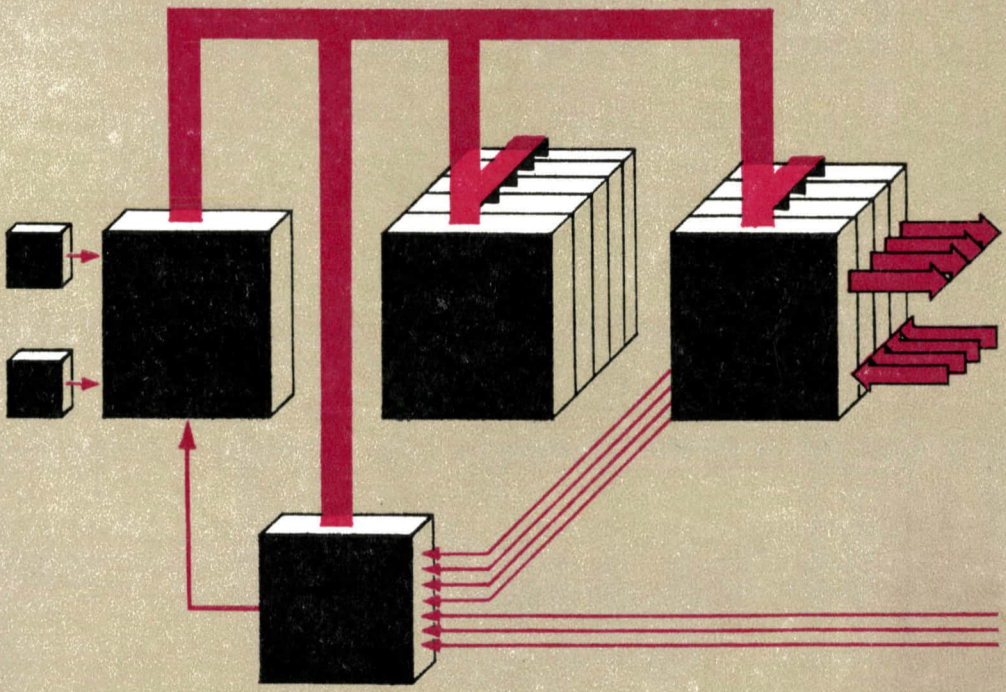
Einführung in die Mikro-rechentechnik



Franke

Einführung in die Mikrorechentechnik

Von Dr. sc. techn. Klaus Franke

2., durchgesehene Auflage

VEB VERLAG TECHNIK BERLIN

Franke, Klaus:
Einführung in die Mikrorechentechnik / von Klaus Franke. — 2., durchges. Aufl. — Berlin: Verlag Technik, 1986. — 136 S. : 64 Bilder, 13 Taf.

ISBN 3-341-00149-2

2., durchgesehene Auflage

Lizenz 201 · 370/191/86
Printed in the German Democratic Republic
Satz und Druck: Gutenberg Buchdruckerei und Verlagsanstalt Weimar, Betrieb der VOB Aufwärts
Buchbinderische Verarbeitung: Druck und Kulturwaren Leipzig
Lektor: Ing. Oswald Orlik
Schutzumschlag: Kurt Beckert
DK 681.32-181.4:62-523.8 (075.8)
LSV 3053 · VT 3/5707-2
Bestellnummer 553 333 7
01400

Vorwort

Die Mikroelektronik hat bewirkt, daß Digitalrechner (Computer) zu einem Massenprodukt geworden sind. Obwohl sich die Mikrorechner *funktionell* nicht von ihren „Vorfahren" – den digitalen Rechenautomaten der ersten Generation, den EDV-Anlagen, den Prozeß- und Kleinrechnern – unterscheiden, haben sie einen grundlegenden Wandel in nahezu allen technischen Gebieten ausgelöst.

Mikrorechner ermöglichen die Informationsverarbeitung direkt am Ort des Bedarfs. Dies führt zu einer *Dezentralisierung* des Rechnereinsatzes und zu einer *Integration* von Rechnern in Maschinen und Geräten der verschiedensten Art.

Mit diesem Wandel ergeben sich folgende Gesichtspunkte, die auch in der Ausbildung auf allen technischen Gebieten zu berücksichtigen sind: Bisher wurden Rechner bzw. EDV-Anlagen als komplette Endprodukte vom Hersteller bezogen. Neben der Hardware lieferte der Produzent zugleich ein umfangreiches Softwarepaket (Betriebssystem), das den Rechner für ein bestimmtes Anwendungsfeld profilierte. Vom Anwender wurde daher „nur" die Fähigkeit der Aufbereitung der Probleme bis zur Programmformulierung (meist in höheren Programmiersprachen) verlangt. Die Nutzung der durch die Mikrorechentechnik gebotenen Möglichkeiten erfordert dagegen vom Anwender ein tieferes Eindringen in die Technik der Rechner und in das Wechselspiel zwischen Hardware und Software. Die Rechner sind nicht mehr allein Hilfsmittel (Werkzeuge), sondern werden zum Arbeitsgegenstand selbst. Kenntnisse über das Funktionselement Rechner, über dessen Arbeitsweise, Struktur und Leistungsmerkmale müssen daher zum Bestandteil zumindest aller ingenieurtechnischen Disziplinen werden, wenn die potentiellen Nutzungsmöglichkeiten auch nur annähernd ausgeschöpft werden sollen.

Es wird auch in Zukunft keine „Anwendungskunde" für Mikrorechner geben. Nur die richtige Einschätzung der Fähigkeiten und Grenzen dieser technischen Systeme befähigt zum Erkennen geeigneter Einsatzfelder. Dieses Buch soll dazu einen Beitrag leisten. Es wendet sich an den Leserkreis, der mit dem Wesen der Mikrorechentechnik insofern vertraut sein muß, daß er in der Lage ist, eigenständig Anwendungsfälle zu erkennen und hinsichtlich ihrer Realisierbarkeit einzuschätzen. Es werden daher keine konkreten mikroelektronischen Schaltkreistypen beschrieben, sondern der Stand der Technik analysiert und die Grundprinzipien der Mikrorechner und ihrer Programmierung behandelt.

Es ist mir ein Bedürfnis, an dieser Stelle Prof. Dr.-Ing. habil. *M. Krauß* für den entscheidenden Anstoß, diese Aufgabe in Angriff zu nehmen, sowie Prof. Dr. sc. techn. *P. Fey* und Dr.-Ing. *H. Sterl* für Hinweise und Diskussionen während der Manuskripterstellung zu danken.

Mein Dank gilt weiterhin Frau Dipl.-Ing. *M. Rumpf* und Frau Dipl.-Ing. *S. Wendav* vom VEB Verlag Technik für die gute Zusammenarbeit sowie Herrn *G. Reimann* für die sorgfältige Bearbeitung der Zeichnungsvorlagen.

Klaus Franke

Inhaltsverzeichnis

1. Einleitung

Obwohl die Geschichte der elektronischen Rechentechnik erst bis in die Mitte dieses Jahrhunderts zurückreicht, hat sie aufgrund eines beispiellosen Entwicklungstempos eine derartige Fülle von Rechnertypen und entsprechenden Bezeichnungen hervorgebracht, als „jüngstes Kind" die Mikrorechner, daß es geraten erscheint, an den Anfang eine Übersicht über die bisherige Entwicklung der Digitalrechner und ihrer Anwendungsgebiete zu stellen (Bild 1.1).

In zeitlicher Hinsicht lassen sich bisher 4 Etappen unterscheiden, die allgemein mit der jeweils verfügbaren Bauelementebasis (Elektronenröhren, diskrete Transistoren, integrierte Schaltkreise, großintegrierte Schaltkreise) im Zusammenhang gesehen werden und für die in den ersten 3 Etappen auch der Begriff Rechnergeneration geprägt wurde.

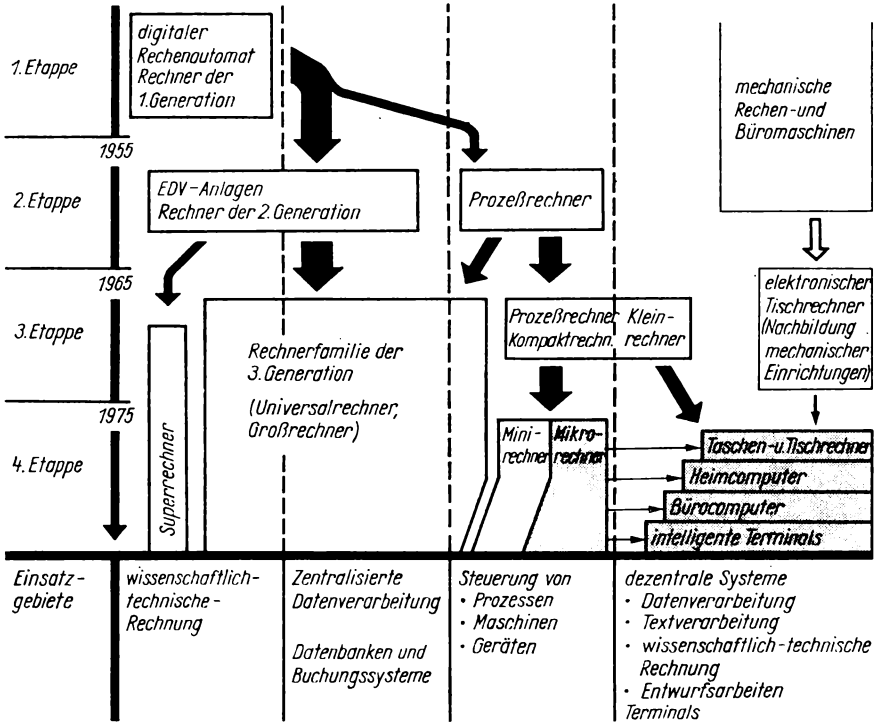


Bild 1.1. Entwicklungsetappen der elektronischen Rechentechnik und ihrer Einsatzgebiete

Die mit Elektronenröhren realisierten ersten Digitalrechner (1. Generation, Elektronenrechner) wurden ausschließlich für die Bewältigung umfangreicher wissenschaftlich-technischer Rechnungen projektiert. Diese Anlagen mit für heutige Maßstäbe riesigen Ausmaßen, großem Energieverbrauch, hohen Anschaffungs- und Betriebskosten wiesen eine Leistung auf, die gegenwärtig bereits von einem mittleren Taschen- oder Tischrechner erwartet wird. Mit ihnen wurde jedoch ein neuer Maschinentyp geboren, dessen prinzipielle Struktur und Arbeitsweise nach wie vor das Bild der Rechentechnik prägt. Aufgrund der Anwendung dieser Anlagen zur Zahlenverarbeitung wurde der Begriff **Rechner (Computer)** eingeführt, der ebenfalls überdauerte, obwohl sich der Schwerpunkt der Anwendungen längst auf andere Aufgaben verlagert hat.

Diese Veränderung des Einsatzgebietes erfolgte bereits in der 2. Etappe: Die Erfindung und industrielle Fertigung des Transistors ermöglichten den Bau von Rechenanlagen mit geringeren Abmessungen, niedrigeren Herstellungs- und Betriebskosten und damit den breiteren Einsatz der Rechner in Industrie und Wirtschaft (2. Generation). Aus der Synthese mit der mechanischen Lochkartentechnik entstanden Anlagen, die vorwiegend zur kommerziellen Datenverarbeitung eingesetzt wurden. Dementsprechend wurde der Begriff **elektronische Datenverarbeitungsanlage (EDVA)** zur am häufigsten verwendeten Bezeichnung für Digitalrechner. Im Unterschied zur wissenschaftlich-technischen Rechnung stehen bei der Datenverarbeitung die Eingabe und Ausgabe und die Archivierung (Speicherung) großer Zahlenmengen im Vordergrund, während die Verarbeitungsvorgänge (Rechenoperationen) im Verhältnis dazu in den Hintergrund treten.

In die 2. Etappe fielen aber auch die ersten Versuche, **Rechner als Steuereinrichtungen** für industrielle Prozesse einzusetzen (Prozeßrechner). Damit wurde ein Anwendungsfeld mit grundsätzlich anderem Anforderungsprofil erschlossen. Diese Rechner hatten und haben in erster Linie die Aufgabe, Meßgrößen des Prozesses zu „beobachten", um daraus Stellbefehle zu „berechnen", die entweder direkt den Prozeß steuern oder (wie für die 2. Etappe typisch war) den Bediener bei der Prozeßführung unterstützen. Entscheidend ist, daß immer bestimmte, durch die Prozeßdynamik bedingte Zeitforderungen zwischen Aufnahme und Ausgabe der Information eingehalten werden müssen. Es entsteht also die neue Situation, daß die „Rechenergebnisse" nur dann brauchbar sind, wenn sie nicht nur richtig, sondern auch in einer vorgegebenen Zeit zur Verfügung stehen. Für diese Klasse von Anwendungen wurde der Begriff **Echtzeitverarbeitung** (real time processing) eingeführt. Technisch gesehen, müssen diese Rechner mit spezifischen Details ausgerüstet werden, um insbesondere ihre Aufnahmebereitschaft zu beschleunigen (Unterbrechungssystem) und ihre Zuverlässigkeit zu erhöhen. Im Gegensatz zur EDV-Anlage, die nacheinander die unterschiedlichsten Aufgaben für mehrere Nutzer lösen muß und damit ständig mit anderen Programmen geladen wird, laufen die Steuerrechner ständig mit einem Programm. Die Aufspaltung in die Linie der EDV-Anlagen und der Prozeßrechner ist auch in den folgenden Entwicklungsetappen zu erkennen, obgleich die technischen Unterschiede beider Rechnertypen wieder weitgehend verschwinden.

