des Quellprogramms geschrieben werden, damit der Assembler seine Arbeit beenden kann. Sonst hörte er nicht auf zu arbeiten und zerstört den Speicherinhalt.
Format:

END

Tabelle 3. Zusätze zum Befehl OPT mit engl. Originalbezeichnungen

SYM	print sorted symbol table after the listing (default)
NOS	do not print the symbol table
GEN	print all code generated by FCB, FDB, or FCC (default)
NOG	print only one line for each FCB, FDB, or FCC
LIS	print the assembled source listing (default)
NOL	suppress the printing of the source listing
PAG	enable page formatting and numbering
NOP	disable page mode (default)
MEM	enable storing of object code in memory
NOM	disable storing of object code in memory (default)
TAB	enable the production of MIKBUG* object tape
NOT	disable the production of MIKBUG* object tape (default)

* Bezeichnung für ein bestimmtes Monitor-Programm

Weder Label noch Kommentar dürfen in dieser Befehlszeile stehen.

NAM oder TTL: Damit bei einem mehrseitigen Programm jede Seite einen Titel bekommt, um es später besser einordnen zu können, schreibt man anch dem Operator NAM oder TTL den gewünschten Titel-Text (nur sinnvoll, wenn auch der Befehl PAG gesetzt wird). Der Titel darf nicht länger als 32 Zeichen werden.
Format:

TTL Text für den Titel

Es darf kein Label gesetzt werden.

RMB: Damit wird der Assembler aufgefordert, einen Speicherbereich für später zu definierende Werte zu reservieren.
Format:

Label RMB Ausdruck

Ein Label muß verwendet werden. Der Ausdruck ist eine 16 bit breite Zahl, die die Größe des Speicherbereichs angibt.

2.2 Der Editor als Arbeitshilfe

Das Quellprogramm wird über eine ASCII-Tastatur in freie Speicherplätze eingeschrieben, wobei die vorher genannten Regeln beachtet werden müssen. Der Assembler wird erst nach Abschluß des Programmeinschreibens gestartet, vorher muß man ihm einige Daten mitteilen:

FILBEG: Quellprogramm-Start ($ 0044 und $ 0045). In den beiden Speicher-Adressen wird die zwei Byte breite Quellprogramm-Startadresse eingetragen.

FILEND: Quellprogramm-Ende ($ 0046 und $ 0047). Hier wird die Adresse des ersten Bytes nach dem letzten Byte des Quellprogramms eingesetzt.
Symbol Tabel limits: Hier wird die zu erwartende Größe der vom Assembler zusammenzustellenden Symboltabelle eingetragen. Berechnung:
N = Anzahl der Symbole (z. B. Labels)
N · 16 Byte = Symboltabellengröße

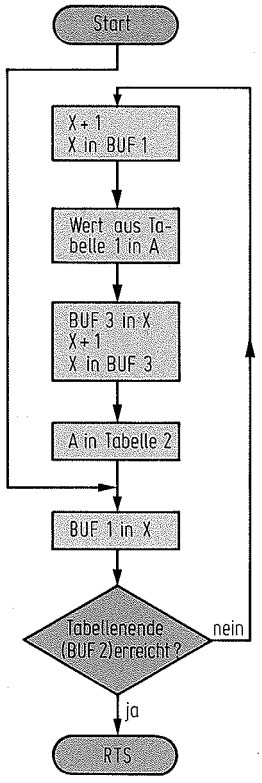


Bild 2. Flußdiagramm als Arbeitsvorlage für die Erstellung des „Transportprogramms"

```
G 1700
NEW FILE:
    1.00=**  TEST PROGRAMM 1  **
    2.00=
    3.00=** UNTERPROGAMM ZUM TRANSPORT EINES DEFINIERTEN **
    4.00=** SOFTWARE-ABSCNITTS IN EINEM ANDEREN SPEICHER **
    5.00=
    6.00=;TTL;PROGRAMMTRANSPORT
    7.00=;ORG;$A100;STARTADRESSE
    8.00=;OPT;SYM,GEN,LIS,PAG;MEM
    9.00=BUF1;EQU;$A0A0
   10.00=BUF2;EQU;$A0A2
   11.00=BUF3;EQU;$A0A4
   12.00=;JMP;PRGBGN;ZUM PROGRAMMSTART
   13.00=TRANS2;INX;;X+1= X
   14.00=;STX;BUF1;SICHERE X IN BUF 1
   15.00=;LDAA;0,X;WERT AUS TABELLE 1 IN X
   16.00=;LDX;BUF3;MOM.TABELLENSTAND 2 IN X
   17.00=;INX;;X+1= X
   18.00=;STX;BUF3;NEUER TABELLENSTAND IN BUF 3
   19.00=;STAA;0,X;WERT AUS A IN TABELLE 2
   20.00=
   21.00=PRGBGN;LDX;BUF1;MOM.TABELLENSTAND 1 IN X
   22.00=;CPX;BUF2;TABELLE 1 ZU ENDE?
   23.00=;BNE;TRANS2;SPRINGE WENN NICHT
   24.00=;RTS;;RUECKSPRUNG ZUM HAUPTPROGRAMM
   25.00=
   26.00=;END
SET TAB=';'
£TAB 9 16 26
£^E!
?
£^EXP!
£^P!
    1.00=**  TEST PROGRAMM 1  **
    2.00=
    3.00=** UNTERPROGAMM ZUM TRANSPORT EINES DEFINIERTEN **
    4.00=** SOFTWARE-ABSCNITTS IN EINEM ANDEREN SPEICHER **
    5.00=
    6.00=          TTL     PROGRAMMTRANSPORT
    7.00=          ORG     $A100       STARTADRESSE
    8.00=          OPT     SYM,GEN,LIS,PAG;MEM
    9.00=BUF1      EQU     $A0A0
   10.00=BUF2      EQU     $A0A2
   11.00=BUF3      EQU     $A0A4
   12.00=          JMP     PRGBGN      ZUM PROGRAMMSTART
   13.00=TRANS2    INX                 X+1= X
   14.00=          STX     BUF1        SICHERE X IN BUF 1
   15.00=          LDAA    0,X         WERT AUS TABELLE 1 IN X
   16.00=          LDX     BUF3        MOM.TABELLENSTAND 2 IN X
   17.00=          INX                 X+1= X
   18.00=          STX     BUF3        NEUER TABELLENSTAND IN BUF 3
   19.00=          STAA    0,X         WERT AUS A IN TABELLE 2
   20.00=
   21.00=PRGBGN    LDX     BUF1        MOM.TABELLENSTAND 1 IN X
   22.00=          CPX     BUF2        TABELLE 1 ZU ENDE?
   23.00=          BNE     TRANS2      SPRINGE WENN NICHT
   24.00=          RTS                 RUECKSPRUNG ZUM HAUPTPROGRAMM
   25.00=
   26.00=          END
```

Bild 3. Eingabe des Quellprogramms und Ordnen durch den Editor

Man muß dem Assembler den Speicherteil angeben, in dem er diese Tabelle aufstellen kann:
LBLBEG ($ 0040...$ 0041)
LBLEND ($ 0042...$ 0043)
LINBYT, Skip count ($ 0048): Wenn die Befehlszeile mit Hilfe eines Editors geschrieben wurden, dann steht an jedem Zeilenanfang eine Zahl, die der Editor zur Arbeitserleichterung eingesetzt hat. Der Assembler erwartet aber bei den ersten Werten einen Label. In diesem Fall teilt man dem Assembler mit, wieviele Bytes er vom Zeilenstart an ignorieren soll.

MEMPTR, Memory pointer ($ 0049). Hier wird die Startadresse eingetragen, an der das endgültige Programm nach der Übersetzung vom Assembler eingetragen werden soll (nur sinnvoll, wenn im Quellprogramm der Operator MEM verwendet wurde).

Bei der hier verwendeten Software wurde nach den Angaben des Programmherstellers der Assembler mit einem Editor verknüpft und in einen 16-KByte-Speicher eingeladen. Beim Aufrufen des Assemblers werden vom Editor alle notwendigen Daten (FILBEG..., LBLBEG..., LINBYT..., usw.) dem Assembler mitgeteilt, so daß man sich darum nicht mehr kümmern muß.

Zudem sorgt der Editor dafür (sofern man ihm die notwendigen Angaben gemacht hat), daß die einzelnen Blöcke in einer Befehlszeile (Label, Operator, Operand und Kommentar) an der richtigen Stelle stehen und den definierten Abstand zueinander beibehalten.

Sollen Werte eingefügt, geändert, gesucht oder gelöscht werden, so gibt es eine ganze Reihe von Befehlen, die dies auf elegante Art ermöglichen. Um den Rahmen dieser Einstieghilfe nicht zu sprengen, werden nur die Anweisungen besprochen, die in den Beispiel-Programmen verwendet werden.

3 Beispielprogramme

An einigen praktischen Beispielen soll der Arbeitsablauf bei der Programmerstellung bis zum Übersetzen mit dem Assembler verständlich gemacht werden.

3.1 Aufgabe

Folgende Aufgaben gilt es zu lösen: Es soll ein Programm geschrieben werden, das einen beliebigen Speicherbereich in einen zu definierenden Speicherteil überträgt. Es sind dazu drei Adressen nötig, die bekanntgegeben werden müssen. Diese drei Werte sind vor dem Programmstart in Puffer (BUF 1...3) einzuladen:
BUF 1, $ A0A0 = Startadresse des zu übertragenden Datenblocks
BUF 2, $ A0A2 = Endadresse (letztes Byte) des Datenblocks
BUF 3, $ A0A4 = Startadresse des neuen Speicherbereichs
Man wählt einen Lösungsweg, der mit Hilfe eines Flußdiagramms dargestellt wird. Dies ist bei der Programmerstellung mit Assembler und Editor der verbleibende Problem- oder Arbeitsteil; alles andere kann

nach mehreren Versuchen und Übungen als Routine betrachtet werden. Das setzt natürlich voraus, daß der Lösungsweg, der im Flußdiagramm dargestellt wird, auch funktionsfähig ist, was man aber oft erst nach dem Assemblieren herausfinden kann.

3.2 Flußdiagramm

Das im *Bild 2* veranschaulichte Programm zeigt den funktionellen Ablauf des „Programmtransportes": In das Indexregister wird die Startadresse (aus BUF 1) geladen, die vom Akkumulator A als Basisadresse betrachtet wird. Mit dem Wert aus BUF 2 wird nun überprüft, ob der ganze Datenblock schon übertragen wurde; wenn nicht, erfolgt ein Sprung zum Transportteil, wobei dort zuerst die Datenblockstartadresse um eins erhöht und in BUF 1 abgelegt wird. Der weitere Ablauf ist leicht erkennbar. Zu bemerken ist noch, daß mit dem Begriff „Tabelle 1" der zu tranportierende Datenblock gemeint ist, „Tabelle 2" steht für den neuen Speicherbereich.