Die 3. Etappe wurde von den Rechnern der 3. Generation (ESER-Anlagen) geprägt. Sie sind eine direkte Weiterentwicklung der EDV-Anlagen der 2. Generation. Die Verfügbarkeit integrierter Schaltkreise, verbunden mit verbesserten Rechnerstrukturen, führte zu einer weiteren Steigerung des Leistungs-Kosten-Verhältnisses dieser großen Rechenanlagen. Charakteristisch für diese Anlagen

ist ein modularer Aufbau, bestehend aus Zentraleinheiten (Digitalrechnern) mit aufsteigender Leistungsfähigkeit (Rechnerfamilien) und aus einer Palette verschiedener Eingabe/Ausgabe-Geräte und Massenspeicher (Magnetplatten-, Magnetbandgeräte). Damit bot sich erstmals die Möglichkeit, an das Anwendungsprofil angepaßte Konfigurationen zusammenzustellen. Obwohl diese Anlagen vorwiegend für die Zwecke der kommerziellen Datenverarbeitung eingesetzt wurden und werden, sind sie aufgrund ihrer technischen Merkmale ebenso für wissenschaftlich-technische Berechnungen als auch für Prozeßsteuerungen einsetzbar (Universalrechner). In jedem Fall handelt es sich aber um Rechenanlagen, die aufgrund ihrer Abmessungen und Klimaforderungen eigene Räume zur Aufstellung erfordern **(Rechenzentrum)**. Die Linie der Prozeßrechner mündete in dieser 3. Etappe zu einem Teil wieder in die Linie der EDV-Anlagen, entwickelte sich aber auch eigenständig weiter, um insbesondere volumenmäßig (zwangsläufig auch leistungsmäßig) kleinere Anlagen bereitzustellen. Damit entstanden neue Begriffe, wie **Kleinrechner, Kleinsteuerrechner, Kompaktrechner, Minirechner.**

Die Kleinrechner eröffneten zugleich ein weiteres Einsatzgebiet der elektronischen Rechentechnik, nämlich die dezentrale Datenverarbeitung und die Verfügbarkeit von Rechnerleistung in der Nähe des Arbeitsplatzes. Die Nachteile des Rechenzentrumbetriebs, insbesondere der begrenzte Zugriff des Nutzers und die langen Programmumschlagzeiten, konnten damit bei einigen Anwendungen wieder vermieden werden. Um das Bild zu vervollständigen, muß auch vermerkt werden, daß in der 3. Etappe die Außenseiterlinie der **Superrechner** begann. Zielstellung beim Entwurf dieser Maschinen war und ist, ohne Rücksicht auf ökonomische Grenzen die für den jeweiligen Stand der Elektronik maximale Rechenleistung zu erreichen. Das Leistungsvermögen solcher Anlagen reicht inzwischen bis zu einigen 100 Millionen Rechenoperationen je Sekunde. Sie sind damit die eigentlichen Nachfolger der Digitalrechner der 1. Generation in dem Sinne, daß sie für die Ausführung wissenschaftlich-technischer Berechnungen extremen Umfangs bestimmt sind (z. B. in der wissenschaftlichen Forschung, für die Simulation komplizierter Systeme). Um die Notwendigkeit und die Leistungsfähigkeit solcher Rechner zu demonstrieren, wollen wir folgendes Zahlenbeispiel betrachten: Für die Vorausberechnung der globalen Wetterentwicklung mit Hilfe eines mathematischen Modells, das die physikalischen Vorgänge in der Atmosphäre näherungsweise nachbildet, seien nach Eingabe des Ist-Zustandes für eine 24-h-Prognose $10^{10}...10^{11}$ Rechenoperationen notwendig. Ein mittlerer Rechner der 3. Generation mit einer typischen Operationsgeschwindigkeit von etwa 300 000 Operationen je Sekunde würde dazu eine Rechenzeit zwischen 10 und 100 Stunden benötigen, also letztlich der realen Wetterentwicklung hinterherlaufen. Ein Superrechner mit 100 Millionen Operationen je Sekunde leistet die gleiche Arbeit in 2...20 Minuten.

In den 70er Jahren wurde nun eine 4. Etappe der elektronischen Rechentechnik eingeleitet, die durch die **Mikrorechner** geprägt ist.

Eigentlich sind die Mikrorechner „nur" ein folgerichtiges Ergebnis der Weiterentwicklung der integrierten Schaltungstechnik, in dem es technisch möglich wird, zunehmend größere Funktionseinheiten auf einem Schaltkreis zu integrieren und damit größere Baugruppen bzw. komplette Rechner mit dieser Technologie herzustellen. In funktioneller Hinsicht unterscheiden sie sich nicht von den anderen Rechnertypen. Leistungsmäßig bildeten sie (zumindest anfangs) die unterste Schicht der Rechner. Trotzdem haben sie einen Umwälzungsprozeß aus-

gelöst, der durchaus berechtigt, von einer neuen Etappe der elektronischen Rechentechnik zu sprechen. Wodurch ist dies begründet? Die neue Qualität dieser Rechner besteht darin, daß leistungsfähige Digitalrechner in Form von Baugruppen bzw. Bauelementen zur Verfügung stehen und nicht mehr wie bisher als eigenständige Geräte produziert werden.

Dies hat vor allem 2 Auswirkungen:

1. Dezentralisierung der Rechnerleistung,
2. Gerät-Rechner-Integration.

Zu 1. Da das Kosten-Leistungs-Verhältnis in der bisherigen Entwicklung zugunsten der großen Rechenanlagen sprach, herrschte als Grundtendenz die zunehmende Zentralisierung der Rechnerleistung vor. Dementsprechend mußte sowohl in der Datenverarbeitung als auch bei wissenschaftlich-technischen Rechnungen der Transport der Eingabe- und Ausgabeinformationen entweder körperlich in Form von geeigneten Datenträgern oder elektrisch mit Hilfe von Datenübertragungseinrichtungen zu bzw. von den Rechenzentren bewerkstelligt werden. In der Prozeßsteuerung war man bemüht, möglichst viele Aufgaben durch einen zentralisierten Rechner parallel abarbeiten zu lassen, was zu einer weiträumigen Verkabelung zwischen Rechner und Peripherie führte.

Dieser Prozeß wird beim Einsatz der Mikrorechner nun wieder umgekehrt, indem eine zunehmende **Dezentralisierung der Rechnerleistung** ökonomisch möglich wird. Die Vorteile einer Dezentralisierung sind in erster Linie

- auf dem Gebiet der Datenverarbeitung und Rechentechnik die Verfügbarkeit von Verarbeitungs- und Speicherkapazität am Arbeitsplatz,
- auf dem Gebiet der Prozeßsteuerung die funktionelle Entflechtung der Steueraufgaben und damit eine bessere Übersicht beim Systementwurf und bei der Wartung sowie eine erhöhte Zuverlässigkeit durch die begrenzte Wirkungsbreite eines Rechnerausfalls.

Die Dezentralisierung führt aber nicht zwangsläufig zu einer Vereinzelung in viele isolierte Rechner. Vielmehr ist es bei der Mehrzahl der Anwendungen erforderlich, zwischen den dezentralen Rechnern Informationen auszutauschen bis hin zum Aufbau hierarchischer Rechnerstrukturen. Der **Rechnerverbund** in Form von Mehrrechnersystemen (bei lokal konzentrierter Anordnung der Mikrorechner) oder in Form von Rechnernetzen (bei entfernt angeordneten und über Leitungen bzw. Nachrichtennetze verbundenen Rechnern) wird daher zunehmend das Bild dieser Etappe der Rechentechnik prägen.

Zu 2. Durch die Volumen- und Kostenreduzierung eröffnet sich eine Vielzahl neuer Anwendungsgebiete für Rechner, deren gemeinsames Merkmal die **Gerät-Rechner-Integration** ist. Durch die direkte räumliche Anordnung von Rechnern in vorhandenen Geräten (z. B. in Verarbeitungsmaschinen, Haushaltsgeräten, Meßeinrichtungen, Fahrzeugen usw.) sind neue Leistungsmerkmale (Intelligenz) erreichbar. Aber auch bisher vollkommen unbekannte Anwendungen werden erschlossen (z. B. elektronische Spiele).

Im Bild 1.1. wurde der Mikrorechner in die Prozeßrechnerlinie eingeordnet. Wenn wir die Steuerung von Geräten und Maschinen aller Art unter dem Oberbegriff Prozeßsteuerung einordnen, entspricht dies durchaus dem ursprünglichen und auch gegenwärtigen Einsatzprofil der Mikrorechner. Dabei darf aber nicht übersehen werden, daß sich inzwischen auch die Rechnerhersteller selbst dieser Schaltkreise bedienen (in erster Linie die Hersteller von Büromaschinen und

Kleinrechnern, aber auch bei den Großrechnern werden die Vorteile der Großintegration zunehmend genutzt). Eine Definition in dem Sinne – Mikrorechner sind Rechner auf Basis großintegrierter Schaltkreise – kann damit nur für eine kurze Übergangsphase Gültigkeit haben, da zukünftig nur noch Mikrorechner in diesem allgemeinen Sinne existieren werden. So erhebt sich letztlich die Frage, was eigentlich das Typische eines Mikrorechners ist und ob eine eigenständige Darstellung der Mikrorechentechnik überhaupt notwendig ist. Zur Beantwortung dieser Frage und zur Abgrenzung des in diesem Buch behandelten Gegenstandes wollen wir daher zunächst die wichtigsten Anwendungsgebiete der Mikrorechner in der Daten- und Rechentechnik selbst analysieren. Es lassen sich die folgenden 3 Produktlinien erkennen:

1. **Dezentrale Rechner für universelle oder spezielle Entwurfsarbeiten.** Das Spektrum dieser Rechner reicht von den programmierbaren Taschenrechnern über Heimcomputer bis zu sehr leistungsfähigen Systemen. Obwohl diese Geräte volumenmäßig oft nur Auf-Tisch-Geräte sind, kann mit ihnen aufgrund ihrer leistungsfähigen Peripherie (z. B. grafische Farbbildschirm-Ausgabe, Plotter) und aufgrund ihrer auf Massenspeichern (Platten- und Floppy-disk-Speicher) verfügbaren Programmsysteme inzwischen ein großer Teil der bisher den Rechenzentren vorbehaltenen Aufgaben direkt am Arbeitsplatz abgewickelt werden.
2. **Geräte zur Rationalisierung der Büroarbeit und dazu angepaßte Kommunikationssysteme.** Während die erste Produktlinie als Dezentralisierung der wissenschaftlich-technischen Rechnung gesehen werden kann, handelt es sich hier um die Dezentralisierung der Datenverarbeitung. Die Ablösung der mechanischen bzw. elektromechanischen Büromaschinen durch elektronische, auf Mikrorechnern basierende Geräte und die dezentrale Anwendung der elektronischen Datenverarbeitung und -speicherung im Büro wird ein wichtiges Rationalisierungsvorhaben in den folgenden Jahren werden. Damit wird zugleich die Wandlung der Kommunikationstechnik einhergehen, da sich der Anschluß dieser Bürocomputer an Nachrichtennetze anbietet und somit erweiterte Möglichkeiten für den Austausch von Text- und Bildinformationen auf elektrischem Wege geboten werden (Teletex- und Faksimiledienste, Bildschirmtext).
3. **Terminals für weiträumige Informationssysteme.** Obwohl die bisher betrachteten Geräte in erster Linie für den dezentralen Einsatz vorgesehen sind, bedeutet es keinen beträchtlichen Mehraufwand, sie so auszustatten, daß eine Kopplung mit weiteren dezentralen Systemen bzw. mit zentralen Rechnern über Datenübertragungseinrichtungen möglich wird. Sie sind dann Endpunkte (Terminals) eines oft weiträumigen Informationssystems (Datenbanken, Auskunfts- und Buchungssysteme). Für eine Reihe von speziellen Anwendungen wird es notwendig, funktionsbezogene Terminals auf der Basis von Mikrorechnern zu fertigen, z. B. Bedienplätze für die Platzreservierung bei Verkehrsgesellschaften, Schalterterminals für Sparkassen und Banken, Kassenterminals usw.

Allen diesen Produkten ist gemeinsam, daß sie im Kern einen oder mehrere Mikrorechner enthalten und durch spezielle Peripherie und Software auf die jeweilige Funktion zugeschnitten sind. Sie werden also von einem Hersteller hardwaremäßig und zum Teil auch softwareseitig als komplette Finalprodukte

geliefert. Dem Anwender treten damit Geräte entgegen, deren Nutzung meist keine Kenntnisse über Arbeitsweise und Programmierung der intern verwendeten Mikrorechner erfordert. Dies trifft auch auf viele rechnerbestückte Systeme auf dem Gebiet der industriellen Steuerungen zu, wie beispielsweise auf Industrieroboter, auf numerisch gesteuerte Maschinen, auf rechnergesteuerte Meß- und Prüfeinrichtungen. Solche Kenntnisse werden wie bisher nur von dem begrenzten (zwar ständig wachsenden) Kreis von Spezialisten benötigt, deren Aufgabe die Entwicklung und Einsatzvorbereitung solcher Systeme ist.

Darüber hinaus werden aber Mikrorechner in Form integrierter Schaltkreise bzw. Schaltkreissätze produziert, die für die Bauelementeindustrie Endprodukte, für die Anwenderindustrie aber **Zwischenprodukte (OEM-Baugruppen)** sind. Die Einsatzmöglichkeiten für derartige Rechner müssen in erster Linie durch die Spezialisten der jeweiligen Anwendungsgebiete erkannt werden. Dazu sind fundierte Kenntnisse über die Hardware- und Softwareseite der Mikrorechentechnik bei einem wesentlich breiteren Personenkreis als bisher erforderlich. Das ist letztlich der Grund für eine gesonderte Darstellung der Mikrorechentechnik.

Das wesentliche Merkmal der 4. Etappe der elektronischen Rechentechnik besteht also darin, daß Rechner zu einem allgemein verfügbaren und handlichen Funktionselement werden (vergleichbar mit solchen klassischen Funktionselementen wie Elektromotor, Stellventil, Schaltgetriebe usw.) und folglich deren Arbeitsweise, Kennwerte und Einsatzmöglichkeiten zum Grundwissen eines Ingenieurs und Technikers zählen müssen.

Diese Aussage wird am besten durch die Tatsache belegt, daß 10 Jahre nach der 1971 erfolgten Ankündigung des ersten Mikrorechner-Schaltkreissatzes bereits über 140 Millionen solcher Schaltkreise jährlich verlötet werden.

Wir wollen daher zum Abschluß dieser einleitenden Betrachtungen den Mikrorechnerbegriff wie folgt präzisieren:

> Mikrorechner sind mit Hilfe hochintegrierter elektronischer Schaltkreise realisierte Digitalrechner, die als Zwischenprodukte hergestellt und für den konkreten Einsatzfall beim Anwender hardware- und/oder softwareseitig komplettiert werden.

Daraus ergeben sich folgende Aspekte für die Darstellung der Mikrorechentechnik, die auch den folgenden Kapiteln zugrunde gelegt wurden:

1. **Der Mikrorechneranwender muß über Kenntnisse der Hardware- und Softwareseite verfügen.** Die bisherige Entwicklung der Rechentechnik hat zu einer „Entfremdung" zwischen Nutzer und Rechner-Hardware geführt. Die Rechner wurden als Hilfsmittel benutzt und zu diesem Zweck vom Hersteller konfiguriert und auch oft bereits mit umfangreichen Programmsystemen (Betriebssystemen) ausgerüstet, so daß zur Nutzung bestenfalls die Erweiterung der Software erforderlich wurde. Im Gegensatz dazu sind die als Zwischenprodukte hergestellten Mikrorechner bzw. Mikrorechnerbausteine für den Anwender Arbeitsgegenstand, in dem sowohl die konkrete Hardwarekonfiguration als auch die dazu passende Software entwickelt werden muß. Dieser Systementwurf wird zwar durch Bereitstellung vorgefertigter Bausteine (sowohl großintegrierte Schaltkreise und Module in Form von Leiterplatten für die Hardware als auch Programmpakete für die Software) erleichtert, erfordert jedoch entsprechende Grundkenntnisse.

2. **Der Mikrorechneranwender muß bevorzugt über Kenntnisse auf funktioneller Ebene verfügen.** Die durch die Fortschritte in der Mikroelektronik ermöglichte Großintegration von Funktionseinheiten hat zur Folge, daß sich der Hardwareentwurf wesentlich vereinfacht und beim Anwender dazu keine tiefergehenden elektronischen Kenntnisse erforderlich sind. Es wird deshalb in diesem Buch auf die Darstellung des „elektronischen Hintergrundes" der Mikrorechentechnik bewußt verzichtet. Die interne Realisierung der Schaltkreise und die zum Einsatz kommende Halbleitertechnologie sind für den Anwender nur insofern von Interesse, als dadurch einige Grundparameter (z. B. Schaltgeschwindigkeit, Leistungsaufnahme, Packungsdichte) bestimmt werden. Entscheidend sind vielmehr Kenntnisse über die nach außen wirkende Funktion, um durch Komposition der Bausteine und Module die für den vorliegenden Anwendungsfall optimale Rechnerkonfiguration zu finden.
3. **Der Mikrorechneranwender hat es mit Anwendungsfällen zu tun, die zur Klasse der Echtzeitsysteme zählen.** Die Integration eines Rechners in ein Gerät, die Anwendung zur Automatisierung technischer oder technologischer Prozesse oder zur Unterstützung eines von Menschen ausgeführten Entwurfsprozesses erfordert immer das Einhalten von Zeitbedingungen bei der Programmabarbeitung. Wir werden deshalb im folgenden Kapitel zunächst diese Anwendungsklasse näher analysieren.

2. Grundlagen

2.1. Echtzeitsysteme

Einleitend hatten wir vermerkt, daß Mikrorechner insbesondere als Steuereinrichtungen für Geräte und Prozesse eingesetzt werden und daß bei diesen Anwendungen die Notwendigkeit einer zeitbezogenen Arbeitsweise besteht, die sog. Echtzeitverarbeitung.

Die Anforderungen, die dabei an das „Zeitgefühl" des Steuerrechners gestellt werden, kann man in 2 Klassen einteilen:

1. Die Reaktion des Rechners auf ein bestimmtes Ereignis (Eingangssignal) soll nach einer exakt vorgegebenen Zeitdauer bzw. zu einer bestimmten Uhrzeit erfolgen. Dazu muß der Rechner über (meist hardwaremäßig realisierte) Zeitmeßeinrichtungen verfügen.
2. Die Reaktion des Rechners auf ein Ereignis darf eine festgelegte Zeitdifferenz nicht überschreiten, kann ansonsten so schnell wie möglich erfolgen. Dazu ist es erforderlich, die Anzahl der dem Rechner zu übertragenden Aufgaben mit seiner Arbeitsgeschwindigkeit richtig abzustimmen und insbesondere durch geeignete Organisation dafür zu sorgen, daß seine Leistung effektiv genutzt wird.

Ziel dieses Abschnitts soll es sein, nach Darstellung einiger Grundbegriffe anhand von Beispielen aufzuzeigen, welche Vielzahl unterschiedlicher Aufgaben auf solche Echtzeitsysteme führt und folglich aus der Sicht der Informationsverarbeitung mit gleichen technischen und organisatorischen Mitteln gelöst werden kann.

2.1.1. Grundstruktur eines Echtzeitsystems

Die allgemeine Grundstruktur eines Echtzeitsystems zeigt Bild 2.1. Die grundsätzliche Zielstellung eines solchen Systems ist, ein **Objekt** (Gerät, Maschine, Prozeß usw.) unter Führung eines **Bedieners** zu einer vorgegebenen Arbeitsweise zu veranlassen. Die Darstellung nach Bild 2.1 läßt vermuten, daß dem Bediener dabei die aktive Rolle zukommt. In Wirklichkeit bildet der als Steuereinrichtung eingesetzte Rechner den Kern dieses Informationsverarbeitungssystems, in dem durch dessen Programmierung sowohl die Zusammenarbeit mit dem Objekt als auch mit dem Bediener festgeschrieben wird. Die Aufgabe des Bedieners besteht lediglich darin, dem Steuerrechner auf Anforderung bestimmte Parameter zu übergeben, die bei dessen Programmierung als „Freiheitsgrade" noch offengelassen wurden. Der Handlungsspielraum des Bedieners ist also von vornherein durch den Programmierer des Steuerrechners festgelegt worden.

Aus Bild 2.1 ist weiterhin ersichtlich, daß zur Kopplung des Steuerrechners an Objekt und Bediener **periphere Einrichtungen** erforderlich sind:

- **Stelleinrichtungen** zur Beeinflussung des Objekts,
- **Meßeinrichtungen** (Sensoren) zur Rückkopplung des Objektzustandes an den Steuerrechner,
- **Bedien- und Anzeigeeinrichtungen** für die Zusammenarbeit mit dem Bediener.

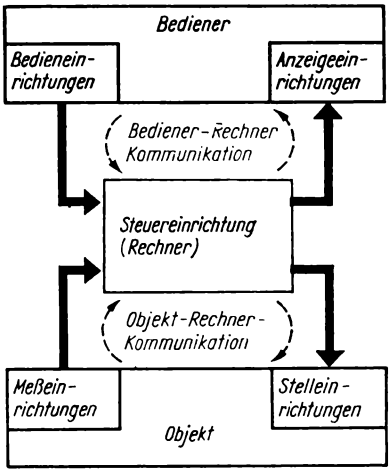


Bild 2.1. Grundstruktur eines Echtzeitsystems

(Die Funktion des Bedieners wird dabei in einigen Anwendungen vom Menschen, in vielen Fällen aber wiederum durch einen übergeordneten Rechner wahrgenommen.)

Die Stellsignale für das Objekt und die Anzeigesignale für den Bediener bilden damit die Menge der Ausgangsgrößen (Resultate) des Steuerrechners, während Meß- und Bediensignale seine Eingangsinformation sind. Oder anders betrachtet: Ein Echtzeitsystem besteht aus 2 von einem Rechner gesteuerten Kommunikationssystemen:

– Rechner-Objekt-Kommunikation
– Bediener-Rechner-Kommunikation.

Im wesentlichen unterscheiden sich diese beiden Systeme durch ihre Zeitforderungen. Während der Bediener-Rechner-Dialog, insbesondere wenn es sich um einen Mensch-Rechner-Dialog handelt, Reaktionszeiten im Sekundenbereich toleriert, sind beim Informationsaustausch zwischen Steuerrechner und Objekt meist wesentlich kleinere Reaktionszeiten einzuhalten.

2.1.2. Grundtypen von Echtzeitsystemen

Hinter der im Bild 2.1 angegebenen Grundstruktur verbirgt sich eine Vielfalt von konkreten technischen Systemen. Wir wollen im folgenden einige Beispiele für elektronische Steuerungen betrachten, die inzwischen zu den typischen Mikrorechneranwendungen zählen, und dabei zugleich eine Systematisierung vornehmen. Je nachdem, welche Kommunikationswege (Meß-, Stell-, Bedien- und Anzeigesignale) dominieren, lassen sich die nachstehend betrachteten Grundtypen von Echtzeitsystemen unterscheiden.

Steuerungssysteme ohne Rückkopplung

Diese Systeme (Bild 2.2) sind durch das Fehlen von Meßsignalen als Rückkopplung zur Steuereinrichtung gekennzeichnet. Den Hauptinformationsfluß bilden die Stellsignale. Ein solches Verfahren setzt voraus, daß die Stellsignale am Objekt eine determinierte Wirkung haben und folglich keiner weiteren Kontrolle bedürfen.

Bezüglich des Grades der Bedienerkopplung kann es dabei große Unterschiede

geben. Nahezu ohne Bedienereinwirkung arbeiten Systeme, die entweder periodische bzw. sehr lange einmalige Abläufe zu organisieren haben (z. B. Steuerung einer Verkehrssignalanlage ohne Messung der Verkehrsströme). Hier beschränkt sich die Bedienerführung in der Regel auf das Einschalten und Ausschalten bzw. auf langfristige Programmumschaltungen.

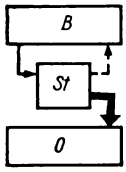


Bild 2.2. Steuerungssystem ohne Rückkopplung

Ganz im Gegensatz dazu stehen solche Steuerungssysteme, die direkt von einem Bediener geführt werden. Betrachten wir zum Beispiel eine elektronische Schreibmaschine. Der Bediener betätigt die Tastatur, die daraufhin Bediensignale an den Steuerrechner liefert. Der Steuerrechner erzeugt Stellsignale für das Druckwerk (Objekt) zum Zeichenabdruck und zur Positionierung des Drucksystems. Der Bediener hat dabei den Eindruck, das Objekt direkt zu steuern. In Wirklichkeit erfolgt aber eine Entkopplung zwischen Bediener und Objekt durch die Steuereinrichtung.

Die Entkopplung eröffnet gänzlich neue Leistungsmerkmale, die insbesondere den Bedienkomfort erhöhen, Fehlbedienungen und menschliche Unzulänglichkeiten bei der Bedienung vermeiden. Bei der elektronischen Schreibmaschine sind das beispielsweise die Beseitigung des Einflusses des Anschlagdruckes auf das Druckbild, die Möglichkeit, mit Sondertasten ganze Buchstabenfolgen auf einmal zu drucken usw.

Eine weitere Modifikation dieses Grundtyps von Echtzeitsystemen ergibt sich, wenn eine Rückkopplung zum Bediener durch Anzeigesignale erfolgt. Bleiben wir beim Beispiel der Schreibmaschine: Wird dafür gesorgt, daß vor dem Auslösen des Druckvorganges erst eine zeilen- oder seitenweise Darstellung auf einem Bildschirm (Anzeigeeinrichtung) erfolgt, ergibt sich ein wesentlich komfortablerer Schreibplatz als mit der herkömmlichen Schreibmaschine, da der Bediener seine Eingaben nochmals kontrollieren kann.

Eine weitere Variante der Bedienerführung eines Steuerungssystems liegt z. B. beim Einsatz von Rechnern in Rundfunk- und Fernsehempfangs- bzw. -aufzeichnungsgeräten vor. Um das Einschalten von Kanälen über größere Zeiträume vorplanen zu können, ist der Rechner so programmiert, daß er die Bedienereingaben abspeichert, laufend die vom Bediener angegebenen Schaltzeitpunkte mit einer im Rechner realisierten Echtzeituhr vergleicht und danach Stellsignale an das Objekt zum Einschalten bzw. Ausschalten und zur Senderwahl auslöst. Der Unterschied zu den oben betrachteten Beispielen besteht hier darin, daß außer der funktionellen Entkopplung zwischen Bediener und Objekt ein beliebig einstellbarer Zeitabstand zwischen dem Bedienerkommando und dem Auslösen des Steuervorganges möglich ist.

Regelungssysteme (Steuerungssysteme mit Rückkopplung)

Diese Systeme sind durch den geschlossenen Kommunikationskreis zwischen Steuerrechner und Objekt charakterisiert (Bild 2.3). Damit wird die Wirkung des Stellsignals am Objekt beobachtbar und entsprechend durch Vergleich mit einem vorgegebenen Zielwert (Sollwert) korrigierbar.

Auch bei diesen Systemen existiert ein breites Spektrum bezüglich der Bedienerführung:

Eine **Festwertregelung** liegt vor, wenn die Sollwerte einmalig vorgegeben bzw. nur in großen Zeitabständen verändert werden. Der Bedienvorgang entfällt entweder („Festschreiben" der Sollwerte bereits durch den Programmierer in das Programm des Steuerrechners) oder erfolgt durch einmalige bzw. selten auftretende Bediensignale. Betrachten wir als Beispiel die Klimaregelung eines Raumes: Vom Steuerrechner werden in bestimmten Zeitabständen z. B. Temperatur und Luftfeuchtigkeit erfaßt und daraufhin versucht, durch Stellsignale auf entsprechende Aggregate diese Größen trotz schwankender Umweltbedingungen so genau wie notwendig an die vorgegebenen Sollwerte anzugleichen.

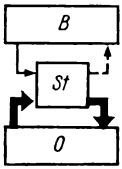


Bild 2.3. Steuerungssystem mit Rückkopplung (Regelungssystem)

Man spricht dagegen von einer **Nachlauf- oder Folgeregelung,** wenn die Sollwerte durch ein Bediensystem in einer für das Steuerungssystem „merklichen" Geschwindigkeit geändert werden. Ein typisches Beispiel hierfür sind die Lageregelungen bei gesteuerten Maschinen und Industrierobotern. Bei diesen Systemen ist es erforderlich, eine oft große Anzahl von mechanischen Bewegungen durch elektrische oder hydraulische Antriebe mit hoher Geschwindigkeit und hoher Präzision auszuführen. Betrachten wir als Zahlenbeispiel die Bewegung eines Roboterarmes um eine Achse mit einer Geschwindigkeit von 180°/s und einer Positionsgenauigkeit von 1°. Wird dem Steuerrechner vom Bediensystem eine neue Position (Sollwert) vorgegeben, schaltet dieser über ein Stellsignal den Antrieb ein und verfolgt laufend das von einem Lagemeßsystem abgerufene Meßsignal. Laufend heißt in diesem Fall, daß mindestens 180 Meßwerte je Sekunde erfaßt werden müssen, also die Eingabe des Meßsignals in den Rechner etwa im Abstand von 5,5 ms erfolgt. Die eingegebene aktuelle Position wird vom Rechner mit dem Sollwert verglichen und nach Berechnung der Differenz entschieden, ob mit voller Geschwindigkeit weitergefahren oder eine Bremsung derart eingeleitet wird, daß trotz der Trägheit der bewegten Massen der Endwert ohne „Überfahren" erreicht wird. Zwischen den Meßwerteingaben ist also jeweils die Abarbeitung eines Programmes im Steuerrechner erforderlich, das die neuen Stellsignale berechnet. Bei den gegenwärtig typischen Arbeitsgeschwindigkeiten für Mikrorechner sind in 5,5 ms mindestens 2000···3000 Befehle ausführbar, eine für ein solches Problem ausreichende Menge.

Eine weitere Klasse von Regelungssystemen – die **Optimalwertregelung** – ist dadurch gekennzeichnet, daß nicht das Einhalten eines Sollzustandes gefordert ist, sondern vielmehr ein Gütekriterium aus mehreren Zustandsgrößen des Objekts formuliert wird. Ziel ist es, durch systematisches Probieren (Verändern der Stellsignale) einen solchen Objektzustand einzustellen, daß dieses Gütekriterium maximal wird. Der Steuerrechner erfaßt zu einem bestimmten Zeitpunkt die Meßgrößen, berechnet daraus den momentanen Wert des Gütekriteriums und bewirkt danach eine geringfügige Änderung einer Stellgröße. Die Auswirkung dieser Veränderung wird nun vom Rechner durch fortlaufende Messung und Güteberechnung verfolgt. Bei negativem Einfluß wird entsprechend die Stellgröße wieder zurückgesetzt, bei positivem Einfluß in gleiche Richtung weiter verändert.

Meß- und Prüfsysteme

Bei diesen Systemen steht die Beobachtung eines Objekts im Vordergrund. Die hauptsächlichen Informationsflüsse werden dementsprechend durch die Meß- und Anzeigesignale gebildet (Bild 2.4). Dem Rechner (der Begriff Steuerrechner

wird bei diesen Systemen z. T. unzutreffend) obliegt die Aufgabe der Meßsignalerfassung und -verarbeitung sowie der Aufbereitung der Anzeige. Im Vergleich zur direkten Anzeige einer Meßgröße bietet die Entkopplung die Möglichkeit, den zeitlichen Verlauf der Meßsignale über endliche Zeiträume zu speichern und somit für eine wiederholte Anzeige und Verarbeitung bereitzuhalten. Als Verarbeitungsaufgaben sind dabei vorrangig die Verdichtung der vom Objekt gelieferten Information (Bildung von Mittelwerten, Ermittlung von Grenzwerten u. ä.) sowie die Verknüpfung mehrerer Meßgrößen (Produktbildung, Berechnung von Gütekriterien) zu lösen.