3.3 Editieren

Der Ausdruck *(Bild 3)* stellt den weiteren Arbeitsablauf dar. Zuerst wird der Editor angewählt. Dies geschieht durch die Anweisung „Springe zur Adresse 1700_{16} und bearbeite das dort stehende Programm (G 1700). Daraufhin meldet sich der Editor mit dem Begriff „New File: 1.00 =". Der Editor hat nun die erste Eingabezeile numeriert und wartet auf den dort einzutragenden Text. In diesem Fall wird der Name des Programms eingetragen, wobei man Textzeilen, die keinerlei Befehle beinhalten, mit Sternchen (Asterisk) einsäumen muß. Betrachtet man eine Zeile als abgeschlossen, dann betätigt man die Return(CR)-Taste, worauf der Editor eine neue Zeile numeriert („2.00 ="). Soll zur übersichtlichen Gestaltung des späteren Programmausdrucks eine Freizeile zwischen den Text eingefügt werden, so kann dies wie in Zeile 2, 5, 20 und 25 durch Betätigen der Return-Taste geschehen. In diesen Zeilen wird demnach nichts eingespeichert.

Gedanklich stellt man sich nun die vier Blöcke einer Befehlszeile vor und setzt zwischen jedes Feld ein Semikolonzeichen: Label; Operator; Operand; Kommentar. Wird ein Feld nicht benutzt, so läßt man es leer und setzt nur die Trennungszeichen, in diesem Fall den Semikolon. In Zeile 6 wird dem Assembler der Programmtitel mitgeteilt. Dazu benötigt man keinen Label, deshalb wird gleich ein Semikolon gesetzt. Dann folgt der Operator TTL und darauf als Operand der Text. Fügt man die Trennungszeichen nicht ordnungsgemäß ein, kann der Editor später keine Blocktrennung vornehmen. In Zeile 7 soll mit dem Operator „ORG" dem Assembler mitgeteilt werden, welche Startadresse das Programm haben soll. Der Operand (Startadresse) soll eine Sedezimalzahl sein, was durch das Dollarzeichen festgelegt wird. Danach folgt der Kommentar. Zeile 8 beinhaltet einige organisatorische Assembler-Abläufe, die durch den Operator „OPT" angekündigt werden:

SYM Drucke alle verwendeten Symbole (Labels) nach dem Assemblieren aus.

GEN Drucke alle zu übersetzenden Zeichen, die durch die Operation FCB, FDB und FCC bestimmt werden, aus.

LSI Drucke nach dem Assemblieren die komplette Liste aus.

PAG Sollte mehr als eine Seite Text nötig sein, gib jeder folgenden eine Seitenzahl und schreibe den Titel dazu.

MEM Lade das übersetzte Programm in den Speicherbereich, der mit dem Operator ORG definiert wird.

Die Label-Definierung erfolgt in den Zeilen 9...11. Der Label-Begriff BUF 1 ist später durch die Sedezimaladresse ($) A0A0 zu ersetzen, BUF 2 durch A0A2 und BUF 3 durch A0A4. Die Label-Definierung kann auch am Programmende erfolgen.

Der erste Befehl, der vom Assembler in den Maschinen-Code zu übersetzen ist, steht in Zeile 12: Springe zum Programmbeginn. Da die Sprungweite und somit die endgültige Adresse nicht bekannt ist, wird im Operanden-Feld ein Label gesetzt, dessen Bezeichnung der eigenen Phantasie überlassen ist. Hier wird der Ausdruck „PRGBGN" benutzt. In Zeile 21 erscheint dieser Begriff im Label-Block, dorthin soll also von Zeile 12 aus gesprungen werden. Der Operand in den Zeilen 15 und 19 definiert eine „indexed" Adresse (,X) für den Akkumulator A ohne Offset (0). Der wichtigste Operator befindet sich in Zeile 26. Er teilt dem Assembler das Ende des Quellprogramms mit.

Um aus dem Zeilenmodus des Editors herauszukommen, gibt man bei der nächsten Zeileneröffnung das Zeichen # ein. Nun können Editor-Befehle eingegeben werden. Der Befehl SET TAB =';' bedeutet, daß der Semikolon mit der Betätigung der Tabulatortaste gleichzusetzen ist. Der Befehl TAB kennzeichnet die einzelnen Tabulatorpositionen. Im Beispiel soll nach dem Erscheinen des ersten Semikolons zum neunten Anschlag weitergesprungen werden; bei den nächsten

```
LAS
LISTING OR TAPE? L
                ** TEST PROGRAMM 1 **

                ** UNTERPROGAMM ZUM TRANSPORT EINES DEFINIERTEN **
                ** SOFTWARE-ABSCNITTS IN EINEM ANDEREN SPEICHER **

                        TTL   PROGRAMMTRANSPORT
A100                    ORG   $A100      STARTADRESSE
                        OPT   SYM,GEN,LIS,PAG;MEM
A0A0            BUF1    EQU   $A0A0
A0A2            BUF2    EQU   $A0A2
A0A4            BUF3    EQU   $A0A4
A100 7E A1 12           JMP   PRGBGN    ZUM PROGRAMMSTART
A103 08         TRANS2  INX             X+1= X
A104 FF A0 A0           STX   BUF1      SICHERE X IN BUF 1
A107 A6 00              LDAA  0,X       WERT AUS TABELLE 1 IN X
A109 FE A0 A4           LDX   BUF3      MOM.TABELLENSTAND 2 IN X
A10C 08                 INX             X+1= X
A10D FF A0 A4           STX   BUF3      NEUER TABELLENSTAND IN BUF 3
A110 A7 00              STAA  0,X       WERT AUS A IN TABELLE 2

A112 FE A0 A0   PRGBGN  LDX   BUF1      MOM.TABELLENSTAND 1 IN X
A115 BC A0 A2           CPX   BUF2      TABELLE 1 ZU ENDE?
A118 26 E9              BNE   TRANS2    SPRINGE WENN NICHT
A11A 39                 RTS             RUECKSPRUNG ZUM HAUPTPROGRAMM

                        END
      NO ERROR(S) DETECTED


   SYMBOL TABLE:
BUF1    A0A0        BUF2    A0A2        BUF3    A0A4        PRGBGN  A112
TRANS2  A103

LAS
LISTING OR TAPE? T
S113A1007EA11208FFA0A0A600FEA0A408FFA0A4A0
S10EA110A700FEA0A0BCA0A226E93915B9
```

Bild 4. Assembler-Ausdruck des lauffähigen Testprogramms

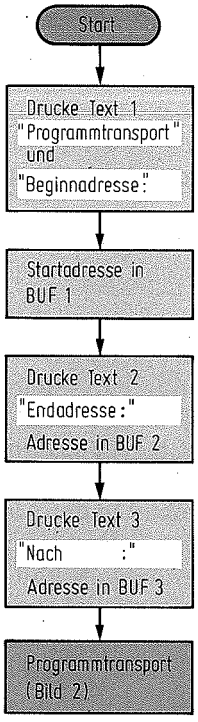


Bild 5. Erweitertes Flußdiagramm

▼ Bild 6. Anwendungsbeispiele der Editor-Befehle: „Setze Zeilenzähler auf 1", „Tausche einen Begriff", „Drucke eine bestimmte Zeile aus und füge eine Zeile ein"

```
T
£F/PRGBGN/
   12.00=          JMP     PRGBGN      ZUM PROGRAMMSTART
£12 C /JMP/BRA/
   12.00=          BRA     PRGBGN      ZUM PROGRAMMSTART
11.2P
   11.20=BADDR     EQU     E0D9
£I
   11.21=


^P!
    1.00=**  TEST PROGRAMM 1  **
    2.00=
    3.00=** UNTERPROGAMM ZUM TRANSPORT EINES DEFINIERTEN **
    4.00=** SOFTWARE-ABSCNITTS IN EINEM ANDEREN SPEICHER **
    5.00=
    6.00=           TTL     PROGRAMMTRANSPORT
    7.00=           ORG     $A100       STARTADRESSE
    8.00=           OPT     SYM,GEN,LIS,PAG;MEM
    9.00=BUF1       EQU     $A0A0
   10.00=BUF2       EQU     $A0A2
   11.00=BUF3       EQU     $A0A4
   12.00=PDATA1     EQU     E130
   13.00=BADDR      EQU     E0D9
   14.00=
   15.00=           LDX     TEXT1       "PROGRAMMTRANSPORT USW."
   16.00=           JSR     PDATA1      UP MINIBUG 3
   17.00=           JSR     BADDR       WARTE AUF ADRESSE
   18.00=           STX     BUF1        SICHERE ADRESSE IN BUF 1
   19.00=
   20.00=           LDX     TEXT2       "CR,LF,BEGINNADR"
   21.00=           JSR     PDATA1      UP MINIBUG 3
   22.00=           JSR     BADDR       WARTE AUF ADRESSE
   23.00=           STX     BUF2        SICHERE ADRESSE IN BUF 2
   24.00=
   25.00=           LDX     TEXT3       "CR,LF,NACH"
   26.00=           JSR     PDATA1      UP MINIBUG 3
   27.00=           JSR     BADDR       WARTE AUF ADRESSE
   27.10=           STX     BUF3        SICHERE ADRESSE IN BUF 3
   27.20=
   28.00=           BRA     PRGBGN      ZUM PROGRAMMSTART
   29.00=TRANS2     INX                 X+1= X
   30.00=           STX     BUF1        SICHERE X IN BUF 1
   31.00=           LDAA    0,X         WERT AUS TABELLE 1 IN X
   32.00=           LDX     BUF3        MOM.TABELLENSTAND 2 IN X
   33.00=           INX                 X+1= X
   34.00=           STX     BUF3        NEUER TABELLENSTAND IN BUF 3
   35.00=           STAA    0,X         WERT AUS A IN TABELLE 2
   36.00=
   37.00=PRGBGN     LDX     BUF1        MOM.TABELLENSTAND 1 IN X
   38.00=           CPX     BUF2        TABELLE 1 ZU ENDE?
   39.00=           BNE     TRANS2      SPRINGE WENN NICHT
   40.00=           RTS                 RUECKSPRUNG ZUM HAUPTPROGRAMM
   40.10=
   40.20=TEXT1      FCC     '* PROGRAMMTRANSPORT *'
   40.30=           FCB     $0D         RETURN
   40.40=           FCB     $0A         LF
   41.00=
   41.10=           FCC     'BEGINNADR.: '
   41.20=           FCB     $04         TEXT ENDE
   41.21=TEXT2      FCB     0D
   41.22=           FCB     $0A
   41.23=           FCC     'ENDADRESSE: '
   42.00=           FCB     $04         TEXT ENDE
   43.00=TEXT3      FCB     $0D
   44.00=           FCB     $0A
   45.00=           FCC     'NACH    : '
   46.00=           FCB     $04         TEXT ENDE
   47.00=
   48.00=           END
```

Bild 7. Editor-Ausdruck des verbesserten Programms mit drei Fehlern

zum 16. und 26. Anschlag. Die in der Programmliste am Zeilenanfang stehenden Pfundzeichen erscheinen, wenn der Editor auf ein neues Kommando wartet.

Nun soll das ganze Programm entsprechend der Tabulatorbefehle geordnet werden. Dies geschieht mit dem Befehl EXP (expandiere). Der Anfang eines Programms wird durch einen Pfeil nach oben ausgedrückt. Ausrufzeichen oder Pfeil nach unten signalisieren das Programmende.