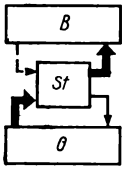


Bild 2.4. Meß- und Prüfsysteme

Betrachten wir als Beispiel aus der Medizintechnik die EKG-Messung. Das direkte Meßverfahren (Aufzeichnung des EKGs mit Hilfe von Schreiber oder Display) ermöglicht zwar, den momentanen Signalverlauf in allen Einzelheiten zu erfassen, ist aber aufgrund der anfallenden Informationsmenge kein geeignetes Verfahren für eine Dauerbeobachtung bzw. für die parallele Beobachtung mehrerer Patienten. Der Einsatz eines Rechners, der den Verlauf der Meßgröße analysiert und nach anormalen Zuständen absucht, schafft daher vollkommen neue Leistungsmerkmale. So ist es beispielsweise möglich, bei Überschreiten vorgegebener Grenzwerte einen Alarm auszulösen (Intensivstationen), über einen längeren Zeitraum die Trends der EKG-Parameter eines Patienten zu registrieren oder kurzzeitige und zufällige Unregelmäßigkeiten mit dem Zeitpunkt ihres Auftretens aufzulisten und erst auf Anforderung zur Anzeige zu bringen.

Aber auch auf dem Gebiet der elektronischen Meßtechnik wird durch die Einbeziehung der Rechentechnik eine neue Generation von Meßgeräten geschaffen: Der Oszillograf, das universelle Meßgerät der Elektronik, ermöglicht die Darstellung periodischer Signalverläufe. Bei einmalig auftretenden Signalen müssen spezielle Oszillografen mit extrem lange nachleuchtenden Bildröhren verwendet werden, damit der Signalverlauf für eine längere Auswertungszeit zur Verfügung steht. Die Kopplung Rechner und Oszillograf führt nun dazu, daß die Messung des einmaligen Vorgangs zum Normalfall wird. Die Meßgröße wird vom Rechner abgetastet, eine Analog-Digital-Wandlung vorgenommen und als Meßreihe abgespeichert. Die Anzeige kann damit vom Meßvorgang vollkommen zeitlich entkoppelt werden. Eine beliebige Dehnung bzw. Komprimierung der Zeitachse, der Beginn der Aufzeichnung erst nach Auftreten bestimmter Signalzustände (z. B. nach Störimpulsen), die parallele Erfassung mehrerer Signale sind nur einige Beispiele für die neuen Leistungsmerkmale eines solchen rechnergesteuerten Oszillografen. Natürlich begrenzen die Arbeitsgeschwindigkeit des Rechners und das Auflösungsvermögen bei der Analog-Digital-Wandlung der Signale die Anwendbarkeit eines solchen Meßgeräts. Ist ausschließlich die Darstellung binärer Signale erforderlich, dann können die Analog-Digital-Umsetzer am Eingang des Rechners entfallen. Derartige Spezialoszillografen für binäre Signale mit integriertem Rechner werden als **Logikanalysatoren** bezeichnet.

Beruhten die betrachteten Beispiele nur darauf, bei an sich klassischen Meßverfahren eine Entkopplung des Meßwertaufnehmers und der Anzeigeeinrichtung durch den Rechner vorzunehmen, so bietet der nächste Schritt, nämlich die Beeinflussung des Meßobjekts durch vom Rechner erzeugte Stellsignale, eine noch höhere Stufe bei der Automatisierung der Meß- und Prüfprozesse. Derartige **Prüf- und Diagnosesysteme** ermöglichen die automatische Aufnahme von Meßreihen und die Durchführung von Meßprogrammen.

Dialogsysteme

Viele Echtzeitsysteme sind dadurch charakterisiert, daß scheinbar nur noch die Kommunikation zwischen Bediener (Nutzer) und Rechner vorhanden ist (Bild 2.5). Eine genauere Analyse läßt jedoch erkennen, daß die Grundstruktur gemäß Bild 2.1 letztlich doch gültig ist, wobei das Objekt aber ausschließlich eine informatorische Seite aufweist.

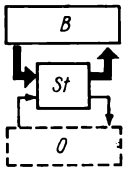


Bild 2.5. Dialogsysteme

Am deutlichsten wird das bei den **Simulatoren.** Solche Systeme werden meist zum Zweck des gefahrlosen Erlernens der Bedienung realer Prozesse und Geräte geschaffen (z. B. Flugsimulatoren). Simuliert wird dabei nur das Objekt, indem dessen Verhalten [Zusammenhang zwischen den Ausgangs-(Meß-)größen und den Eingangs-(Stell-)größen] durch einen Rechner mit einem entsprechenden Programmsystem nachgebildet wird. Dieser Rechner kann dabei separat vorhanden sein oder aber Steuereinrichtung und Objekt werden durch einen Rechner realisiert.

In diese Kategorie sind auch viele der inzwischen auf der Basis von Mikrorechnern geschaffenen **elektronischen Spiele** einzuordnen. Sind sie doch nichts anderes als Simulatoren für Unterhaltungszwecke.

Der nächste gedankliche Schritt besteht nun darin, auch solche Systeme einzubeziehen, bei denen es um die Simulation geistiger Tätigkeiten und Fähigkeiten geht. Betrachten wir als Beispiel einen **elektronischen Taschenrechner** mit festem Funktionsumfang. Er besteht aus einem Mikrorechner, einem Tastensatz als Bedienelement und einer Displayzeile als Anzeigeeinrichtung. Der Rechner ist dabei so programmiert, daß er laufend den Tastensatz „beobachtet" und bei Erkennen einer gedrückten Taste ein tastenspezifisches Verarbeitungsprogramm startet. Diese Programme können unterschiedliche Aufgaben übernehmen. Bei Eingabe einer 1. Ziffer muß die Ziffer zur Anzeige gebracht und ihr Zahlenwert in einem Register gespeichert werden. Bei Eingabe weiterer Ziffern muß die Anzeige weitergeschoben und der Zahlenwert entsprechend der Dezimaldarstellung neu berechnet werden. Wird dagegen eine Funktionstaste (z. B. sin) erkannt, so wird ein umfangreicheres Programm gestartet, das den Funktionswert berechnet und für die Anzeige bereitstellt.

Das Gesamtprogramm des im Taschenrechner enthaltenen Mikrorechners gliedert sich also in 2 Komponenten:

1. Steuerprogramm (Dieses Programm steuert den Dialog mit dem Nutzer, indem es die Tastatur abfragt, die gedrückte Taste analysiert, spezifische Verarbeitungsprogramme aktiviert und die Anzeigeeinrichtung betätigt.)
2. Verarbeitungsprogramme (Diese Programme bestimmen den eigentlichen Leistungsumfang des Taschenrechners und bilden damit das zu steuernde (oder zu verwaltende) Objekt des Systems. Im Verhältnis zum Steuerprogramm gehören diese Programme einer untergeordneten Ebene an, die jeweils nur kurzzeitig zur Lösung einer speziellen Aufgabe aktiv wird. Danach wird die Regie wieder vom Steuerprogramm übernommen.)

Der Unterschied zu den anderen Typen von Echtzeitsystemen liegt also darin, daß bei den Dialogsystemen die Trennung zwischen Objekt und Steuerrechner nicht mehr unbedingt hardwaremäßig existieren muß, sondern oft nur in der Gliederung der Software eines Rechners erkennbar ist.

Damit ordnet sich auch die breite Anwendungsklasse der **Informations-, Auskunfts- und Buchungssysteme** hier ein, bei der durch Rechner der Zugriff von vielen Nutzern (Bedienern) zu einem oder mehreren Objekten gewährleistet wird, die jeweils bestimmte Funktionen der Informationsspeicherung und -verarbeitung leisten können.

2.1.3. Mehrrechneranordnungen/Hierarchische Systeme

Automatisierungsvorhaben führen im allgemeinen zu Echtzeitsystemen, die eine Mischung der oben behandelten Grundtypen sind und folglich komplexe Strukturen aufweisen.

Bisher war dabei die Tendenz zu verzeichnen, möglichst viele Funktionen auf einen Rechner zu konzentrieren. Dies führte auf Strukturen gemäß Bild 2.6: Ein zentraler Rechner steuert mehrere (oft auch funktionell voneinander unabhängige) Einzelobjekte oder kann von mehreren Bedienern (Nutzern) scheinbar gleichzeitig benutzt werden.

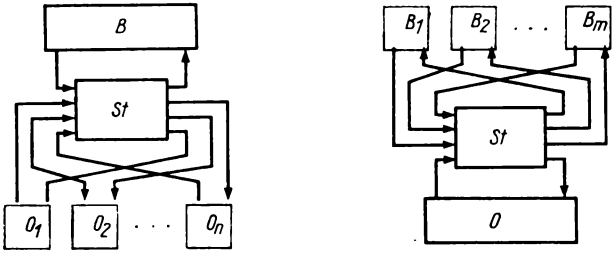


Bild 2.6. Strukturen mit zentralisierter Steuerung

Die Kosten-, Volumen- und Energiereduzierung und die individuelle Konfigurierbarkeit der Rechner, wie sie durch die Mikrorechentechnik gegeben sind, führen durch Dezentralisierung der Steuerfunktionen zunehmend zu Mehrrechneranordnungen.

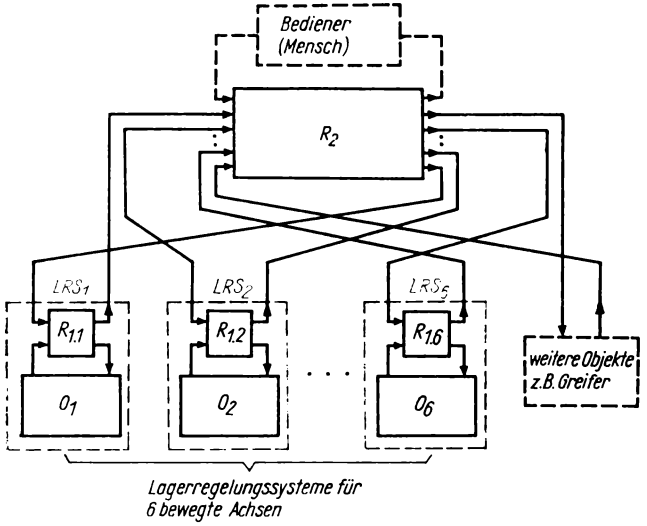


Bild 2.7. Steuerungsstruktur eines Industrieroboters mit verteilten Steuerrechnern (Beispiel)
LRS Lageregelungssystem

Bild 2.7 zeigt als Beispiel die Steuerungsstruktur eines Industrieroboters, bei dem allein zur Bewegung des Greifers im Arbeitsraum sechs Antriebe mit hoher Geschwindigkeit und Präzision zu steuern sind, so daß entsprechend die gleiche Anzahl von Lageregelungssystemen erforderlich ist. Bei dezentraler Realisierung erhält jedes Bewegungssystem (Objekt) einen separaten Steuerrechner, der seinerseits die Bedienanweisungen (anzufahrende Position) von einem übergeordneten Rechner übernimmt und diesen durch Anzeigesignale über bestimmte Zustände des Lageregelungssystems informiert. Dieser übergeordnete Rechner ist damit für die Steuerrechner der Lageregelungssysteme der Bediener. Werden mehrere solcher Roboter in einer technologischen Kette angeordnet, erweitert sich diese Hierarchie, indem die Steuerungen der einzelnen Roboter wiederum zu Objekten eines weiteren übergeordneten Rechners werden, wobei man davon ausgehen kann, daß diese Hierarchie nach oben letztlich immer durch einen Menschen als Bediener abgeschlossen wird (Bild 2.8). Bei solchen Hierarchien kann dementsprechend ein Rechner, je nachdem von welcher Ebene aus er gesehen wird, sowohl Steuerrechner als auch Bedienrechner oder sogar Objekt sein.

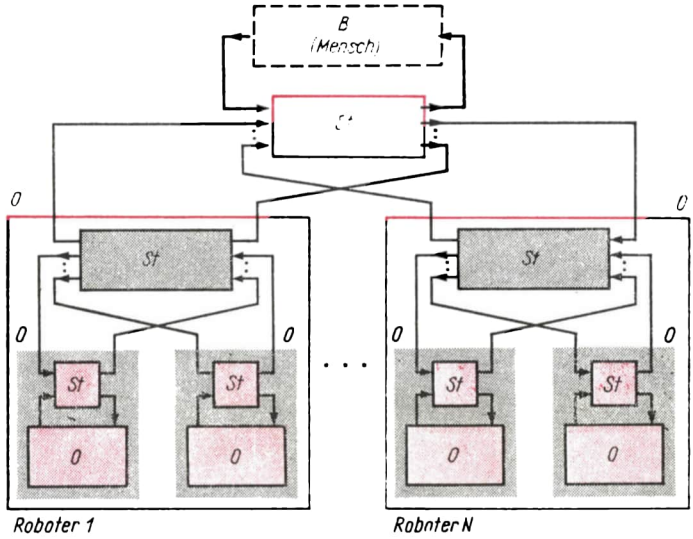


Bild 2.8. Hierarchisches Steuerungssystem (technologische Kette von N Industrierobotern)

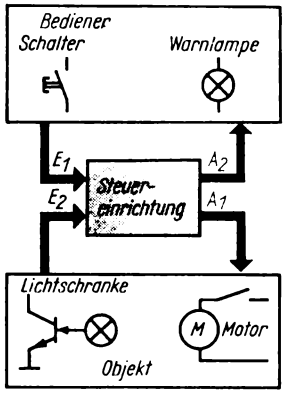
Es gilt dabei nach wie vor, daß zwar die Anweisungen über die auszuführenden Funktionen von oben nach unten erfolgen, daß aber die überhaupt ausführbaren Funktionen von unten nach oben festgelegt werden. Das heißt, bei der Programmerstellung für die Steuerrechner einer bestimmten Ebene wird festgelegt, welche Bedienereinwirkung von der nächsthöheren Ebene möglich und welche Information (Anzeigesignale) von oben abrufbar ist.

Diese Dezentralisierung führt nicht, wie man vielleicht anhand der Strukturbilder vermuten könnte, zu komplizierteren Systemen, sondern bewirkt im Gegenteil durch die objektbezogene Steuerungsstruktur eine Vereinfachung beim Systementwurf und größere Übersichtlichkeit. Dies schlägt sich auch meist in einer Verbesserung der Zuverlässigkeit (Begrenzung der Wirkungsbreite von Störungen), in einer leichteren Fehlersuche und damit in einer Erhöhung der Servicefreundlichkeit solcher Anlagen nieder.

2.1.4. Verdrahtete oder progammierbare Steuerung

Bisher haben wir vorausgesetzt, daß die Steuereinrichtung durch einen Rechner realisiert wird, wie es auch dem Anliegen dieses Buches entspricht. Trotzdem sollen im folgenden alternative Lösungen betrachtet werden, um zu erkennen, unter welchen Bedingungen ein Einsatz von Rechnern zweckmäßig und möglich ist.

Betrachten wir zunächst ein einfaches Beispiel für eine Steuerungsaufgabe entsprechend Bild 2.9: Der Antrieb einer Bearbeitungsmaschine soll von einem Bediener nur unter der Bedingung manuell einschaltbar sein ($A_1 = 1$), daß zuvor eine Schutzvorrichtung den Arbeitsraum absichert. Eine Lichtschranke dient zur Signalisierung der Lage dieser Schutzvorrichtung. Ist die Schutzvorrichtung eingeschaltet, so ergibt sich das Signal $E_2 = 1$ der Lichtschranke. Das Einschalten der Maschine ($E_1 = 1$) ohne ordnungsgemäße Sicherung wird dem Bediener als Fehlbedienung mit Hilfe eines Warnsignals ($A_2 = 1$) angezeigt. Die Steuerung muß in diesem Fall verhindern, daß der Motor arbeitet ($A_1 = 0$).



Schalttabelle

Bedien-signal (Schalter) E_1	Maß-signal (Lichtschranke) E_2	Stell-signal (Motor) A_1	Anzeige-signal (Lampe) A_2
0	1	0	0
0	0	0	0
1	1	1	0
1	0	0	1

logische Gleichungen

$A_1 = E_1 \cdot \overline{E_2}$

$A_2 = E_1 \cdot \overline{E_2}$

Bild 2.9. Beispiel einer Steuerungsaufgabe

Der Zusammenhang zwischen den Ausgangs- und Eingangsgrößen der zu realisierenden Steuereinrichtung kann in diesem Fall durch eine Schalttabelle angegeben bzw. durch kombinatorische logische Funktionen analytisch beschrieben werden (Bild 2.9).

Dieses Beispiel steht für eine große Klasse von Steuerungsaufgaben, bei denen die Zustände der Ausgangsgrößen zu jedem Zeitpunkt allein von den Zuständen am Eingang zum gleichen Zeitpunkt abhängen, also die vorhergehenden Situationen keinen Einfluß auf den weiteren Ablauf haben. Zur Realisierung solcher Steuereinrichtungen bietet die Elektronik folgende Möglichkeiten:

Verwendung von Logikbausteinen

Von der elektronischen Industrie werden seit langem (noch vor der Herstellung integrierter Schaltkreise) Sätze von Logik-Grundbausteinen (UND-, ODER-Bausteine, Negatoren, Flipflops usw.) zum Aufbau von Steuerungen angeboten. Der Entwurf einer Steuerung mit diesen Bausteinen erfordert folgende Schritte:

– Ermitteln der zu realisierenden logischen Funktionen $A_i = f(E_1, E_2 \cdots, E_n)$ zwischen den Ausgangs- und Eingangsgrößen aus der vorgegebenen Aufgabenstellung,

– Entwicklung der elektrischen Schaltung für diese Funktionen auf der Basis des verfügbaren Sortiments an Grundbausteinen,
– Herstellung und Bestückung der Leiterplatten.

Bild 2.10a zeigt die Schaltung für das betrachtete Beispiel. Die Programmierung (Festlegung der Funktion) der Steuereinrichtung erfolgt bei dieser Methode durch die Art der Verdrahtung der Logikbausteine. Mit den gleichen Bausteinen würden sich bei Änderung ihrer gegenseitigen Verdrahtung noch einige andere Schaltfunktionen realisieren lassen.

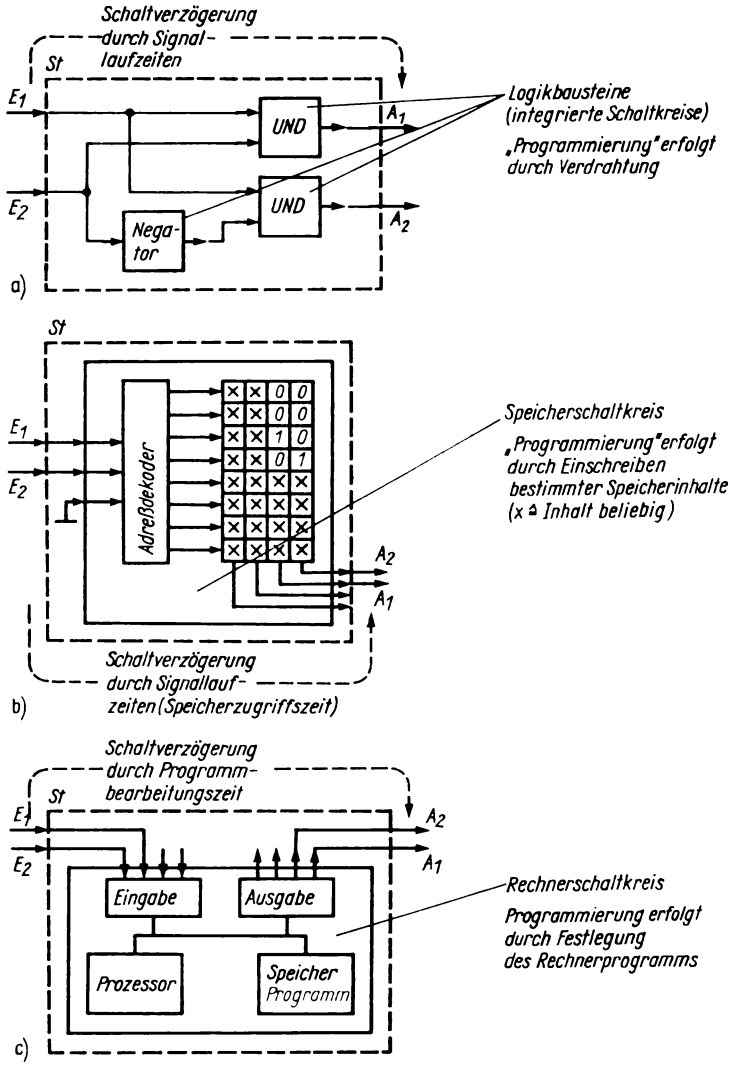


Bild 2.10. Realisierungsvarianten für elektronische Steuereinrichtungen

a) Verwendung von Logikbausteinen
b) Einsatz von Speicherbausteinen
c) Einsatz eines Rechners

Daher wird diese Steuerungsvariante als **verdrahtete Steuerung** oder als reine Hardwarelösung bezeichnet. Sie zeichnet sich durch schnelle Arbeitsweise (die Schaltverzögerung zwischen Änderungen am Eingang und Ausgang wird durch die Summe der Schaltverzögerungen der zu „überwindenden" Logikbausteine bestimmt; je nach verwendeter Bausteinfamilie liegt sie etwa im Bereich zwischen 10···150 ns), aber auch durch eine funktionelle Starrheit aus (eine geringfügige Änderung der Aufgabenstellung erfordert einen neuen Entwurf mit allen o. g. Schritten).

Verwendung von Speicherbausteinen

Bei elektronischen Speicherschaltkreisen kann durch Anlegen bestimmter Eingangssignale (Adresse) der Inhalt einer Zeile von Speicherplätzen beliebig oft ausgelesen werden. Verwendet man folglich die Eingangssignale der Steuerungseinrichtung als Adreßsignale für einen Speicherbaustein und sorgt dafür, daß auf der adressierten Zeile genau der Inhalt steht, der entsprechend der vorgegebenen Schalttabelle an Ausgangssignalen benötigt wird, dann läßt sich die Steuerungsaufgabe ebenfalls lösen (Bild 2.10b).

Die Schaltverzögerung ist bei diesem Verfahren durch die sog. Zugriffszeit des Speicherbausteins (je nach verwendeter Halbleitertechnologie etwa im Bereich zwischen 50···500 ns) bedingt. Der Vorteil dieser Methode ist, daß die Hardware (Verdrahtung des Speicherbausteins auf einer Leiterplatte) unabhängig von der zu leistenden Funktion ist, denn die Funktion wird nur durch den Inhalt des Speicherbausteins festgelegt.

Allein durch steckbare Ausführung des Speicherbausteins auf der Leiterplatte können die verschiedensten Steuerungsaufgaben gelöst werden (indem unterschiedlich programmierte Speicherbausteine nacheinander aufgesteckt werden). Speicherbausteine werden sowohl als Festwertspeicher (ROM-Bausteine) produziert und damit bereits beim Hersteller auf Kundenwunsch mit einem vorgegebenen Inhalt versehen oder als programmierbare Speicher (PROM-, EPROM-Bausteine) angeboten, die erst vom Anwender entsprechend der gewünschten Funktion beschrieben werden müssen. Realisierungen der letzteren Art werden auch als Firmwarelösungen bezeichnet.

Verwendung von Rechnern (Rechnerbausteinen)

Diese Lösung verwendet weitgehend vorgefertigte Hardware in Form von arbeitsfähigen Rechnern (z. B. Einchiprechnern). Die Funktion wird durch Programme festgelegt (programmierbare Steuerung).

Die Programme müssen in dem betrachteten Beispiel folgende Funktionen ausführen (Bild 2.10c):

- zyklisches Abtasten der Eingangsgrößen bzw. Erkennen von Änderungen am Eingang
- Berechnung der aktuellen Ausgangsgrößen
- Ausgabe dieser Resultate.

Der Vorteil dieser Methode besteht darin, daß mit gleicher Hardware eine (auch im Verhältnis zur Anwendung von Speicherbausteinen) sehr große Anzahl unterschiedlicher Steuerungsaufgaben allein durch Veränderung des Programms (oder Teilen davon) realisierbar wird. Dieses hohe Maß an Flexibilität ist aber mit einem entscheidenden Nachteil verbunden: Die Schaltverzögerung ist in diesem Fall durch die Programmlaufzeiten bedingt und liegt damit merklich höher (etwa um 2 Zehnerpotenzen) als bei der reinen Hardwarelösung.

Vergleicht man diese 3 Varianten, so läßt sich zusammenfassend feststellen:

Die Anpassungsfähigkeit der programmierbaren Steuerungen an unterschiedliche Aufgaben und die damit verbundene Senkung des Aufwandes beim Entwurf sprechen zugunsten des Rechnereinsatzes. Die Einsatzgrenze für Rechner wird

damit immer seltener durch ökonomische Faktoren als vielmehr durch die Tatsache ihrer gegenüber einer Hardwarelösung langsameren Arbeitsweise bestimmt. Für solche Anwendungen, die sehr hohe Forderungen hinsichtlich der Reaktionsgeschwindigkeit der Steuereinrichtung stellen, wird auch zukünftig der Aufbau verdrahteter Steuerungen notwendig sein.

2.2. Aufbau und Arbeitsweise eines Rechners

Die Bezeichnung Rechner (Computer) ist dem ursprünglichen Anwendungsgebiet dieser Anlagen entlehnt, nämlich der Ausführung wissenschaftlich-technischer Berechnungen. Für das Verstehen des Aufbaus und der Arbeitsweise ist es dagegen zweckmäßig, zunächst keine Assoziationen zu solchen Anwendungen herzustellen oder gar mit dem Rechnerbegriff die Vorstellung von elektronischen Tisch- und Taschenrechnern zu verbinden.

Als Rechner wollen wir vielmehr programmgesteuerte elektronische Einrichtungen bezeichnen, die Informationen verarbeiten, speichern, an die Umwelt ausgeben und von dieser aufnehmen können.

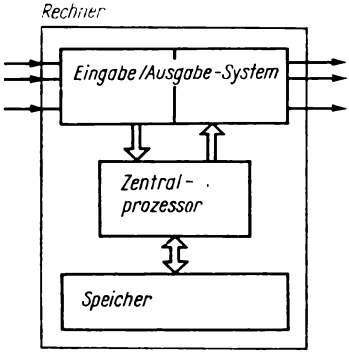


Bild 2.11. Funktionseinheiten eines Rechners

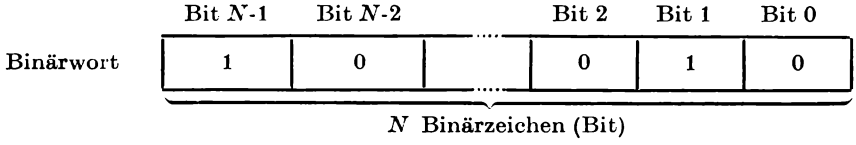
Dementsprechend besteht ein Rechner aus folgenden Funktionseinheiten (Bild 2.11):

- **Zentralprozessor** (Verarbeitung der Information, Steuerung der Programmabarbeitung)
- **Speicher** (Speicherung der Information, Speicherung des Programms)
- **Eingabe/Ausgabe-System** (Informationsaustausch mit der Umwelt).

Genauer müssen wir dabei noch klären, was in diesem Fall unter Information zu verstehen ist.

2.2.1. Informationsdarstellung in Rechnern

Informationen können rechnerintern nur durch **Binärwörter endlicher Breite** (Datenwort, Daten) dargestellt und verarbeitet werden. Vereinfachend nehmen wir zunächst an, daß die Wortbreite für einen bestimmten Rechnertyp konstant ist, also die Information immer in folgender Form transportiert, verarbeitet und gespeichert wird:



Jede Wortstelle besteht aus einem Binärzeichen (Bit), das die Werte 0 oder 1 annehmen kann. (Diese Informationsdarstellung ist durch die Tatsache bedingt, daß Rechner bisher ausschließlich mit Hilfe solcher elektronischer Grundbausteine realisiert werden, die zwei verschiedene elektrische Zustände einnehmen können).

Bei einer Wortbreite von N bit lassen sich insgesamt 2^N verschiedene Binärwörter bilden, also 2^N verschiedene Zustände darstellen.

Betrachten wir als Beispiel den Fall $N = 3$. Die 8 verschiedenen Binärwörter lauten:

000, 001, 010, 100, 011, 110, 101, 111.

Diesen Wörtern kann nun eine beliebige Bedeutung zugeordnet werden. So könnten diese 8 Wörter zum Beispiel benutzt werden, um 8 Zahlenwerte (0, 1, 2, ···, 7 oder −4, −3, ···, 0, ···, +3), 8 Buchstaben (A, B, C, ···, H), 8 verschiedene Farben eines Gegenstandes oder 8 Steueranweisungen für ein Gerät (Motor ein, Heizung ein, . . .) usw. zu verschlüsseln.

Die Festlegung der jeweils gültigen Zuordnung (Verschlüsselung) wird als **Kode** bezeichnet. Damit können wir den Rechnerbegriff wie folgt konkreter fassen:

> Ein Rechner ist eine programmgesteuerte elektronische Einrichtung zur Verarbeitung, Speicherung und Eingabe/Ausgabe von Binärwörtern endlicher (meist konstanter) Breite, denen eine beliebige Bedeutung zugeordnet werden kann. (Die Zuordnung von Zahlenwerten ist dabei ein Spezialfall.)

Die **Wortbreite** ist somit eine wichtige Kenngröße eines Rechners. Sie ist bei den einzelnen Rechnertypen verschieden. Die Rechenanlagen der 3. Generation (ESER-Reihe) weisen z. B. eine Standard-Wortbreite von 32 bit, Prozeß- und Minirechner meist 16 bit auf. Die Mikrorechner werden inzwischen mit Wortbreiten von 4, 8, 16 und 32 bit produziert, wobei der Schwerpunkt der Fertigung bei 8- und 16-bit-Prozessoren liegt.

Wir wollen uns deshalb der Frage zuwenden, wie muß oder sollte die Wortbreite eines Rechners gewählt werden. Die folgende Tabelle zeigt anhand einiger Zahlenpaare, wie durch Verdoppelung der Wortbreite die Anzahl der kodierbaren Zustände exponentiell wächst:

Wortbreite N	Anzahl der Binärwörter 2^N
4	16
8	256
16	65536
32	4294367296

Es ist zunächst naheliegend anzunehmen, daß die Wortbreite entsprechend der Anzahl der Zustände gewählt wird, die im jeweiligen Anwendungsfall unterschieden werden müssen.

Betrachten wir einige *Beispiele*:

1. Um die Dezimalziffern 0···9 zu kodieren, reicht bereits eine Wortbreite von 4 bit. Die Kodetabelle könnte wie folgt lauten:

0	0000	Dabei bleiben 6 Binärwörter übrig, die in diesem
1	0001	Fall nicht verwendet werden:
2	0010	1010
3	0011	1011
4	0100	1100
5	0101	1101
6	0110	1110
7	0111	1111
8	1000	
9	1001	

Diese Zuordnung wird oft angewendet und als BCD-Kode bezeichnet.