Damit das vom Editor neu geordnete Programm überprüft werden kann, läßt man es ausdrucken (P = print). ^P! befiehlt, das Quellprogramm von Anfang bis Ende auszudrucken. In der neuen Form ist das Programm assemblergerecht. Mit dem Kommando LAS *(Bild 4)* ruft man den Assembler: Er meldet sich mit der Frage „Listing or Tape?". Gibt man daraufhin das Zeichen L ein, übersetzt der Assembler das Quellprogramm und gibt das Ergebnis in einem Ausdruck bekannt. Dieser Ausdruck besteht zum größten Teil aus der Editor-Eingabe, mit dem Unterschied, daß die Zeilennumerierung entfällt und stattdessen die in den Maschinen-Code übersetzten Mnemonik-Befehle mit den dazugehörigen Adressen stehen. Mit dem Ausdruck „No Errors detected" bestätigt der Assembler ein lauffähiges Programm. Der letzte Ausdruck stellt eine alphabetisch geordnete Label-Tabelle dar mit den dazugehörigen reellen Werten (Adressen).

Auf die Assembler-Frage „Listing or Tape?" kann man auch mit einem T antworten, worauf das übersetzte Programm auf Tonbandkassette überspielt wird. Nach Abschluß der Assembler-Arbeit meldet sich der Editor mit dem Pfundzeichen.

Im Beispiel traten keine Fehler auf – der Idealfall. Um die Arbeitsweise des Editors deutlicher zu machen, wird das Quellprogramm erweitert *(Bild 5)*. Das Quellprogramm liegt noch vor, da der Assembler keinerlei Veränderung darin vornimmt. Die Transportroutine soll nun so ausgebaut werden, daß es in der Lage ist, den Benutzer zu fragen, welcher Software-Abschnitt wohin transportiert werden soll. Es sind dazu Routinen aus dem Monitor-Programm MINIBUG-3 notwendig.

3.4 *Programmumarbeitung*

Im weiteren werden einige Befehle erklärt, ohne daß dabei auf den genauen Arbeitsablauf bei der Quellprogrammerweiterung geachtet wird.

Bild 6 zeigt, wie ein Begriff bzw. eine bestimmte Zeile gesucht wird. Da der Suchbefehl F *(Found)* nur in einer Richtung, von der Zeile ausgehend, die der Editor angewählt hat, operieren kann, setzt man den Zeilenzähler mit dem Befehl T *(Top)* auf 1. Durch die Anweisung F/PRGBGN/aktiviert, sucht der Editor eine Zeile, in der diese Zeichenkombination vorkommt, und druckt sie aus. Der Befehl 12 C/JMP/BRA/ ordnet an, in Zeile 12 den Zeichensatz JMP zu löschen und stattdessen BRA einzusetzen. Zur Kontrolle wird die geänderte Zeile automatisch ausgedruckt. 11.2P veranlaßt, daß die Zeile 11.2 ausgedruckt wird. Mit dem Befehl I *(Insert)* wird in das Quellprogramm eine neue Zeilenfolge eingefügt. Der Editor eröffnet nun

```
 A0A0            BUF1    EQU    $A0A0
 A0A2            BUF2    EQU    $A0A2
 A0A4            BUF3    EQU    $A0A4
 0000            PDATA1  EQU    E130
**   UNDEFINED SYMBOL
 0000            BADDR   EQU    E0D9
**   UNDEFINED SYMBOL

 A100 FE A1 3E           LDX    TEXT1   "PROGRAMMTRANSPORT USW."
 A103 BD 00 00           JSR    PDATA1  UP MINIBUG 3


 A155 42                 FCC    'BEGINNADR.:
 A156 45 47
 A158 49 4E
 A15A 4E 41
 A15C 44 52
 A15E 2E 3A
 A160 20
 A161 04                 FCB    $04       TEXT ENDE
 A162 00         TEXT2   FCB    0D
**   ILLEGAL CHARACTER FOR SPECIFIED BASE
 A163 0A                 FCB    $0A
```

Bild 8. Assembler-Hinweise auf die entdeckten Fehler

eine Zeile mit der Benennung 11.21. Man kann nun insgesamt 9 Zeilen einfügen.

Sind alle Möglichkeiten der Zeilenerweiterung ausgeschöpft, muß man alle Zeilen neu beziffern. Dies geschieht mit der Befehlsfolge T, REN (Renumber) und ^P!. Die Programmauflistung ist notwendig, um zu sehen, welche Zeilenbestimmung ausgegeben wurde.

Bild 7 gibt eine vorläufige Version wieder, die der Assembler bearbeiten soll. Die beiden Begriffe PDATA1 und BADDR im Label-Feld verweisen auf Unterprogramme im Monitor. BADDR: Nimm über die ASCII-Tastatur vier Sedezimalzeichen auf, bilde daraus eine 16-bit-Adresse und lade sie in das Indexregister. PDATA1: Drucke einen ASCII-Text aus. Die Startadresse steht im Indexregister, Stopp wird durch den Wert 04 am Textende markiert.

In der Zeile 41.10 soll der Textteil „Beginnadr.:" in Form einer ASCII-Zeichen-Auflistung in das Programm übernomen werden. Der Sedezimalwert 04 in Zeile 41.20 schließt den Textteil ab. Die Konstanten 0D und 0A sind Druckeranweisungen: 0D = Wagenrücklauf (CR), 0A = Zeilenvorschub (LF).

An drei Stellen entdeckte der Assembler eine fehlerhafte Operanden-Eingabe *(Bild 8)*; es fehlten die Dollarzeichen (Sedezimaldefinition). Mit dem Editor müssen diese Zeichen nachträglich eingefügt werden. Die Entwicklungszeit eines Programms dieser Größenordnung beträgt 2...4 Stunden, einschließlich der Flußdiagrammerstellung.

Literatur

[1] Beschreibung des TSC-Assember/Editors. Ing. W. Hofacker GmbH, 8000 München 75, Postfach 754 37.

[2] *Tireford H.*: Die Adressierungsarten bei Mikroprozessoren. ELEKTRONIK Sonderheft 2, Software, 1978, S. 54...58.

Dr. Jürgen Rathlev

Als sich dem Verfasser dieses Beitrags vor einigen Jahren die Aufgabe stellte, größere Mikroprozessorprogramme zu erstellen [3], stand kein entsprechendes Entwicklungssystem zur Verfügung. Er entschloß sich deshalb, auf dem vorhandenen, gut ausgebauten Minicomputersystem (NOVA mit Plattenlaufwerk und Digital-Magnetbandgeräten) die geeignete Cross-Software (Assembler und Simulator) selbst zu entwickeln. Das gewählte Konzept läßt sich auf beliebige 8-bit-Mikroprozessoren anwenden.

Ein Universal-Cross-Assembler für 8-bit-Mikroprozessoren

Hilfsprogramme für die Textbearbeitung sind bereits in der Betriebssoftware des Minicomputers enthalten [2]. Ebenso kann auch bei speziellen Benutzerprogrammen auf die im Betriebssystem enthaltenen Treiberroutinen für die Peripherie (z. B. Plattenlaufwerk) zurückgegriffen werden. Der Aufwand für die Entwicklung der Cross-Software, die durchweg im Assembler geschrieben wurde, bleibt damit durchaus in vertretbaren Grenzen. Gleichzeitig erhält man die Möglichkeit einer komfortablen Programmbearbeitung über Plattenfiles.

Obwohl ursprünglich nur für den Mikroprozessor 8080 vorgesehen, wurde bei der Konzipierung des Assemblers von vornherein versucht, eine Beschränkung der Benutzung nur für diesen einen Typ zu vermeiden. Das Konzept, das sich weitgehend an das Prinzip des für den Minicomputer vom Hersteller beigefügten Assemblers hält [2], hat sich dann später sehr bewährt, als auch Programme für andere Mikroprozessoren mit dem Cross-Assembler bearbeitet werden sollten.

1 Struktur des Universal-Cross-Assemblers

Wie die meisten Assembler arbeitet auch der beschriebene das Quellprogramm in zwei Durchgängen ab (*Two Pass Assembler*). Im ersten Durchgang wird die Zuweisung zwischen Adreßsymbolen (z. B. Labels) und Adressen vorgenommen, im zweiten Durchgang folgt dann die eigentliche Übersetzung des Programms.

Bereits im ersten Durchgang wird geprüft, wieviele Bytes von den jeweiligen Befehlen im Quellprogramm belegt werden. Damit kann durch einfaches Abzählen für jeden Befehl die zugehörige Speicheradresse bestimmt werden. Vor dem zweiten Durchgang ist also bereits die Anordnung des Programms im Speicher des Rechners, auf dem das Programm später ausgeführt werden soll, festgelegt.

Im zweiten Durchgang wird der spätere Inhalt der Speicherzellen aus den symbolischen Befehlen des Quellprogramms abgeleitet. Da es sich hier um einen Cross-Assembler handelt, kann das so erzeugte Bitmuster natürlich nicht sofort in den Speicher des Mikrocomputers geschrieben werden. Es muß vielmehr auf einem Datenträger zwischengespeichert werden (z. B. Platte oder Lochstreifen). Um diese Datenmenge klein zu halten und gegen Übertragungsfehler abzusichern, wird das Bitmuster in Blöcken formatiert binär auf den Datenträger geschrieben (*Object Code*) und mit Prüfsummen versehen (siehe Abschn. 5).

Wie man sieht, muß offensichtlich zwischen verschiedenen Arten von Symbolen unterschieden werden. Genannt wurden bereits die symbolischen Adressen (*User Symbols*) und die den Rechnerbefehlen zugeordneten Symbole (Operationen oder *Permanent Symbols*). Hinzu kommt noch eine dritte Gruppe, die man als Steuerbefehle für den Assembler benötigt (*Pseudo Operation*). Diese Symbole erzeugen kein Bitmuster im Objekt-Code. Sie beeinflussen lediglich den Funktionsablauf des Assemblers.

Das Format des Quellprogramms ist zeilenorientiert, d. h. jede Zeile des Programms entspricht genau einem Mikroprozessorbefehl. Innerhalb einer Zeile arbeitet der Assembler formatfrei. Den verschiedenen Symbolen (Label, Operation, etc.) ist also kein fester Platz in der Zeile zugeordnet. Die Trennung voneinander erfolgt durch Leer- oder Tabulationszeichen. Trotzdem ist die Einhaltung einer bestimmten Reihenfolge erforderlich:

1. Am Anfang einer Zeile können eine oder mehrere symbolische Adressen stehen, die durch einen nachfolgenden Doppelpunkt als solche gekennzeichnet werden. Ihnen wird im ersten Durchgang vom Assembler der Wert zugewiesen, den die Adresse hat, unter der später der nachfolgende Befehl im Speicher wiederzufinden ist. Natürlich darf ein Label in einer Zeile auch ganz fehlen, da man diese im allgemeinen nur bei Ansprungstellen im Programm einsetzt.
2. Es folgt das Symbol für die Operation (Mikroprozessorbefehl oder Pseudo Op.). Bei den meisten Operationen ist zusätzlich die Angabe eines oder mehrerer Operanden erforderlich (z. B. Angabe von Registern oder Adressen), wodurch die Operation zu einem vollständigen Befehl ergänzt wird (Bitmuster).
3. Die Operanden dürfen beliebig komplizierte arithmetische Ausdrücke sein, die sich zusammensetzen aus Zahlen, Symbolen (sie stehen für den ihnen im ersten Durchgang zugewiesenen Wert), Klammern

und Rechenoperationen (z. B. +, –, usw. oder AND, OR usw.). Mehrere Operanden werden voneinander durch Kommas getrennt.