2. Die Kodierung eines kompletten Zeichensatzes für Textausgabe (Groß- und Kleinbuchstaben, Ziffern, Sonderzeichen) erfordert bereits eine Wortbreite von 7 bit. Hierfür ist eine Kodetabelle (ISO-7-bit-Kode) international standardisiert worden.

3. Sollen rationale Zahlen rechnerintern kodiert werden, dann sind mit wachsendem Zahlenbereich sehr schnell große Wortbreiten notwendig. Die Darstellung aller sechsstelligen positiven und negativen Zahlen im Festkommaformat (Komma beliebig, aber fest vereinbart) erfordert z. B. die Unterscheidung von 1 999 999 Zahlen (Zuständen) und damit eine Wortbreite von 21 bit.

Aus den Beispielen ist ersichtlich, daß bei Anwendung der Rechner zur Zahlenverarbeitung im Interesse eines großen darstellbaren Zahlenbereichs (und damit einer hohen Rechengenauigkeit) eine möglichst große Wortbreite zu realisieren ist.

Dem steht entgegen, daß die Wortbreite entscheidend den technischen Aufwand und damit die Kosten eines Rechners bestimmt. Elektronische Speicherzellen können nur ein Bit speichern, die Logikbausteine verarbeiten nur einstellige Binärwörter, so daß alle diese Baustufen N-fach parallel angeordnet werden müssen. Als Transportwege zwischen den Baugruppen werden Bündel von jeweils N Leiterbahnen erforderlich.

Es ist folglich beim Entwurf eines Rechners bzw. seines Zentralprozessors notwendig, einen Kompromiß zwischen Wortbreite und Leistungsfähigkeit zur Zahlenverarbeitung zu schließen. Dies führt oft dazu, daß die durch die Wortbreite gegebene Anzahl unterschiedlicher Wörter nicht ausreicht, um die für den jeweiligen Anwendungsfall benötigten Zustände zu kodieren. Insbesondere gilt das bei Zahlenrechnungen. Der Ausweg kann nur darin bestehen, daß mehrere Binärwörter zur Informationsdarstellung herangezogen werden. Da aber Rechner meist nur Binärwörter in ihrer Standardwortbreite in einem Schritt verarbeiten, transportieren und speichern können, müssen dann diese Vorgänge in zeitlich aufeinanderfolgenden Schritten ausgeführt werden. Das heißt, anstelle eines Befehls sind bereits Programme notwendig, um Elementaroperationen zu realisieren. Folglich werden sich die Ausführungszeiten erheblich vergrößern. Es gilt also zu vermerken:

> Trotz einer endlichen Wortbreite lassen sich rechnerintern beliebig viele Zustände (bzw. ein beliebig großer Zahlenbereich) darstellen und verarbeiten. Überschreitet die Anzahl der Zustände aber den durch die Wortbreite des Rechners vorgegebenen Bereich, dann geht dies zu Lasten der Arbeitsgeschwindigkeit des Rechners.

Durch eine „geschickte" Kodierung der Zahlen kann die Arbeitsgeschwindigkeit in bestimmtem Maße beeinflußt werden. In der rechentechnischen Literatur wird daher dieser Problematik gewöhnlich breiter Raum gewidmet. Beim Rechnereinsatz für Steuerungsaufgaben spielen dagegen Rechenprozesse meist eine untergeordnete Rolle. Es soll deshalb nur die einfachste Möglichkeit der Kodierung von Dezimalzahlen und zwar die Kodierung als Dualzahl ohne Vorzeichen, betrachtet werden:

In diesem Fall wird jeder Bitstelle des Datenwortes wie folgt eine Wertigkeit mit aufsteigenden Potenzen zur Basis 2 zugeordnet:

	Bit	N-1	N-2	...	2	1	0
Binärwort		1	0	...	1	1	0
	Wertigkeit	2^{N-1}	2^{N-2}	...	2^2	2^1	2^0

Folglich lassen sich mit dieser Kodierung die natürlichen Zahlen im Bereich von $0 \cdots 2^N - 1$ darstellen.

Betrachten wir einige *Kodierbeispiele* für die bei Mikrorechnern oft verwendete Wortbreite von 8 bit: Der darstellbare Zahlenbereich reicht dementsprechend von 0···255, und es gelten die folgenden Zuordnungen:

1	0	1	1	0	0	1	1	
128		+32	+16			+2	+1	= 179

0	1	0	0	1	1	0	0	
	64			+8	+4			= 76

1	1	1	1	1	1	1	1	
128	+64	+32	+16	+8	+4	+2	+1	= 255.

2.2.2. Befehlszyklus

Von den im Bild 2.11 dargestellten Funktionseinheiten ist der Zentralprozessor der eigentliche Kern, der aktive Teil des Rechners.

> Als **Zentralprozessor** (ZVE zentrale Verarbeitungseinheit oder engl. CPU central processing unit) wird die Funktionseinheit eines Rechners bezeichnet, die, von einem Grundtakt gesteuert, dafür sorgt, daß in endloser Folge Befehle ausgeführt werden.

Jeder Prozessor besitzt dabei einen bestimmten **Befehlssatz,** d. h. eine Menge von Befehlswörtern, die er „versteht" und ausführen kann.

Mit Anlegen des Grundtaktes beginnt der Prozessor seine Arbeit und durchläuft fortlaufend sog. Befehlszyklen. Unter einem **Befehlszyklus (Befehlsschleife)** wird der Zeitabschnitt zur Ausführung eines Befehls verstanden. Während dieser Zeit muß der Prozessor die folgenden Aufgaben erfüllen (Bild 2.12):

1. **Holen des Befehls.** Der Prozessor verfügt über einen Speicherplatz für jeweils einen Befehl, das **Befehlsregister.** In dieses Befehlsregister muß daher zuerst ein Befehl aus dem Programmspeicher geladen werden. Der Prozessor startet dazu einen Lesevorgang mit dem Programmspeicher und erhält den in diesem Befehlszyklus auszuführenden Befehl übermittelt.
2. **Ausführung des Befehls.** Der Inhalt des Befehlsregisters wird anschließend vom Prozessor analysiert (dekodiert) und eine für den speziellen Befehl typische Folge von elementaren Ausführungsschritten vollzogen. In Abhängigkeit vom Befehlswort sind dazu entweder nur Aktionen innerhalb des Prozessors oder aber Transportvorgänge (Kommunikationen) mit der Speicherbaugruppe (Lesen oder Schreiben) bzw. mit den Eingabe/Ausgabe-Kanälen notwendig. Die Dauer dieser Zeitphase kann daher unterschiedlich lang sein.
3. **Bestimmung der Folgebefehlsadresse.** Bevor ein weiterer Befehlszyklus gestartet werden kann, ist es erforderlich, die entsprechende Adresse für den Programmspeicher zu ermitteln. Naheliegend wäre anzunehmen, daß die Befehle im Programmspeicher lückenlos, in aufeinanderfolgenden Speicherplätzen abgespeichert sind. Dann müßte die Speicheradresse einfach weitergezählt (erhöht) werden. Dieser „Mechanismus" allein reicht nicht aus, denn dann

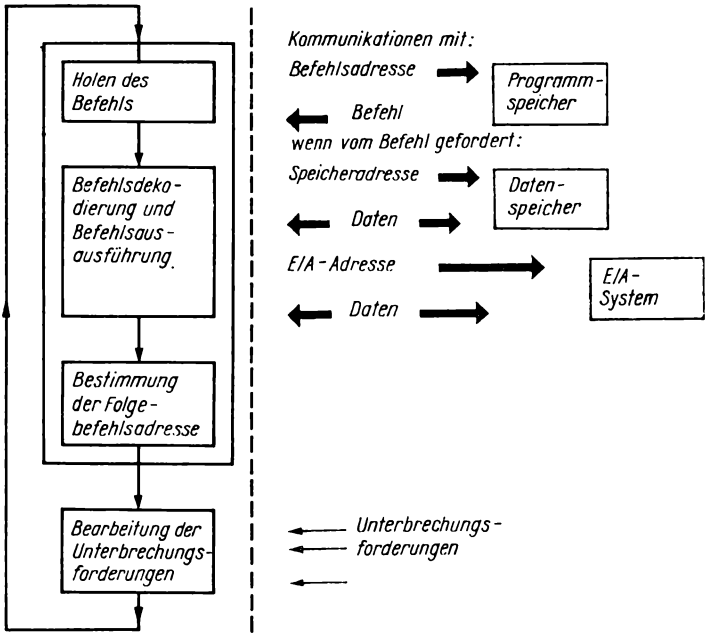


Bild 2.12. Befehlszyklus und Kommunikationsarten in den Phasen des Befehlszyklus

würde sich das Arbeitsprinzip der Rechner nicht von den schon seit Jahrhunderten bekannten Automaten unterscheiden, die durch ein sequentielles Programm (damals gespeichert auf rotierenden nockenbehafteten Walzen u. ä.) gesteuert wurden. Die neue Qualität der Rechner als programmgesteuerte Automaten besteht jedoch gerade darin, daß sie fähig sind, Programmverzweigungen auszuführen. Das heißt, in Abhängigkeit von zuvor erhaltenen Verarbeitungsergebnissen (Resultat Null, Überlauf des Wertebereichs usw.) ist eine alternative Programmfortsetzung möglich, indem durch spezielle Befehle **(Sprungbefehle)** das Programm vom Programmierer in mindestens 2 Richtungen verzweigt werden kann. Die Wirkung dieser Befehle besteht darin, daß bei Erfüllung der Sprungbedingungen durch die zuvor erhaltenen Resultate nicht die Befehlsadresse weitergezählt wird, sondern eine im Befehlswort angegebene neue Adresse übernommen wird. Diese bildet die Basis für das Weiterzählen in den folgenden Befehlszyklen, solange, bis ein weiterer Sprungbefehl im Programm vorkommt.
Da die Sprungadresse auch eine zurückliegende Befehlsadresse sein kann, lassen sich auf diese Weise auch Programmschleifen programmieren.

4. **Prüfung auf Unterbrechungsforderungen.** Mit Hilfe der Sprungbefehle sind nur Programmverzweigungen programmierbar, die zu einem determinierten Zeitpunkt erfolgen, nämlich zu dem Zeitpunkt, an dem der Prozessor bei der sequentiellen Programmabarbeitung bei diesem Sprungbefehl angelangt ist. Bei vielen Echtzeitanwendungen ist es jedoch erforderlich, möglichst schnell auf bestimmte Ereignisse zu reagieren, die sich in der Umwelt des Prozessors zu zufälligen Zeitpunkten ereignen können. Es muß also bei Eintreten einer solchen Situation ein „Sprung" zu einem Programmabschnitt erfolgen, der die dann notwendige Reaktion des Rechners bewirkt. Dies bedeutet, daß eine

Unterbrechung (interrupt) des gerade in Abarbeitung befindlichen Programmteils vorgenommen werden muß. Aufgrund des Zufallcharakters kann diese Art der Programmverzweigung nicht im Programm vorausgeplant werden, sie muß vielmehr durch eine spezielle Hardwareeinrichtung erkannt und erzwungen werden. Diese Baugruppe eines Rechners wird als **Unterbrechungssystem** bezeichnet und ist (zumindest teilweise) Bestandteil jedes Zentralprozessors.

Selbstverständlich kann eine auftretende Unterbrechungsforderung (Unterbrechungssignal) nicht sofort die Arbeit des Prozessors abbrechen. Nachdem aber der laufende Befehl abgearbeitet und die neue Befehlsfolgeadresse ermittelt worden ist, wird vor dem Start eines neuen Zyklus immer erst geprüft, ob in der Zwischenzeit Unterbrechungsforderungen aufgetreten sind (Bild 2.12). Ist dies der Fall, dann wird die eigentliche Folgeadresse „gerettet" (auf speziell dafür vorgesehene Speicherplätze) und durch das Unterbrechungssystem die für die konkrete Unterbrechungsursache vorgesehene Startadresse des **Unterbrechungsbehandlungsprogramms (interrupt-service-routine)** bereitgestellt. Auf diese Weise wird gewährleistet, daß nach Abarbeitung dieses Unterbrechungsbehandlungsprogramms die ursprüngliche Folgebefehlsadresse zurückgeholt und das unterbrochene Programm ohne Informationsverlust (allerdings mit Zeitverzug) fortgesetzt werden kann.

Es gilt also zu unterscheiden: Die sequentielle Abarbeitungsfolge eines Programms kann verlassen werden durch:

- Sprungbefehle und
- Unterbrechungsforderungen.

Die Wirkung ist in beiden Fällen gleich: Die Fortsetzung des Programms erfolgt nicht mit dem Folgebefehl des Programmspeichers, sondern an einer vom Programmierer festgelegten anderen Stelle.

Das Wesen ist dabei aber grundverschieden: Durch einen **Sprung** wird eine determinierte Verzweigung im Programm geschaffen, um auf bestimmte Ereignisse, die bei der Programmausführung eingetreten sind, zu reagieren. Die **Unterbrechung** ist dagegen eine Programmverzweigung aufgrund zufälliger Ereignisse, die unabhängig vom gerade in Arbeit befindlichen Programmabschnitt eingetreten sind, aber einer sofortigen Reaktion durch den Rechner bedürfen.

Die Unterbrechungsfähigkeit ist ein entscheidendes Instrument bei der Echtzeitverarbeitung. Die Leistungsfähigkeit des Unterbrechungssystems (z. B. die erforderliche Umschaltzeit) ist deshalb eine der wichtigsten Kenngrößen für Rechner dieser Einsatzklasse.

Kehren wir nun noch einmal zurück zum Befehlszyklus mit den vier Aufgaben: Befehl holen, Befehl ausführen, Folgebefehlsadresse ermitteln, Unterbrechungen prüfen. Den Antrieb für diesen sich fortlaufend wiederholenden Zyklus bildet der dem Zentralprozessor zugeführte **Grundtakt.** Für jede Aufgabe wird eine Anzahl solcher Taktperioden benötigt. Die Phase der Befehlsausführung erfordert bei den meisten Prozessoren eine unterschiedliche Taktanzahl entsprechend dem Umfang der zu leistenden Arbeit. Folglich gilt, daß die Befehlsausführungszeiten zwar verschieden lang sind, aber stets ein ganzzahliges Vielfaches dieser Grundtaktperiode betragen (Bild 2.13). Der Grundtakt bildet somit ein synchrones Zeitraster, nach dem nicht nur alle internen Transport- und Verarbeitungsschritte, sondern auch die Eingabe/Ausgabe-Vorgänge zur Umwelt ablaufen.

Dieser Takt bestimmt zugleich die Arbeitsgeschwindigkeit des Rechners und wird folglich so schnell wie möglich gewählt. Aufgrund der inneren Verzögerungszeiten der elektronischen Grundbausteine wird er jedoch begrenzt. Ein Hauptziel bei der Weiterentwicklung der Herstellungsverfahren integrierter Schaltkreise besteht deshalb in der Erhöhung der zulässigen Taktfrequenzen. Gegenwärtig

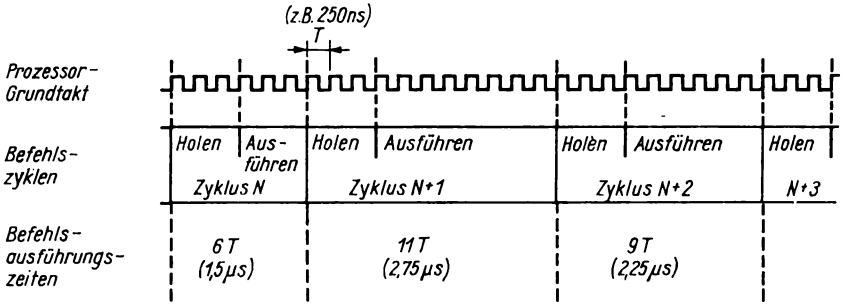


Bild 2.13. Prozessortakt und Befehlszyklus

arbeiten die am häufigsten eingesetzten Mikrorechner- und Mikroprozessorschaltkreise mit Taktfrequenzen bis etwa 10 MHz.