4. Der Rest der Zeile darf nach einem Semikolon einen beliebigen Text (Kommentar) enthalten. Diese Erläuterungen sind für die Dokumentation von Gedankengängen beim Programmieren unerläßlich. Es ist auch zulässig, daß in einer Zeile nur Kommentar steht oder daß eine Zeile ganz leer ist. Von diesen Möglichkeiten sollte man beim Programmieren der Übersichtlichkeit halber häufigen Gebrauch machen.

2 Symboltabellen

Unter einem Symbol soll, wie bereits erwähnt, eine Aneinanderreihung von alphanumerischen Zeichen verstanden werden, die vom Programmierer meist in sinnfälliger Form gewählt werden. Der Universal-Cross-Assembler zieht zur Unterscheidung die ersten fünf Zeichen heran, das erste muß grundsätzlich ein Buchstabe sein.

Eine Abspeicherung dieser Symbole könnte grundsätzlich im ASCII-Format (8 bit pro Zeichen) erfolgen. Da aber nur 36 Zeichen (Buchstaben und Ziffern) unterschieden werden, würde dies wegen der enthaltenen Redundanz Speicherplatz verschenken. Der Universal-Cross-Assembler ordnet deshalb jedem Symbol einen „Radix-50"-Wert zu (16 + 11 bit pro Symbol). Die Zuordnung für die einzelnen Zeichen ist in *Tabelle 1* zu sehen. Jeweils drei Zeichen lassen sich in 16 bit unterbringen:

$s_1 = (c_1 \cdot 40 + c_2) \cdot 40 + c_3$

$s_2 = c_4 \cdot 40 + c_5$

c_j sind die Werte für die einzelnen Zeichen aus Tabelle 1. Die in s_2 freien Bits können bei zukünftigen Erweiterungen zur Unterscheidung verschiedener Befehlstypen genutzt werden.

Um die Zeiten für die Übersetzung eines Quellprogramms kurz zu halten, müssen die Wertzuweisungen aller Symbole schnell greifbar sein. Man ordnet sie deshalb in Tabellenform an. Die Ordnung in dieser Tabelle sollte so vorgenommen werden, daß die Suche nach einem bestimmten Symbol möglichst wenig Zeit erfordert. Gleiches gilt für das Einsetzen eines neuen, noch nicht definierten Symbols im ersten Durchgang.

Tabelle 1. Zuordnung der einzelnen Zeichen

Zeichen	ASCII (oktal)	Radix-50 (oktal)
NULL	0	0
0	60	1
.	.	.
.	.	.
.	.	.
9	71	12
A	101	13
.	.	.
.	.	.
.	.	.
Z	132	44
@	100	45
?	77	46

Ordnet man die Symbole in der Reihenfolge ihres Auftretens im Quellprogramm, ist nur die zweite Bedingung optimal erfüllt. Zum Wiederfinden eines Symbols kann nur ein lineares Suchverfahren, das sehr zeitaufwendig ist, verwendet werden.

Besser sind die beiden folgenden Verfahren:

1. Alphabetische Ordnung:
Bei einer alphabetischen Sortierung nimmt das Anlegen einer Symboltabelle relativ viel Zeit in Anspruch, da für jedes neue Symbol Platz geschaffen werden muß, indem der Rest der Tabelle nach hinten geschoben wird. Da jedes Symbol aber nur einmal im ersten Durchgang definiert wird, ist dies durchaus vertretbar. Die Suche nach einem Symbol im zweiten Durchgang, die ja sehr viel öfter notwendig sein kann, läßt sich dagegen ausgesprochen schnell durchführen, wenn man dabei nach dem Verfahren der Intervallschachtelung arbeitet. Die Anordnung der Symbole in alphabetischer Reihenfolge in der Tabelle ist nämlich gleichbedeutend mit einer Monotonie der zugeordneten Radix-50-Werte. Man hat also lediglich zu prüfen, ob der Radix-50-Wert des zu suchenden Symbols in die erste oder zweite Hälfte der Tabelle gehört, und wiederholt dieses Halbierungsverfahren mit der zutreffenden Hälfte, bis nur noch ein Symbol übrigbleibt. Durch diese Strategie findet man in einer Tabelle mit N Symbolen in i = ld (N) Schritten das gewünschte. Ist z. B. N = 512, so genügen 9 Iterationsschritte (bei der linearen Suche sind im Mittel N/2 = 256 Schritte auszuführen). Ein weiterer Vorteil dieses Verfahrens ist, daß ohne Umsortieren eine solche Symboltabelle als Inhaltsverzeichnis für die Wertzuweisungen der Benutzersymbole ausgedruckt werden kann.

2. Hash-Algorithmus:
Dieses Verfahren geht von dem Prinzip aus, die große Tabelle in m kleine Tabellen zu unterteilen, die dann wegen ihrer Kürze linear durchsucht werden können. Die Entscheidung, in welche dieser Untertabellen das Symbol abgespeichert werden soll, wird durch einen Algorithmus getroffen, der aus dem Radix-50-Wert des Symbols einen Hash-Wert k von 0...m-1, z. B. durch Division dieses Wertes durch m und Benutzung des Divisionsrestes, ableitet. Ein neues Symbol wird dann in den ersten freien Platz der so angewählten Tabelle abgespeichert. Sollte sie bereits gefüllt sein, kann man entweder einen zweiten Satz von Untertabellen anlegen oder den ersten freien Platz der Untertabelle k+1 verwenden. Der Algorithmus zur Bestimmung des Hash-Wertes sollte in jedem Fall gewährleisten, daß alle Untertabellen möglichst gleichmäßig gefüllt werden.

Um die Suche innerhalb der Tabellen schnell durchführen zu können, sollten diese außerdem kurz sein, d. h. je mehr Untertabellen vorgesehen werden, desto kürzer werden die Suchzeiten.

Die Leistungsfähigkeit dieses Verfahrens hängt auch sehr wesentlich vom Füllgrad der Tabelle ab [1]. Man sollte eine solche Tabelle von vornherein so groß anlegen, daß sie im ungünstigsten Fall nur zu etwa zwei Drittel gefüllt ist. Dabei bleiben die Suchzeiten

deutlich unter denen des Intervallschachtelungsverfahrens. Eine alphabetische Sortierung für ein Inhaltsverzeichnis muß allerdings zusätzlich vorgenommen werden.

Im Universal-Cross-Assembler wurde wegen der einfacheren Programmierung das erstgenannte Verfahren gewählt. Dabei wurde auch berücksichtigt, daß Mikroprozessorprogramme verhältnismäßig kurz sind. Für ein Programm, das im Speicher etwa 4 K belegt, werden für das Assemblieren etwa 40 s benötigt. Die Zeit teilt sich auf beide Durchgänge etwa gleich auf.

3 Pseudo-Operationen

Hier kann je nach Funktion zwischen verschiedenen Typen unterschieden werden: Die erste Gruppe übt ausschließlich Steuerfunktionen beim Ablauf des Assemblers aus (n als Operand steht für einen arithmetischen Ausdruck, wie er in Abschn. 1 beschrieben wurde).

- RDX n: *(radix)* Hiermit wird die Basis n festgelegt, mit der alle nachfolgenden Zahlen zu lesen sind (z. B. oktal, dezimal usw.).
- PRDX n: *(print radix)* Diese Pseudo-Op. legt fest, mit welcher Basis Adresse und zugeordneter Speicherinhalt bei Programmlisten *(listings)* gedruckt werden sollen.
- WORD n: Es wird zwischen zwei Betriebsarten gewählt, mit denen 16-bit-Werte in zwei Bytes aufgespalten werden (zuerst niedriges Byte, dann hohes Byte wie beim 8080, oder umgekehrt wie beim 6800).
- END: Hierdurch wird das Ende des Programms gekennzeichnet.
- EOF: *(end of file)* Es wird darauf hingewiesen, daß das Quellprogramm in einem anderen Plattenfile fortgesetzt wird.
- RSTSY: *(reset symbol table)* Alle Symboltabellen werden gelöscht, mit Ausnahme der Pseudo-Op.-Tabelle.

Die zweite Gruppe von Pseudo-Operationen erlaubt Änderungen am Adreßzähler und das Setzen von Speicherplätzen auf Festwerte:

- ORG n: *(origin)* Der Adreßzähler wird auf den Wert n gesetzt (z. B. am Programmanfang).
- DS n: *(define storage)* Es werden n freie Speicherplätze an dieser Stelle reserviert.
- DB n_1,...: *(define bytes)* Die nachfolgenden Speicherplätze werden auf die 8-bit-Werte n_1, n_2, ... gesetzt.
- DW n_1,...: *(define word)* Die nachfolgenden Speicherplätze werden auf die 16-bit-Werte n_1, n_2,... in der Byte-Reihenfolge gesetzt, wie sie durch die letzte vorangegangene WORD-Pseudo-Op. festgelegt wurde.

Die letzte Gruppe von Pseudo-Operationen dient schließlich dazu, Symbole (insbesondere auch den Befehlssatz des Mikroprozessors) zu definieren:

- EQU symbol = n: Dem Symbol wird der Wert n zugewiesen. So definierte Werte können nicht mehr verändert werden.
- SET symbol = n: Dem Symbol wird ein Wert zugewiesen, der nachträglich verändert werden darf.
- Dyxxx symb. = n: *(Tabelle 2)* Mit dieser Pseudo-Op. werden die permanenten Symbole (Befehlssatz) definiert. y gibt die Anzahl der Bytes an, die der Befehl später im Speicher belegt, xxx den Typ des Befehls.

Die zuletzt genannten Pseudo-Operationen sind der Grund für die universelle Verwendbarkeit des Cross-Assemblers, da durch sie der mikroprozessorspezifische Befehlssatz definiert wird. Für jeden Typ von Befehl muß eine besondere Pseudo-Op. dieser Gruppe vorgesehen werden. Der Assembler erkennt an einer internen Tabelle, wieviele und welche Operanden den so definierten Operationen im Quellprogramm folgen müssen. So wird z. B. der 8080-Befehl MOV durch die Pseudo-Op. D1TRG (Tabelle 2) zugewiesen (D1TRG MOV = 100[oktal]). Die noch fehlenden Angaben über die Register werden dann erst beim Assemblieren des Quellprogramms durch die dort angegebenen Operanden festgelegt, die jeweils nur drei Bit beanspruchen dürfen (Wert des Operanden kleiner als 8).

4 Anpassen an verschiedene Mikroprozessoren

Da alle permanenten Symbole nur durch Pseudo-Operationen definiert werden können, läßt sich der Universal-Cross-Assembler ohne Softwareände-

Tabelle 2. Definition der permanenten Symbole (Befehlssatz)

Name	Bedeutung	Argumente	Bytes	Bitzuordnung 7 6 5 4 3 2 1 0
D1 WOF	define 1 byte instruction without operand field	–	1	
D1 SRG	define 1 byte instruction with source register	S (≦7)	1	S
D1 DRG	define 1 byte instruction with destination register	D (≦7)	1	D
D1 TRG	define 1 byte instruction with two register	D, S (≦7)	1	D S
D1 SRP	define 1 byte instruction with single reg. pair	RP (≦7)	1	RP/2
D1 SPR	define 1 byte instruction with special register	R (≦7)	1	R/4
D1 REG	define 1 byte instruction with 4 bit register	RG (≦15)	1	RG
D2 DAT	define 2 byte instruction with data byte	DAT ($<2^8$)	2	DAT
D2 RDT	define 2 byte instruction with destination register and data byte	D, DAT	2	D / DAT
D2 REL	define 2 byte instruction with relative address	ADD ($-2^7 \leqq$ ADD-LOC $<2^7$)	2	ADD-LOC
D3 ADD	define 3 byte instruction with address word	ADD ($<2^{16}$)	3	ADD low (high) / ADD high (low)
D3 RAD	define 3 byte instruction with dest. reg. pair and address word	RP, ADD	3	RP/2 / ADD low (high) / ADD high (low)

rungen für verschiedene Mikroprozessoren verwenden. Der jeweilige Befehlssatz wird dem Assembler erst nachträglich durch eine Initialisierungstabelle (*Tabelle 3*) mitgeteilt. So wäre es z. B. leicht möglich, sich eine persönliche Assemblersprache für einen Mikroprozessor, abweichend von den Herstellerangaben, zu schaffen.

Die Initialisierungstabellen werden wie normale Quellprogramme behandelt. Man bricht das Assemblieren aber bereits nach dem ersten Durchgang ab, wenn alle Symboltabellen aufgebaut sind, und kopiert sich den Cross-Assembler mit Tabellen auf einen externen Datenspeicher (Platte). Das Betriebssystem des verwendeten Minicomputers kennt hierfür einen speziellen Befehl. Die so entstandene Cross-Assembler-Version kann später beim Assemblieren anderer Quellprogramme als Assembler verwendet werden, der alle in der Initialisierungstabelle vereinbarten Symbole (also auch den speziellen Befehlssatz) kennt.

Tabelle 3. Zuweisung des Befehlssatzes

```
;INITIALIZE THE INTEL 8080 INSTRUCTIONS
D1WOF   A=      7       D1TRG   MOV=     100
D1WOF   B=      0       D1SRP   DAD=     011
D1WOF   C=      1       D1SRP   INX=     003
D1WOF   D=      2       D1SRP   DCX=     013
D1WOF   E=      3       D1SRP   PUSH=    305
D1WOF   H=      4       D1SRP   POP=     301
D1WOF   L=      5
D1WOF   M=      6       D1SPR   STAX=    002
D1WOF   SP=     6       D1SPR   LDAX=    012
D1WOF   PSW=    6
                        D2DAT   IN=      333
D1WOF   XCHG=   353     D2DAT   OUT=     323
D1WOF   XTHL=   343     D2DAT   ADI=     306
D1WOF   SPHL=   371     D2DAT   ACI=     316
D1WOF   PCHL=   351     D2DAT   SUI=     326
D1WOF   RLC=    007     D2DAT   SBI=     336
D1WOF   RRC=    017     D2DAT   ANI=     346
D1WOF   RAL=    027     D2DAT   XRI=     356
D1WOF   RAR=    037     D2DAT   ORI=     366
D1WOF   NOP=    000     D2DAT   CPI=     376
D1WOF   HLT=    166
D1WOF   EI=     373     D2RDT   MVI=     006
D1WOF   DI=     363
D1WOF   CMC=    077     D3ADD   STA=     062
D1WOF   STC=    067     D3ADD   LDA=     072
D1WOF   CMA=    057     D3ADD   SHLD=    042
D1WOF   DAA=    047     D3ADD   LHLD=    052
D1WOF   RET=    311     D3ADD   CALL=    315
D1WOF   RC=     330     D3ADD   CC=      334
D1WOF   RNC=    320     D3ADD   CNC=     324
D1WOF   RZ=     310     D3ADD   CZ=      314
D1WOF   RNZ=    300     D3ADD   CNZ=     304
D1WOF   RP=     360     D3ADD   CP=      364
D1WOF   RM=     370     D3ADD   CM=      374
D1WOF   RPE=    350     D3ADD   CPE=     354
D1WOF   RPO=    340     D3ADD   CPO=     344
                        D3ADD   JMP=     303
D1SRG   ADD=    200     D3ADD   JC=      332
D1SRG   ADC=    210     D3ADD   JNC=     322
D1SRG   SUB=    220     D3ADD   JZ=      312
D1SRG   SBB=    230     D3ADD   JNZ=     302
D1SRG   ANA=    240     D3ADD   JP=      362
D1SRG   XRA=    250     D3ADD   JM=      372
D1SRG   ORA=    260     D3ADD   JPE=     352
D1SRG   CMP=    270     D3ADD   JPO=     342

D1DRG   INR=    004     D3RAD   LXI=     001
D1DRG   DCR=    005
D1DRG   RST=    307     END
```

Auf diese Weise entstanden die Versionen für die Mikroprozessoren 8080, 6800, 6502 und COSMAC. Bei den letzteren waren dabei leichte Änderungen gegenüber den Original-Assemblersprachen notwendig, da diese ein festes Format innerhalb einer Zeile des Quellprogramms vorsehen. Für den Benutzer ist dies aber nach kurzer Gewöhnung ohne Bedeutung.

Ebenso besteht jederzeit die Möglichkeit, den Universal-Cross-Assembler auch für andere Mikroprozessoren einzusetzen. Es wäre sogar eine Erweiterung auf 16 bit Wortlänge durchführbar. Dafür müßten eventell neue Pseudo-Operationen geschaffen werden, die Operationstypen definieren, die im bisherigen Satz noch nicht enthalten sind. Durch die gewählte Struktur bleibt die dazu erforderliche Arbeit aber mit minimalem Aufwand verbunden.

5 Organisation der Ein- und Ausgabe

Der Universal-Cross-Assembler läuft unter der Kontrolle eines Plattenbetriebssystems (NOVA-RDOS). Die Struktur der Ein- und Ausgabe sowie der Betrieb durch den Benutzer wurden weitgehend angelehnt an die im genannten Betriebssystem üblichen Vereinbarungen. Der Cross-Assembler liegt ebenso wie die Quellprogramme als File auf der Platte vor.

Der Aufruf des Assemblers erfolgt in einer Befehlszeile (*Command Line*), in der zuerst der Name des Cross-Assemblers (z. B. ASM80), dann die der Quellprogramme angegeben werden. Des weiteren können in dieser Befehlszeile Steuerungen (*Switches*) für die Ein- und Ausgabe gesetzt werden.

Ohne Angabe zusätzlicher Steuerungen werden ein oder mehrere in der Befehlszeile genannte Quellprogramme, die jeweils entweder durch die Pseudo-Operation EOF als Teilprogramme oder durch END als letztes Programm gekennzeichnet sind, nacheinander in zwei Durchgängen assembliert. Der Objekt-Code wird in einen Plattenfile geschrieben, der den Namen des ersten in der Befehlszeile angegebenen Quellprogramms mit angehängtem „.BN" (für binär) erhält.

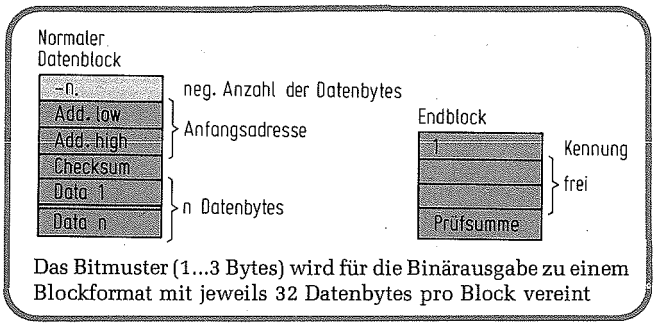
Sollen daneben andere Ausgaben erzeugt werden, z. B. eine Programmliste, die neben dem Quelltext in jeder Zeile den Wert der zugeordneten Adresse sowie deren Inhalt als Zahl enthält, kann dies in der Befehlszeile durch Setzen der entsprechenden Steuerung veranlaßt werden.

Eine Besonderheit stellt die Steuerung mit dem Switch „S" dar. Hier wird das Assemblieren nach dem ersten Durchgang abgebrochen und, wie bereits in Abschn. 4 erwähnt, der Speicherinhalt, d. h. der Cross-Assembler mit den momentanen Symboltabellen, in einen Plattenfile geschrieben, der später wie die Urversion des Assemblers aber mit neuen Symbolen verwendet werden kann.

6 Programmablauf beim Assemblieren

Der Universal-Cross-Assembler ist weitgehend modular aufgebaut, um zukünftige Änderungen für andere Mikroprozessoren leicht einfügen zu können. Die Basis bildet eine Reihe von Unterroutinen für verschiedene Zwecke:

– Suchen eines Symbols in einer Tabelle.



Das Bitmuster (1...3 Bytes) wird für die Binärausgabe zu einem Blockformat mit jeweils 32 Datenbytes pro Block vereint

- Einfügen eines neuen Symbols in eine Tabelle.
- Auswerten eines arithmetischen Ausdrucks mit Zahlen, Symbolen, Klammern und verschiedenen Rechenoperatoren.
- Erzeugen des Binärformats für die Ausgabe des Objekt-Codes.
- Erzeugen einer Zeile für die Programmliste.
- Erzeugen einer Fehlermeldung auf der Konsole bei Programmfehlern.
- Umcodieren von ASCII- auf Radix-50-Format und zurück.

Daneben existieren natürlich noch zahlreiche kleinere Unterroutinen für die verschiedensten Zwecke.

Der Programmablauf beim Assemblieren eines Quellprogramms geschieht nach folgendem Muster:

1. Analysieren der Befehlszeile (Abschn. 5) sowie Anlegen und Öffnen der erforderlichen Plattenfiles.
2. Einlesen einer Zeile des Quellprogramms in einen Textpuffer.
3. Nacheinander werden die verschiedenen Symbole in einer Zeile analysiert:
   a) Dem Symbol folgt ein Doppelpunkt, es handelt sich also um einen Label. Im Durchgang 1 wird die Benutzersymboltabelle abgesucht, ob der neue Label bereits in ihr enthalten ist. Wenn ja, erfolgt eine Fehlermeldung, andernfalls wird die Tabelle mit dem neuen Symbol ergänzt. Ihm wird der Wert der Speicheradresse zugeordnet, in der der Befehl dieser Zeile später steht.
   b) Es handelt sich um eine Pseudo-Operation. Sie wird wie in Abschn. 3 beschrieben ausgeführt. War es EOF, wird der momentane Quellfile geschlossen und der in der Befehlszeile als nächster angegebene geöffnet. Bei END wird im ersten Durchgang der zweite vorbereitet, indem wieder der erste in der Befehlszeile genannte Quellfile geöffnet wird. Im zweiten Durchgang wird nach END die Binärausgabe durch einen Endblock abgeschlossen und die Programmliste durch eine alphabetische Tabelle der Benutzersymbole ergänzt.
   c) Es handelt sich um ein permanentes Symbol (Operation). Aus der diesem Symbol in der Tabelle zugeordneten Kennung wird abgeleitet, wieviele Bytes dieser Befehl belegt und welche bzw. wieviele Operanden in der Zelle des Quellprogramms folgen müssen. Im ersten Durchgang wird nur die erste Information für die Zuordnung der Speicheradressen benötigt. Die Operanden werden nur im zweiten Durchgang ausgewertet. Der Wert der Operanden wird aus dem angegebenen arithmetischen Ausdruck berechnet und mit dem festen Bitmuster dieser Operation zu einem kompletten Mikroprozessorbefehl kombiniert.