Bei der bisherigen Betrachtung des Befehlszyklus wurde davon ausgegangen, daß alle vier Aufgaben zeitlich nacheinander ausgeführt werden. Eine genauere Analyse ergibt, daß es möglich ist, sowohl innerhalb eines Befehlszyklus bestimmte Vorgänge zeitlich parallel abzuwickeln als auch gewisse Aufgaben der aufeinanderfolgenden Zyklen zu überlappen (pipelining), um die Arbeitsgeschwindigkeit zu erhöhen. Bild 2.14 zeigt, wie noch während der Phase der Befehlsausführung bereits der Folgebefehl geholt wird, der allerdings nur dann zur Ausführung gelangt, wenn keine Unterbrechungsforderungen vorliegen.

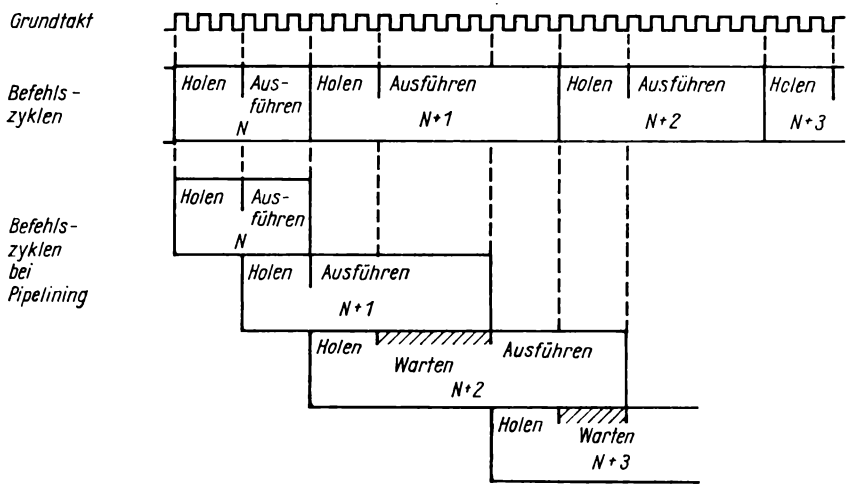


Bild 2.14. Überlappung der Befehlszyklen (pipelining)

2.2.3. Baugruppen eines Rechners

Aus der Aktionsfolge des Befehlszyklus (Bild 2.12) lassen sich unmittelbar die notwendigen Baugruppen eines Prozessors und eines Rechners ableiten. Bild 2.15 zeigt die Blockstruktur, die wir im folgenden schrittweise entwickeln wollen.

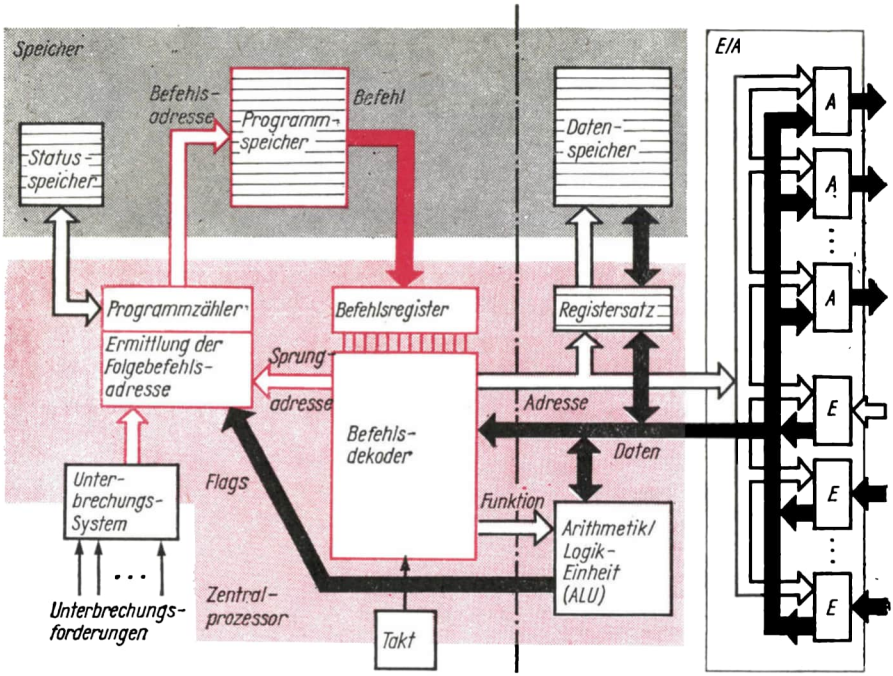


Bild 2.15. Baugruppen eines Rechners

(Dieses Bild ist zugleich eine wichtige Grundlage für die im nächsten Kapitel zu behandelnde Mikrorechner-Hardware.)

Betrachten wir zunächst die in der linken Hälfte angeordneten Baugruppen, die für die fortlaufende Durchführung der Befehlszyklen sorgen: Das **Befehlsregister** bewahrt für die Dauer eines Zyklus den abzuarbeitenden Befehl auf und liefert damit die Eingangsinformation für den **Befehlsdekoder,** der vom Grundtakt gesteuert alle Steuersignale für die übrigen Rechnerbaugruppen erzeugt.

Der Befehlsdekoder ist dementsprechend eine komplexe Baugruppe, durch deren Aufbau der Befehlssatz des Rechners festgelegt ist.

Die für den konkreten Anwendungsfall benötigte Gesamtheit an Befehlen (Programmsystem) wird im **Programmspeicher** aufbewahrt. Am Anfang des Befehlszyklus wird der Befehl aus diesem Speicher gelesen und ins Befehlsregister transportiert. Die Adresse für den Programmspeicher wird im **Befehlsadreßregister** (Programmzähler oder engl. PC programm counter) zuvor bereitgestellt. Der Inhalt des Programmzählers wird danach entweder weitergezählt oder es wird eine neue Folgebefehlsadresse eingetragen, falls ein Sprungbefehl in diesem Zyklus dekodiert worden ist.

Das **Unterbrechungssystem** fängt die Unterbrechungsforderungen auf und bewirkt, nachdem der laufende Befehl ausgeführt worden ist und solche Forderungen vorliegen, daß der Inhalt des Programmzählers gerettet und eine Startadresse zur Unterbrechungsbehandlung eingetragen wird. Die Rettung des Programmzählerinhalts und evtl. weiterer Zustandsinformationen des Zentralprozessors zum Unterbrechungszeitpunkt erfolgt in einem speziellen Speicherbereich, dem **Statusspeicher.** Auf der rechten Seite im Bild 2.15 sind jene Baugruppen dar-

gestellt, die zur Realisierung der eigentlichen Arbeitsaufgaben eines Rechners dienen:

Den Kern bildet die **Arithmetik/Logik-Einheit** (ALU – arithmetic logic unit oder älter: Rechenwerk). Durch Steuersignale vom Befehlsdekoder ausgewählt, wird in dieser Baugruppe jeweils eine arithmetische oder eine logische Operation ausgeführt, mit der 1 oder 2 Operanden zu einem neuen Binärwort verarbeitet werden. Dieser Vorgang erfolgt sehr schnell, meist innerhalb einer Grundtaktperiode. Der Umfang der mit dieser Baugruppe realisierbaren Funktionen bestimmt damit entscheidend die Leistungsfähigkeit eines Rechners. Die Arithmetik/Logik-Einheit ist neben dem Befehlsdekoder eine zweite komplexe Baugruppe. Dementsprechend wird oft zur Aufwandsverringerung an dieser Baugruppe gespart und nur ein Mindestumfang, insbesondere bei den arithmetischen Operationen, vorgesehen. Höhere Arithmetikoperationen müssen dann durch Programme (softwaremäßig) auf die implementierten (hardwaremäßig realisierten) Grundoperationen zurückgeführt werden. Einsparungen an diesem Teil der Hardware führen also zu größerem Programmaufwand und langsamerer Arbeitsweise des Rechners.

Die in der ALU zu verarbeitenden Binärwörter (Operanden, Daten) werden von bestimmten Plätzen des **Datenspeichers** entnommen und das Ergebnis dort wieder abgespeichert. Da für die Transportvorgänge (Kommunikationen) zwischen der ALU und dem Datenspeicher einige Grundtaktperioden nötig sind, wird oft ein kleiner Teil dieses Datenspeichers „in Nähe" der ALU angeordnet und dann als **Registersatz (Registerbank)** bezeichnet. Der Zugriff zu diesen Registern erfolgt sehr schnell und führt zu kürzeren Befehlsausführungszeiten.

Zwischen Datenspeicher/Registersatz und ALU ist folglich ein Transportweg für Daten erforderlich, der auch zu allen E/A-Toren führt. Die E/A-Tore (E/A-Ports) sind die Koppelelemente zwischen Rechner und Umwelt. Ihre Funktion kann am einfachsten wie folgt erklärt werden: E/A-Ports sind Speicherplätze für jeweils ein Wort, mit denen aber im Unterschied zu den Speicherplätzen des Datenspeichers der Prozessor entweder nur Leseoperationen (Eingabeport) oder nur Schreiboperationen (Ausgabeport) ausführen kann. Die jeweilige Gegenfunktion wird dementsprechend von der Umwelt ausgeführt.

Zur Auswahl eines Platzes aus der Gesamtheit ist sowohl für den Datenspeicher und Registersatz als auch für die E/A-Tore die Angabe einer **Adresse** erforderlich. Diese Adreßinformation wird über einen weiteren Transportweg geführt, der vom Befehlsdekoder gespeist wird.

Die Darstellung gemäß Bild 2.15 soll die Gliederung eines Rechners in 2 Hauptteile verdeutlichen: Den steuernden Teil (dazugehörige Baugruppen rot umrandet), der vom Grundtakt angetrieben in synchroner Arbeitsweise laufend Befehlszyklen durchläuft, und den gesteuerten Teil (dazugehörige Baugruppen schwarz umrandet), der entsprechend der aus dem Programmspeicher ausgelesenen Befehlsfolge zur Ausführung bestimmter Verarbeitungs- und Transportoperationen zwischen ALU, Datenspeicher und E/A-Ports veranlaßt wird. Die Kopplung zwischen diesen beiden Teilen erfolgt in erster Linie durch vom Steuerteil ausgehende Signale mit 2 Ausnahmen: Zur Ausführung bedingter Sprünge ist eine Rückkopplung von der rechten Seite zu der Baugruppe, die über die Folgebefehlsadresse entscheidet, notwendig, um die Eigenschaften (engl. flags) des letzten Resultats zu übermitteln. Außerdem muß für den Steuerteil die Möglichkeit bestehen, auf Registerinhalte zurückzugreifen.

Das Bild 2.15 soll in erster Linie die funktionellen Zusammenhänge zwischen den Baugruppen eines Rechners verdeutlichen. Die praktische Realisierung kann davon durchaus abweichen. So werden z. B. die verschiedenen Speicher oft zu einem Speicherblock vereinigt und dementsprechend über einen gemeinsamen Transportweg angeschlossen (s. Abschnitt 3.).

2.2.4. Der Befehlssatz eines Zentralprozessors

Die Menge der Befehle, die ein Rechner ausführen kann, wird durch den Aufbau des Zentralprozessors, insbesondere des Befehlsdekoders und der Arithmetik/Logik-Einheit, festgelegt.

Die Betrachtung der Befehlssätze verschiedener Rechner läßt zunächst vermuten, daß jeder einen individuellen Befehlssatz aufweist. In Wirklichkeit wird diese Vielfalt in hohem Maß durch unterschiedliche Bezeichnungs- und Darstellungsweisen vorgetäuscht. Aufgrund des einheitlichen Arbeitsprinzips und der elektronischen Realisierungsmöglichkeiten existiert ein Elementarbefehlssatz, der in jedem Prozessor nahezu einheitlich implementiert ist. Darüber hinaus können durchaus beträchtliche Unterschiede durch zusätzliche Befehle vorhanden sein, die das Spezifikum eines Prozessors ausmachen und diesen für bestimmte Anwendungen besonders befähigen.

Die genaue Kenntnis des Befehlssatzes ist eigentlich nur für einen kleinen Kreis von Programmentwicklern erforderlich. Aus Effektivitätsgründen wird weitgehend darauf orientiert, den Anwendern höhere Programmiersprachen zur Programmerstellung bereitzustellen. Dazu ist aber zu bemerken: Um ein Programm auf einem Rechner ausführen zu können, muß es aus Befehlen bestehen, die im Befehlssatz des Prozessors enthalten sind. Also muß jedes in höherer Programmiersprache geschriebene Programm durch ein oder mehrere Übersetzungsschritte auf dieses Prozessorniveau zurückgeführt werden. Für das Verständnis der Arbeitsweise und zur Einschätzung der Leistungsfähigkeit eines Rechners ist es deshalb erforderlich, über Grundkenntnisse bezüglich des Charakters solcher Befehlssätze zu verfügen. Erst dann wird verständlich, warum Rechner mit Verarbeitungsgeschwindigkeiten von einigen 100000 Befehlen je Sekunde schließlich doch beträchtliche Rechenzeiten zur Lösung scheinbar einfacher Probleme benötigen.

Wir wollen deshalb im folgenden den prinzipiellen Aufbau eines Befehlssatzes anhand der Befehlstypen kennenlernen und als Beispiel uns einen einfachen Musterbefehlssatz zusammenstellen.