Eventuelle Fehler im Quellprogramm werden erkannt und dem Benutzer auf der Konsole mitgeteilt. Das in c) erzeugte Bitmuster (1...3 Bytes) wird für die Binärausgabe zu einem Blockformat mit jeweils 32 Datenbytes pro Block vereint *(Bild)*. Für die Programmliste werden 16-bit-Adresse und 8-bit-Inhalt als Oktal-, Dezimal- oder Sedezimalzahlen und die zugehörige Zeile des Quellprogramms in den vereinbarten Plattenfile oder auf dem Zeilendrucker geschrieben.

4. Bei Punkt 2 fortfahren.
5. Beenden des Assemblierens nach Auffinden der END-Pseudo-Operation (siehe 3b).

7 Praktische Erfahrungen

Inzwischen hat sich der beschriebene Weg der Softwareerstellung für Mikroprozessoren gut bewährt. Es existieren zur Zeit Cross-Assembler-Versionen für die Mikroprozessoren 8080, 6800, 6502 und COSMAC, mit denen mehrere Benutzer ihre Assemblerprogramme übersetzen. Die Programmbearbeitung wird dabei durch das komfortable Betriebssystem des Minicomputers sehr erleichtert. Das Austesten der Programme erfolgt entweder mit einfachen Mikroprozessor-Entwicklungssystemen, die die einschlägigen Möglichkeiten zur Fehlersuche bieten, oder mit Simulatorprogrammen auf dem Minicomputer. Hier existieren zur Zeit zwei Versionen für den 8080 und den 6800. Mit ihnen wird die Fehlersuche durch Programmhilfen, wie Breakpoints, Einzelschrittbetrieb und Protokoll des Programmablaufs, wesentlich erleichtert. Standardperipherie wie Konsole (TTY) sowie Lochstreifenleser und -stanzer stehen am Simulator ebenfalls zur Verfügung, wobei Leser und Stanzer durch Plattenfiles simuliert werden. Außerdem wurden kleine Hilfsprogramme entwickelt, die aus dem vom Universal-Cross-Assembler erzeugten Binärcode die verschiedenen, von den Herstellern der Mikroprozessoren vorgeschlagenen speziellen Sedezimal-Lochstreifenformate erzeugen. Solche Lochstreifen können dann auf den Mikroprozessor-Entwicklungssystemen geladen werden.

Literatur

[1] *Barron, D. W.*: Assembler und Lader. Carl Hanser Verlag, München 1970.
[2] RDOS – Software-Beschreibung. Data General Corp.
[3] *Rathlev, J.*: Der Mikroprozessor in der Datenerfassung. ELEKTRONIK 1978, H. 8, S. 53...58.
[4] *Schmitt, G.*: Aufgabe und Arbeitsweise eines Assemblers. ELEKTRONIK 1978, H. 4, S. 85...88.
[5] *Brinkmann, K.-D., Kreißel, F.*: Ein universell einsetzbarer Assembler. ELEKTRONIK 1978, H. 11, S. 81...84.
[6] *Speiser, A.*: Digitale Rechenanlagen. Springer Verlag, Berlin-Heidelberg.

Dr. rer. nat. Jürgen Rathlev wurde in Schleswig-Holstein geboren. Er studierte an der Universität Kiel Physik und ist dort seit 1974 als wissenschaftlicher Angestellter am Institut für Angewandte Physik tätig. Das Hauptarbeitsgebiet liegt bei der Entwickung von neuen Meßmethoden und Geräten zur Erfassung verschiedener physikalischer und chemischer Eigenschaften des Meerwassers sowie von Verfahren zur automatischen Erfassung und Auswertung solcher Meßdaten. Der Einsatz dieser Meßgeräte erfolgt auf Expeditionen mit verschiedenen Forschungsschiffen.
Diensttelefon: (04 31) 8 80-39 54

Übersicht der Distributoren, die Mikroprozessor-Produkte und verwandte Systeme im Programm führen

Distributor	Distributor-Verträge mit		Art der Produkte	Eigene µP-Abteilung		Software (in Zusammenarbeit mit Software-Firmen)
	µP-Herstellern	Geräte/ System-Herstellern		Sitz	Zuständig	
Aktiv-Electronic GmbH, Ballinstr. 12-14, 1000 Berlin 47, Tel. (0 30) 6 84 50 88	AMI	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme, Floppys, Datensichtgeräte	–	–	Ja
Altron KG; Germaniastr. 10, 3160 Lehrte Tel. (0 51 32) 5 30 24	General Instrument MOS	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Lehrte	Ja H. Menze	Nein
Astronic GmbH, Winzererstr. 47d, 8000 München 40, Tel. (0 89) 30 40 11	Synertek	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	München	Ja L. Glas	Ja
Atlantik Elektronik GmbH, Hofmannstr. 20, 8000 München 70, Tel. (0 89) 7 85 31 68	Standard Microsystems Corp. (SMC) (Exkl.-Vertrag)	–	Peripherie-Bausteine, µP-Platinen aus Eigenentwicklung	München	Ja G. Böning	Nein
Gustav Beck KG, Postfach 15 02 80, 8500 Nürnberg 15 Tel. (09 11) 3 49 61*	AMI, RCA, SGS-ATES	–	µPs und Speicher, Entwicklungssysteme	Nürnberg	Ja A. Pfeiffer	Nein
Berger-Elektronik GmbH, Am Tiergarten 14, 6000 Frankfurt, Tel. (06 11) 49 03 11	Fairchild, General Instrument	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Frankfurt	Ja H. Schulte	Nein
Bitronic GmbH, Einsteinstr. 127, 8000 München 80, Tel. (0 89) 4 70 20 98	Synertek	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	München	Ja T. Oberbichler	Eigene Software-Entwicklung
Celdis GmbH, Henschelring 5, 8011 Kirchheim, Tel. (0 89) 9 03 20 57	Texas Instruments	Digital Equipment (für Norddeutschland)	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	München	Ja W. D. Straub	Nein
Cosmos Energie GmbH, Hegelstr. 16, 8000 München 83, Tel. (0 89) 60 20 88	AMD	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	–	–	Ja
Dahms-Elektronik GmbH, Bürgermstr.-Neff-Str. 19, 6806 Viernheim, Tel. (0 62 04) 30 33	–	NAS Elektronik	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Viernheim	Ja D. Strauß	Nein
DEMA Electronic GmbH, Blütenstr. 21, 8000 München 40, Tel. (0 89) 28 80 18/19	Mostek (für Speicher auch mit EMM-SEMI)	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme, Minicomputer, Peripherieger.	München	Ja H. Nassauer	Ja
DITTHA Elektronik Distributor, Südfeldstr. 7, 3000 Hannover 91, Tel. (05 11) 45 86-1	Valvo/ Signetics	–	µP-, Speicher-, Interface-Bauelemente	–	–	Nein
Dr. G. Dohrenberg, Bayreuther Str. 3, 1000 Berlin 30, Tel. (0 30) 2 13 80 45	Fairchild, Mostek, Ferranti	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	Berlin	Ja R. Nachtweh	Ja
Ing.-Büro Karl-Heinz Dreyer Flensburger Str. 3, 2380 Flensburg, Tel. (0 46 21) 2 31 21	National Semiconductor (NSC), SGS-ATES	Matsushita Electric (Kleindrucker)	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme, Bildschirme, Drucker, Floppy-Disks	Hamburg Bojeweg 54	Ja D. Wellach	Nein
EBV Elektronik GmbH, Gabriel-Max-Str. 72, 8000 München 90 Tel. (0 89) 64 40 55*	Motorola, AMD, National Semiconductor	PEP	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Stuttgart	Ja H. Kizler	Ja
Siegfried Ecker, Postfach 33 44, 6120 Michelstadt Tel. (0 60 61) 22 33	Mostek, SGS-ATES, SSS	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme		Nein	Ja
Elbatex GmbH, Cäcilienstr. 24, 7100 Heilbronn, Tel. (0 71 31) 8 90 01	AMD NEC Toshiba	–	µP-, Speicher-, Interface-Bauelemente	–	–	Nein
Elcowa GmbH, Postfach 12 94 09, 6200 Wiesbaden, Tel. (0 61 21) 6 50 05	Fairchild NEC	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Wiesbaden	Ja E. Gros	Nein

Distributor	Distributor-Verträge mit µP Herstellern	Geräte/System-Herstellern	Art der Produkte	Eigene µP-Abteilung Sitz	Zuständig	Software (in Zusammenarbeit mit Software-Firmen)
elecdis Ruggaber GmbH, Hertichstr. 41, 7250 Leonberg, Tel. (0 71 52) 4 70 81	Valvo/ Signetics, SGS-ATES	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	Freiburg-Lehen	K. Leyk	Ja
Electronic 2000 Vertriebs GmbH, Neumarkter Str. 75, 8000 München 80 Tel. (0 89) 43 40 61	Intel, Fairchild, General Instrument, SGS-ATES, Synertek	Digital Equipment (Mini-computer), E-H International	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme, Minicomputer, Peripheriegeräte, PROM-Programmierger.	München	H. Sprogies	Ja
Elkose GmbH, Postfach 9, 7141 Schwieberdingen Tel. (0 71 50) 14-1	Texas Instruments, Valvo/Signetics, RCA	–	µP-, Speicher-, Interface-Bauelemente, Entwicklungssysteme	im Aufbau		Ja
Hilmar Frehsdorf KG Carl-Zeiss-Str. 3, 2085 Quickborn Tel. (0 41 06) 7 10 58/59	keine		µP-, Speicher-, Interface-Bausteine	Quickborn	H. Wiencke	Nein
Halbleiter Spezialvertrieb Carroll + Co. GmbH, Burnitzstr. 34, 6000 Frankfurt Tel. (06 11) 63 80 41	Keine Distributor-Verträge (Vertreibt Produkte der Firmen SWTPC, Motorola, DEC, Mostek, General Instr.)		µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Peripheriegeräte	Frankfurt	P. Bestian	Ja
IBH Ingenieurbüro Gutenbergring, 2000 Hamburg-Norderstedt 3, Tel. (0 40) 5 23 40 06	Fairchild		µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	Norderstedt	–	Nein
Jermyn GmbH, Schulstr. 36, 6277 Camberg-Würges, Tel. (0 64 34) 60 05 (23-1)	Intel, Motorola, Texas Instruments	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Drucker, Terminals, Programmiergeräte	Camberg	H. Maser U. Pein J. Teepe	gelegentlich
Walter Kluxen Nordkanalstr. 52, 2000 Hamburg 1, Tel. (0 40) 24 89-1	Texas Instruments, Valvo/Signetics	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	im Aufbau		Nein
Kontron Elektronik GmbH, Breslauer Str. 2, 8051 Eching, Tel. (0 89) 3 19 01-1	Zilog, Harris	Zahlreiche Vertretungen	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Terminals, Drucker, Floppy Disks, Programmiergeräte	Eching		Ja
Mania GmbH Hauptstr. 86, 6384 Schmitten 2, Tel. (0 60 82) 29 72/3	Intel	Eigene Systeme	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Peripheriegeräte	Schmitten	K. H. Kohn	Eigene Software-Abteilung
Maristron Electronic GmbH, Jebenstr. 1, 1000 Berlin 12, Tel. (0 30) 3 12 12 03	keine		Mikroprozessor-Bausteine von NSC, Motorola, Intel	–	–	Nein
Matronic GmbH, Lichtenberger Weg 3, 7400 Tübingen, Tel. (0 70 71) 2 43 31	Mostek	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Drucker, Programmierg.	Tübingen	C. Caroli	Ja
MEV-Mikro Elektronik Vertrieb GmbH, Münchner Str. 16a, 8021 Deining, Tel. (0 81 70) 72 89	AMD	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	im Aufbau		–
Microscan GmbH, Überseering 31, 2000 Hamburg 60, Tel. (0 40) 6 30 50 67	NEC, AMI	Münzer u. Diehl, Centronics	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Programmiergeräte	Nein		Eigene Software-Entwicklung
Mikrotec GmbH, Johannesstr. 91, 7000 Stuttgart 1 Tel. (07 11) 22 80 27	AMI	Conrac	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Terminals, Drucker Floppy Disks, Plattenspeicher	Stuttgart	R. Fürst	Eigene Software-Erstellung
Mütron, Müller + Co. KG, Postfach 10 30 67, 2800 Bremen 1 Tel. (04 21) 31 04 85	Motorola, Valvo/Signetics	PEP	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Drucker	Bremen	R. Warnatsch	Nein
Neumüller GmbH, Eschenstr. 2, 8021 Taufkirchen, Tel. (0 89) 61 18-1	Commodore/Mos-Technology, TI, Brutech Electronics	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Terminals, Drucker, Floppy-Disks usw.	Taufkirchen	H. Zoschke	Nein

Distributor	Distributor-Verträge mit µP-Herstellern	Geräte/System-Herstellern	Art der Produkte	Eigene µP-Abteilung Sitz	Zuständig	Software (in Zusammenarbeit mit Software-Firmen)
Alfred Neye-Enatechnik GmbH, Schillerstr. 14, 2085 Quickborn, Tel. (0 41 06) 6 12-1	Harris, Intel, Mostek, Motorola, RCA, Valvo/ Signetics	Data General, Perkin Elmer, Tally, Princeton Electronic Products	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Terminals, Drucker, Programmiergeräte	in Quickborn, Düsseldorf, Stuttgart, Darmstadt, München	Ja T. Heydolph	Ja
PANEL Electronic Vertr.-GmbH, Hermann-Oberth-Str. 7, 8011 Putzbrunn, Tel. (0 89) 46 40 24-26	NSC (National Semiconductor)	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, Peripheriegeräte	Putzbrunn	Ja H. Lutz	Ja
Positron Vertriebs GmbH, Benzstr. 1, 7016 Gerlingen, Tel. (0 71 56) 2 30 51	NSC, Valvo/Signetics	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, CRTs, Drucker, Floppy-Disks, TTY	Gerlingen	Ja D.Staegemann	Ja
Raffel u. Co. Electronics GmbH, Lochner Str. 1, 4030 Ratingen, Tel. (0 21 02) 2 80 24/26	Mostek	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer	Ratingen	Ja H. Schicks	Nein
Retron GmbH, Rodeweg 18, 3400 Göttingen Tel. (05 51) 9 20 07	Texas Instruments, Valvo/Signetics	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	Göttingen	Ja H. J. Gellert	Ja
RTG E. Springorum GmbH & Co. Bronner Str. 7, 4600 Dortmund 1, Tel. (02 31) 54 95-1	Motorola, NSC, RCA, Valvo/ Signetics, Thomson-CSF	PEP	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Drucker, Terminals, Floppy-Disks	Dortmund	Ja S. Kaiser	Ja
Sasco GmbH, Hermann-Oberth-Str. 16, 8011 Putzbrunn, Tel. (0 89) 46 11-1	Motorola, RCA, NSC, Valvo/ Signetics	PEP	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, Peripheriegeräte	Putzbrunn	Ja H. Baumann	Nein
Dietrich Schuricht, Richtweg 30, 2800 Bremen 1, Tel. (04 21) 32 14 44	keine		µP-, Speicher-, Interface-Bausteine der Firmen Siemens, Valvo, TI	Bremen	Ja H. Köpp	Nein
SE Spezial Electronic KG, Hermann-Lingg-Str. 16, 8000 München 2, Tel. (0 89) 53 03 87	Intersil, Fairchild, Western Digital	Pro-Log, Millenium	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, Peripherieger., µP-Analysierer	München Bückeburg	Ja H. Heutink F. Dorfner J. Schwerte	Nein
Spoerle Electronic, Otto-Hahn-Str. 13, 6072 Dreieich, Tel. (0 61 03) 3 04-1	Motorola, RCA, Valvo/Signetics, Thomson-CSF, Texas Instr.	Facit, Elbit, Digitronics, Centronics	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, Terminals, Drucker, Stanzer/Leser	Dreieich	Ja R. Zörb	Ja
System Kontakt GmbH, Siemensstr. 5, 7107 Bad Friedrichshall, Tel. (0 71 36) 50 31	Rockwell	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer	Bad Friedrichshall	Ja M. Strobl	Ja
Technoprojekt, Heinr.-Ebner-Str. 13, 7000 Stuttgart, Tel. (07 11) 56 17 12	Fairchild, Motorola, Toshiba (Speicher)	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	Neckarhausen	Ja E. Walz	Ja
Fred Trommelschläger, Kirchbüchel 1, 5330 Königswinter 1, Tel. (0 22 23) 2 13 68	keine		µP-, Speicher-, Interface-Bausteine, Minicomputer, Sichtgeräte, Drucker	Nein –	– –	Ja
Ultratronic GmbH, Münchner Str. 6, 8031 Seefeld Tel. (0 81 52) 77 74	Valvo/Signetics, NEC, AMI,	–	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme	Seefeld	Ja P. Rath	Eigene Software-Abteilung
Unitronic GmbH, Münsterstr. 338, 4000 Düsseldorf, Tel. (02 11) 63 42 14	Fairchild, Mostek	Videoton, Stoppani	µP-, Speicher-, Interface-Bausteine, Entwicklungssysteme, Minicomputer, Terminals, Drucker, Floppy Disks	Bad Segeberg	Ja B. Kurbjuhn	Ja

Weitere Firmen, die noch in diesem Jahr oder nächstes Jahr µP-Produkte in ihr Distributionsprogramm aufnehmen wollen:
Elkobra GmbH (8041 Haimhausen), Manager Electronic (7800 Freiburg/7000 Stuttgart 81), Paul Opitz & Co. (2000 Hamburg 26), Setron Schiffer-Elektronik GmbH & Co. KG (3300 Braunschweig), J. W. Zander GmbH + Co. (7800 Freiburg).

Weitere Zweigstellen (Verkaufsbüros) von in dieser Tabelle genannten Distributoren:

1) Beck KG: a) Neu-Isenburg, Tel. (0 61 02) 80 20; b) 8000 München 90, Tel. (0 89) 66 34 17
2) Berger-Elektronik GmbH: a) 8000 München 80, Tel. (0 89) 4 48 45 00; b) 7000 Stuttgart-Degerloch, Tel. (07 11) 76 90 95

3) DITTHA Elektronik Distributor: a) 4600 Dortmund, Tel. (02 31) 1 78 87; b) 6500 Mainz, Tel. (0 61 31) 6 92-1; c) 7000 Stuttgart, Tel. (07 11) 64 20 93/94; d) 8000 München 45, Tel. (0 89) 3 51 60 31

4) EBV Elektronik: a) 6000 Frankfurt 1, Tel. (06 11) 72 04 16-7; b) 4000 Düsseldorf, Tel. (02 11) 8 48 46-7; c) 7000 Stuttgart 80, Tel. (07 11) 24 74 83; d) 3006 Burgwedel/Hannover, Tel. (0 51 39) 45 70

5) elecdis Ruggaber GmbH: 7800 Freiburg-Lehen, Tel. (07 61) 8 31 43

6) Electronic 2000: a) 7257 Ditzingen, Tel. (0 71 56) 70 83; b) 8500 Nürnberg, Tel. (09 11) 3 67 39; c) 4000 Düsseldorf, Tel. (02 11) 76 71 41

7) Elkose GmbH: a) 4600 Dortmund, Tel. (02 31) 81 82 84; b) 1000 Berlin, Tel. (0 30) 2 61 15 86

8) Kontron Elektronik GmbH: a) 4000 Düsseldorf 1, Tel. (02 11) 72 30 17; b) 6000 Frankfurt, Tel. (06 11) 63 60 61; c) 2000 Hamburg 70, Tel. (0 40) 68 23 21; d) 7000 Stuttgart 30, Tel. (07 11) 81 46 21; e) 1000 Berlin 41, Tel. (0 30) 7 92 30 31-3; f) 8500 Nürnberg 16, Tel. (09 11) 53 33 06

9) Mania GmbH: a) 4800 Bielefeld 1, Tel. (05 21) 2 51 41; b) 8950 Kaufbeuren, Tel. (0 83 41) 1 60 46

10) Microscan GmbH: 8045 Ismaning, Tel. (0 89) 9 61 63

11) Alfred Neye-Enatechnik: a) 1000 Berlin, Tel. (0 30) 4 33 30 52; b) 3000 Hannover, Tel. (05 11) 88 60 86; c) 4000 Düsseldorf, Tel. (02 11) 66 61 45; d) 6100 Darmstadt, Tel. (0 61 51) 2 64 46; e) 7000 Stuttgart, Tel. (07 11) 73 63 57; f) 8000 München, Tel. (0 89) 47 30 23

12) RTG Springorum: a) 1000 Berlin 10, Tel. (0 30) 3 42 10 41; b) 2000 Hamburg 76, Tel. (0 40) 29 29 66; c) 6200 Wiesbaden, Tel. (0 61 21) 52 73 09; d) 7000 Stuttgart, Tel. (07 11) 76 64 28; e) 8000 München 40, Tel. (0 89) 36 65 00

13) Sasco GmbH: a) 3000 Hannover 27, Tel. (05 11) 86 25 86; b) 4005 Düsseldorf/Meerbusch 3, Tel. (0 21 50) 14 33; c) 7000 Stuttgart 1, Tel. (07 11) 24 45 21-23; d) 8500 Nürnberg, Tel. (09 11) 20 41 52

14) Dietrich Schuricht: a) 3000 Hannover, Tel. (05 11) 31 10 44; b) 5000 Köln, Tel. (02 21) 72 04 51; c) 7000 Stuttgart, Tel. (07 11) 64 10 01

15) Spezial Electronic KG: a) 3062 Bückeburg/Hannover, Tel. (0 57 22) 10 11-5

16) Spoerle Electronic: a) 5000 Köln 1, Tel. (02 21) 23 50 98; b) 8000 München 1, Tel. (0 89) 22 74 17

17) Unitronic GmbH: 2360 Bad Segeberg, Tel. (0 45 51) 20 64

Übersicht der Halbleiterfirmen, die Mikroprozessor-Produkte herstellen und ihre offiziellen Distributoren

Hersteller	Distributoren
AMD	Cosmos Energie GmbH, EBV-Elektronik, Elbatex GmbH, Nordelektronik Vertriebs GmbH (Quickborn)
AMI	Aktiv-Electronic GmbH, Beck KG, Microscan GmbH, Mikrotec GmbH, Ultratronic GmbH
Fairchild	Berger Elektronik, Dr. Dohrenberg, Elcowa, Electronic 2000, IBH Ingenieurbüro, Technoprojekt, Unitronic GmbH
Ferranti	Altron KG, Astronic GmbH, Dr. Dohrenberg, Elbatex GmbH, Mansfeld GmbH + Co. KG (Frankfurt), Nordelektronik KG (Quickborn), Weisbauer Elektronik GmbH (Dortmund)
General Instrument	Altron KG, Berger Elektronik, Electronic 2000, Roederstein-Bauelemente Vertriebs GmbH (Berlin)
Harris Semiconductor	Kontron Elektronik GmbH, Alfred Neye-Enatechnik GmbH, Jermyn GmbH
Intel	Electronic 2000, Jermyn GmbH, Mania GmbH, Alfred Neye-Enatechnik GmbH
Intermetall	Beck KG, Elkose GmbH, Henskes GmbH + Co. KG (Hannover), Walter Kluxen, Plastronic GmbH (Berlin), Spoerle Electronic
Intersil	Spezial Electronic KG
MOS Technology	Neumüller GmbH
Mostek	Dema Electronic GmbH, Dr. Dohrenberg, Siegfried Ecker, Matronic GmbH, Alfred Neye-Enatechnik GmbH, Raffel & Co. Electronics GmbH, Unitronic GmbH
Motorola	EBV Elektronik, Jermyn GmbH, Alfred Neye-Enatechnik GmbH, Mütron Müller & Co. KG, Spoerle Electronic, RTG E. Springorum, Sasco GmbH, Technoprojekt GmbH
National Semiconductor	Bodamer GmbH (München), Ing.-Büro Dreyer, EBV Elektronik, Mütron Müller & Co. KG, PANEL Elektronik, Positron GmbH, Sasco GmbH, RTG E. Springorum KG
NEC	Elbatex GmbH, Elcowa GmbH, Microscan GmbH, Ultratronic GmbH
RCA	Beck KG, Elkose GmbH, Alfred Neye-Enatechnik GmbH, RTG E. Springorum KG, Sasco GmbH, Spoerle Electronic
Rockwell	System-Kontakt
SGS-ATES	Beck KG, Ing.-Büro Dreyer, Siegfried Ecker, elecdis Ruggaber GmbH, Electronic 2000, Getronik (Hamburg), MBS-Elektronik GmbH (München), Setron Schiffer-Elektronik GmbH & Co. KG (Braunschweig), Weißbauer Elektronik GmbH (Dortmund)
Synertek	Astronic GmbH (München), Bitronic GmbH, Electronic 2000
Texas Instruments	Celdis GmbH, Elkose GmbH, Jermyn GmbH, Walter Kluxen, Metronik GmbH (Unterhaching/München), Neumüller GmbH, Retron GmbH, Schiffers-Elektronik (Aachen), Spoerle Electronic, TISCO (München)
Thomson-CSF	RTG E. Springorum KG, Spoerle Electronic
Toshiba	Elbatex GmbH, Roederstein (Landshut), Technoprojekt GmbH
Valvo/ Signetics	DITTHA Elektronik Distributor, EBV Elektronik, elecdis Ruggaber GmbH, Elkose GmbH, Walter Kluxen, Mütron Müller & Co. KG, Alfred Neye-Enatechnik GmbH, Retron GmbH, RTG E. Springorum KG, Sasco GmbH, Spoerle Electronic, Ultratronic GmbH
Western Digital	Spezial Electronic
Zilog	Kontron Elektronik GmbH

Elektronik

...die Sicherheit besser informiert zu sein

Die Elektronik revolutioniert beinahe jede Branche und jedes Arbeitsgebiet. Daraus ergibt sich ein gewaltiger Informationsfluß, der nur mit viel Sachkenntnis und Umsicht bewältigt werden kann. Diese Auswahl der relevanten Informationen bietet Ihnen die Fachzeitschrift ELEKTRONIK, damit Sie immer zum richtigen Zeitpunkt über alles Wesentliche informiert sind. Eigene Korrespondenten in den USA und Japan garantieren eine stets aktuelle Berichterstattung auch von diesen Märkten. Exklusive Copyright-Abkommen mit führenden amerikanischen Fachzeitschriften ermöglichen die Übernahme von Aufsätzen über neueste Technologien in deutscher Sprache.
Die ELEKTRONIK berichtet über: Meß-, Steuer- und Regelungstechnik, Analog- und Digitaltechnik sowie Datentechnik, mit Anwendungen in Forschung und Industrie, Maschinen- und Gerätebau, Verkehrswesen und Medizin. Industrielle Prozeßautomatisierung, Fertigungsmethoden für die Elektronik-Industrie. Bauelemente einschließlich neuer Technologien und Anwendungen, Schaltungstechnik und -praxis, Nachrichtentechnik, Optoelektronik. Mikroprozessor und Mikrocomputer werden in der ELEKTRONIK besonders intensiv betreut. Auch die Programmierung (software) wird eingehend behandelt. 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 Ingenieurwissens in Form der *Elektronik-Arbeitsblätter*, marktgerechte Neuheitenberichte im *Elektronik-Markt* und aktuelle Branchen-Nachrichten im *Elektronik-Express.*

Zwei *Lexikon-Karten* und zwei *Normen-Karten* pro Heft bereichern laufend den technischen Wortschatz bzw. ermöglichen den Aufbau einer Normenkartei. Der Rubrik *ELEKTRONIK-Notizen* kann man mit einem Blick das Neueste aus Forschung und Anwendung der Elektronik entnehmen. Eine umfassende internationale Literaturschau vermittelt in jedem Heft den unerläßlichen „Blick über den Zaun".

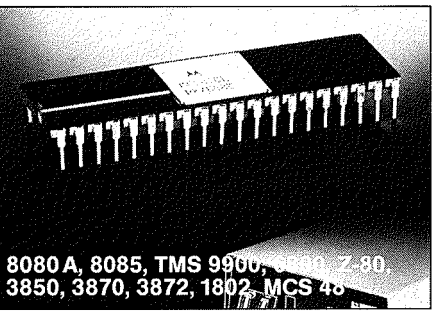
Die ELEKTRONIK erscheint 26mal im Jahr, alle 14 Tage am Donnerstag.
Das Einzelheft kostet DM 4.20, das Jahresabonnement DM 94.– im Inland, DM 104.– im Ausland.

Bitte bestellen Sie – mit nebenstehender Bestellkarte

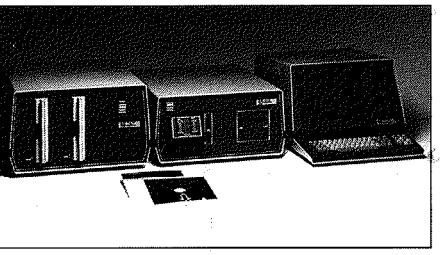
F/ZV/779/164/12'

18,—

Mikroprozessor-Entwicklungssystem



Das universelle TEKTRONIX-Entwicklungssystem MDA 8002A eignet sich für alle aufgeführten Prozessoren.



Mikroprozessor-Entwicklungssystem MDA 8002 A.

Workshop: Einführung in das Entwicklungssystem durch System-Spezialisten.

„MDA 8002 A" – ein universelles Mikroprozessor-Entwicklungssystem, das in allen Phasen, von der Programmentwicklung mit Dokumentation, Assemblierung oder Übersetzung von Hochsprachen und der Hardware/Softwareintegration, dazu beiträgt, Entwicklungszeit einzusparen. Das „MDA 8002 A" ist ein modular aufgebautes System, dessen „Mainframe" bis zu 20 steckbare Platinen aufnehmen kann. Im Grundsystem sind folgende Module enthalten:

- Systemprozessor (2650) mit Schnittstelle für das System-Terminal
- Debug-Modul mit Steuereinrichtung für die Konsole
- Systemspeicher (32 k-Byte)
- System-Communications-Modul
- Assembler Prozessor (Z-80)
- Diskettenstation mit Doppellaufwerk

Zur Ausrüstung als Entwicklungsplatz für einen speziellen Mikroprozessor werden folgende Optionen benötigt:

- Betriebssystem für einen ausgewählten Mikroprozessor aus o.g. Palette mit Editor, Assembler (Macro-Assembler), Timer, Communications-Software und Dienstprogrammen für PROM-Programmer, Debug, Real-Time Prototype Analyzer und Disassembler.
- Emulator Prozessor Modul für einen ausgewählten Mikroprozessor zur Ausführung der Anwenderprogramme. Dieses Modul ersetzt zusammen mit der Prototype Control Probe den Mikroprozessor der Anwenderschaltung.
- Prototype Control Probe für einen ausgewählten Mikroprozessor.

Für einen Entwicklungsplatz, der allen Ansprüchen gerecht wird, stehen folgende Ergänzungen zur Verfügung:

- Programmspeichererweiterung auf 64 k-Byte (als 16 k oder 32 k-Modul)
- PROM-Programmer (für 1702A, 2704/08)
- Real-Time Prototype Analyzer zur Verfolgung von Programmlauf in Echtzeit (Darstellung: Adressen, Daten, Steuerleitung und acht periphere Signale mit Acht-Kanal-Tastkopf).
- Systemterminal CT 8100 Video-Terminal CT 8101 Printing Terminal 4024/4025 Computer Terminal
- Line-Printer zur Programmdokumentation LP 8200

Das Mikroprozessor-Entwicklungs-LAB.
Ein vollständiger Entwicklungsplatz für ein Mikroprozessorlabor.
Neben den Möglichkeiten des „MDA 8002 A" werden hier zusätzliche Hilfsmittel und Laborausrüstungen eingesetzt. Logikanalyse, Darstellung zeitbezogener Signale mit dem Oszilloskop, sowie die Takterzeugung mit dem Funktionsgenerator seien hier nur als Beispiele genannt.
Unterlagen über Logikanalyse, Laborgeräte der TM 500er Serie oder das Mikroprozessor-Entwicklungssystem „MDA 8002 A" senden wir Ihnen gerne zu.

Ernst-Reuter-Platz 10 · 1000 Berlin-West 10
Ruf (030) 3414036
Große Bergstraße 213 · 2000 Hamburg 50
Ruf (040) 380191
Sedanstraße 13-17 · 5000 Köln 1
Ruf (0221) 7722-1
Kriegsstraße 39 · 7500 Karlsruhe 1
Ruf (0721) 27981
Plieninger Straße 150 · 7000 Stuttgart 80
Ruf (0711) 722039
Herzogstraße 61 · 6078 Neu-Isenburg
Ruf (06102) 3136
Tassiloplatz 7 · 8000 München 90
Ruf (089) 4162-1
Donaustraße 36 · 8500 Nürnberg
Ruf (0911) 64881

ROHDE & SCHWARZ
VERTRIEBS-GMBH