Befehlstypen

Nach der Zweckbestimmung läßt sich der Befehlssatz in folgende Gruppen (Befehlstypen) unterteilen:

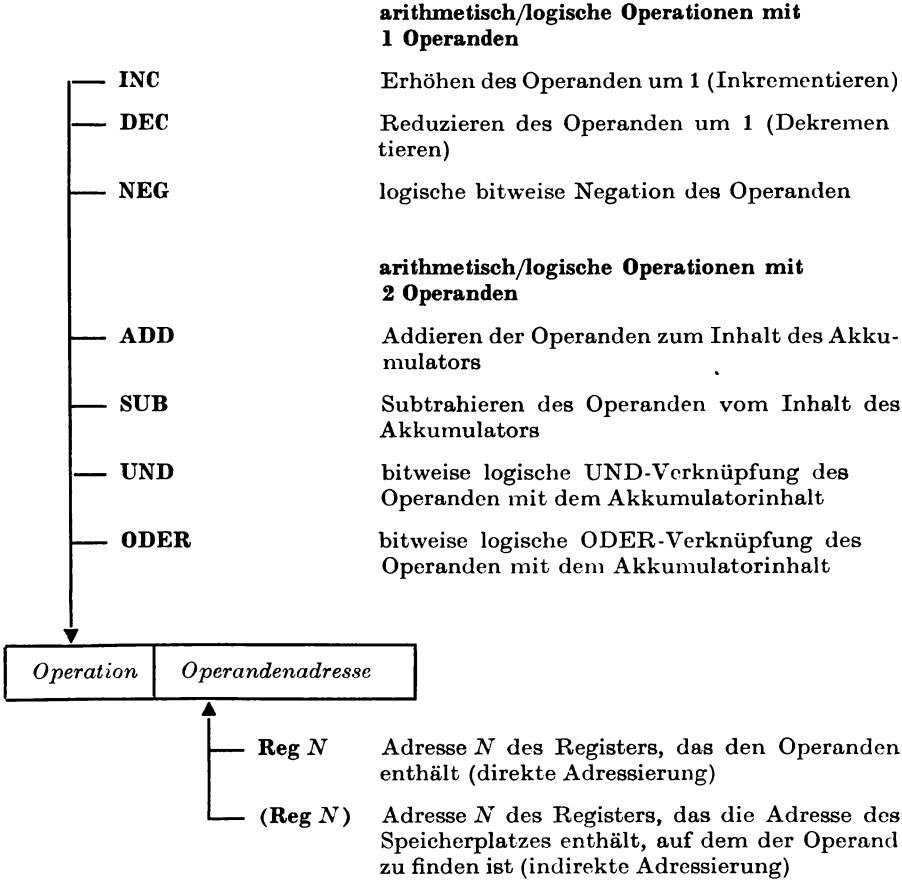
Verarbeitungsbefehle. Sie steuern die Arithmetik/Logik-Einheit und bewirken, daß 1 oder 2 Operanden zu einem neuen Ergebnis verarbeitet werden. Neben der Art der auszuführenden Operation (z. B. Addition, Multiplikation, logische UND-Verknüpfung usw.) müssen im Befehl Angaben über die Speicher- bzw. Registerplätze enthalten sein, von denen die Operanden entnommen werden bzw. auf denen das Ergebnis aufzubewahren ist. Es sind also maximal 3 Adreßangaben erforderlich. Um diese Anzahl zu reduzieren (letztlich die Länge des Befehlswortes zu verringern), werden bestimmte Beschränkungen bezüglich der zulässigen Plätze für die Operanden vorgenommen. Danach können wir die Prozessoren einteilen in:

- **Zweiadreßprozessoren:** Diese Prozessoren speichern das Ergebnis immer auf dem Platz des 1. Operanden. Es sind im Befehl also nur noch 2 Adreßangaben erforderlich, aber der 1. Operand wird überschrieben.
- **Einadreßprozessoren:** Bei diesen Prozessoren ist von vornherein für den 1. Operanden und das Ergebnis ein bestimmter Speicherplatz (Register) festgelegt. Dieses Register wird als Akkumulator (Ergebnisregister) bezeichnet. Im Befehl ist folglich nur noch eine Adresse für den 2. Operanden anzugeben. Notwendig wird aber, vor dem Verarbeitungsbefehl zunächst den 1. Operanden in den Akkumulator zu transportieren.

Die Verarbeitungsbefehle unseres Musterbefehlssatzes sind in Tafel 2.1a zusammengestellt. Dabei wurde vorausgesetzt, daß ein Einadreßprozessor vorliegt (also nur die Registeradresse des 2. Operanden anzugeben ist) und der Zentralprozessor nur mit einem Mindestumfang an Befehlen ausgerüstet ist.

(Zur Darstellung der Befehle in den folgenden Tafeln ist zu bemerken: Rechnerintern werden die Befehle wie alle Informationen durch Binärwörter kodiert. Für die externe Programmbeschreibung ist es jedoch günstiger, sich einer **mnemonischen Darstellung** zu bedienen, d. h. Buchstabenfolgen zu verwenden, aus denen die Funktion des Befehls direkt zum Ausdruck kommt. Dabei ist aber ein Kompromiß zwischen Schreibaufwand und Ausdruckskraft zu schließen. In der Programmierpraxis werden deshalb generell Kurzformen verwendet, denen meist englische Bezeichnungen zugrunde liegen. Da in diesem Buch in erster Linie die wesentlichen Merkmale eines Rechnerbefehlssatzes erfaßt werden sollen, wurde bewußt eine ausführlichere, von der Praxis abweichende Befehlsdarstellung angewendet.)

Tafel 2.1a. Musterbefehlssatz Verarbeitungsbefehle



Transportbefehle. Sie bewirken den Transport von Datenworten zwischen Speicherzellen bzw. Registern und müssen immer 2 Adreßangaben (Quelle und Ziel) enthalten. Zu dieser Befehlsgruppe können auch die Eingabe/Ausgabe-Befehle gezählt werden, die den Transport zwischen Registern und E/A-Toren bewirken. Tafel 2.1b zeigt die in den Musterbefehlssatz aufgenommenen Transportoperationen. Der Ladebefehl bewirkt das Füllen eines Registers oder Speicherplatzes mit einem Binärwort. Da dieses Wort im Befehl enthalten ist, findet damit bei diesem Befehl letztlich ein Transport vom Programmspeicher nach einem Register oder Datenspeicher statt.

Tafel 2.1b. Musterbefehlssatz Transportbefehle

TRANSPORT	nach *ZIELADR*	von *QUELLADR*	**Transportoperation:** Der Inhalt des durch die Quelladresse adressierten Platzes wird zu dem mit Hilfe der Zieladresse bezeichneten Platz transportiert. Nach Ausführung des Befehls steht auf beiden Plätzen der gleiche Inhalt. Quelle und Senke können dabei sowohl Register als auch Speicherplätze sein.
LADE	**Reg** *N*	mit *KONSTANTE*	**Ladeoperation:** Das Register *N* wird mit dem Wort geladen, das für Konstante im Befehl angegeben ist. Im Register kann nur ein Binärwort (8 bit) stehen. Im Befehlswort sollen aber zur Vereinfachung auch Dezimalzahlen angebbar sein, die aber in eine Dualzahl umwandelt werden müssen, bevor das Befehlswort in den Programmspeicher eingegeben werden kann.
EINGABE	**Port** *N*		**Eingabeoperation:** Der Inhalt des Eingabeports *N* wird in den Akkumulator transportiert.
AUSGABE	**Port** *N*		**Ausgabeoperation:** Der Akkumulatorinhalt wird nach dem Ausgabeport *N* transportiert.

Programmverzweigungsbefehle. Sie ermöglichen, durch Angabe einer neuen Folgebefehlsadresse die sequentielle Programmabarbeitung zu beenden und an einer anderen Stelle im Programmspeicher fortzusetzen. Bereits behandelt wurde ein spezieller Befehl, der Sprung (jump).

Je nachdem, ob dieser Sprung ohne Vorbedingungen oder nur bei Eintreten bestimmter Eigenschaften des zuvor ermittelten Ergebnisses erfolgen soll, wird zwischen **unbedingtem** und **bedingtem Sprung** unterschieden. Die Eigenschaften des Ergebnisses müssen zu diesem Zweck im Prozessor gespeichert werden. Dies

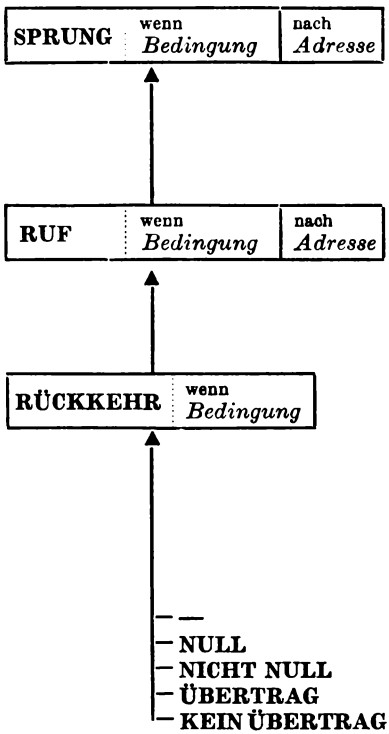
erfolgt in speziellen 1-bit-Speicherzellen, deren Inhalt jeweils eine Eigenschaft signalisiert und die dementsprechend als **Flag-Bits** bezeichnet werden. Typische Beispiele für solche ausgewählten Eigenschaften, die als Flag-Bits gespeichert und für bedingte Sprungbefehle angewendet werden können, sind:

- Ergebnis gleich Null (Null-Flag)
- Ergebnis liegt außerhalb des mit der Wortbreite darstellbaren Zahlenbereichs (Überlauf-Flag).

Tafel 2.1c zeigt die mit Flag-Bits realisierbaren Sprungbefehle.

Der Sprungbefehl erzwingt einen neuen Programmzählerinhalt. Oft stellt sich aber für den Programmierer die Notwendigkeit einer Programmverzweigung der folgenden Art: An einer bestimmten Stelle der sequentiellen Programmabarbeitung soll ein Programmteil eingeblendet werden, das an einer beliebigen Stelle des Programmspeichers steht (Unterprogrammtechnik). Dazu ist zwar ebenfalls ein Sprung notwendig, aber es muß sichergestellt werden, daß nach Abarbeitung dieses Unterprogramms die Fortsetzung an der Absprungstelle erfolgt. Diese Aufgabe wird durch ein Befehlspaar bestehend aus **Rufbefehl** (Call-Befehl, Sprung mit Rückkehrabsicht) und **Rückkehrbefehl** (Return-Befehl) gelöst. Der Rufbefehl entspricht in seiner Wirkung grundsätzlich einem Sprungbefehl, rettet aber vor dem Überschreiben den Inhalt des Programmzählers in den Statusspeicher. Am Ende des einzublendenden Unterprogramms muß ein Rückkehrbefehl stehen, dessen Wirkung allein darin besteht, diesen sichergestellten Inhalt aus dem Statusspeicher wieder in den Programmzähler zu laden und damit das aufrufende Programm an der richtigen Stelle fortzusetzen.

Tafel 2.1c. Musterbefehlssatz Programmverzweigungsbefehle



Sprungbefehl: Bei Erfüllung der Bedingung wird die Adresse in den Programmzähler geladen und damit das Programm mit dem auf dieser Adresse stehenden Befehl fortgesetzt. Bei nichterfüllter Bedingung wird der auf diesem Sprungbefehl im Programmspeicher folgende Befehl abgearbeitet.

Rufbefehl (Sprung mit Rückkehrabsicht, Call-Befehl): Bei erfüllter Bedingung wird die Adresse in den Programmzähler geladen, nachdem dessen Inhalt zuvor in den Statusspeicher gerettet wurde. Ansonsten gleiche Wirkung wie Sprungbefehl.

Rückkehrbefehl (Return-Befehl): Die zuletzt in den Statusspeicher gerettete Adresse wird in den Programmzähler zurückgeholt und damit das Programm an einer der einem Rufbefehl folgenden Stelle fortgesetzt.

Bedingungsarten

— —	unbedingte Verzweigung
NULL	Ergebnis 0
NICHT NULL	Ergebnis ungleich 0
ÜBERTRAG	bei Ergebnis Übertrag entstanden
KEIN ÜBERTRAG	bei Ergebnis kein Übertrag entstanden

Steuerbefehle. Sie dienen zur Beeinflussung der Arbeitsweise des Zentralprozessors, insbesondere zur Modifizierung des Ablaufs des Befehlszyklus. Einige typische Beispiele sind in Tafel 2.1d zusammengestellt. Die Befehle Unterbrechungserlaubnis und Unterbrechungssperre ermöglichen das Einschalten und Ausschalten des Unterbrechungssystems. Damit besteht die Möglichkeit, im Programm festzulegen, ob die folgenden Befehle von Unterbrechungen ungestört abgearbeitet werden können. Der NOP- und der Halt-Befehl verändern den Ablauf des Befehlszyklus. Während der NOP-Befehl einen kompletten Befehlszyklus bewirkt, nur ohne jede Wirkung (allein Zeit wird verbraucht), beginnen nach Erkennen eines Halt-Befehls Befehlszyklen, die nur noch die Erkennung von Unterbrechungsforderungen zur Aufgabe haben. Es erfolgt keine Weiterzählung des Programmzählers, und keine weiteren Befehle werden geholt. Folglich können nur Unterbrechungsforderungen den Prozessor aus diesem Halt-Zustand bringen.

Tafel 2.1d. Musterbefehlssatz Steuerbefehle

EININT	**Interrupterlaubnis**
AUSINT	**Interruptsperre**
HALT	**Halt-Befehl:** Prozessor verbleibt solange in einem Wartezustand, bis Unterbrechungen eintreffen.
NOP	**Befehl ohne Wirkung** (No-Operation)

Aufbau eines Befehlswortes

Je nach Befehlstyp müssen im Befehlswort unterschiedliche Angaben enthalten sein. Mindestens ist jedoch der **Operationskode** notwendig. Er gibt an, welche Funktion auszuführen ist. Werden für diesen Teil des Befehlswortes 8 bit verwendet, so lassen sich 256 verschiedene Funktionen kodieren. Darüber hinaus sind (wie oben beschrieben) bei einigen Befehlstypen Adreßangaben erforderlich (Bild 2.16). Daraus folgt, daß die Befehlsworte unterschiedlich lang sind. Im Widerspruch dazu steht, daß der Programmspeicher aus Plätzen fester Wortbreite besteht, die in der Regel mit der Datenwortbreite übereinstimmt. Die meisten Befehlsworte benötigen dementsprechend mehrere Speicherplätze im

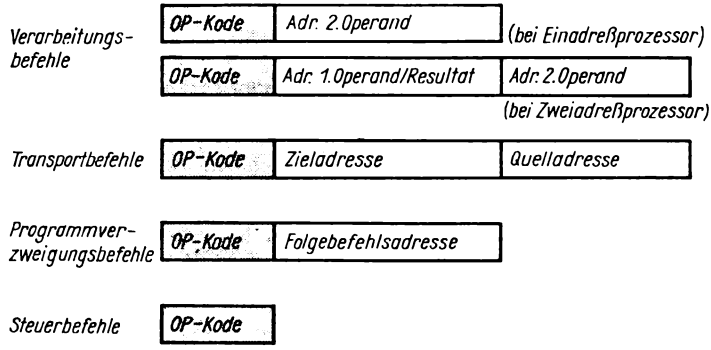


Bild 2.16. Befehlswortformate

Programmspeicher. Das Holen des Befehls am Anfang des Befehlszyklus erfordert daher entsprechend viele Leseoperationen. Die Notwendigkeit dazu wird vom Prozessor anhand des Operationskodes erkannt und gesteuert.

Indirekte Adressierung

Bisher haben wir angenommen, daß die erforderlichen Adreßangaben direkt in den Befehlsworten enthalten sind. Das heißt jedoch, daß der Programmierer diese Adressen nur fest vorgeben könnte. Die Programmierpraxis zeigt jedoch sehr schnell, daß sich wesentlich kürzere Programme schreiben lassen, wenn der Befehlssatz auch solche Befehle enthält, die mit indirekten Adressen arbeiten. Indirekt heißt, daß im Befehlswort anstelle der erforderlichen Adreßangabe nur der Hinweis enthalten ist, in welchem Register die Adresse zu finden ist. Dieses Verfahren bringt den Vorteil, daß die Adressen jetzt Inhalte von Speicherzellen und folglich durch Befehle zu manipulieren sind (durch Transportbefehle transportierbar, durch Verarbeitungsbefehle veränderbar usw.).

(Im Abschnitt 4.1. (Tafel 4.5) wird ein Beispiel für die Anwendung der indirekten Adressierung behandelt.)

Bei der Notierung der Befehle wollen wir die indirekte Adreßangabe durch Verwendung von Klammern zum Ausdruck bringen.

Die Angabe

ADD	89

heißt, daß zum Inhalt des Speicherplatzes (Registers) 89 der Inhalt des Akkumulators addiert werden soll. Dagegen bedeutet

ADD	(89)

daß zum Inhalt des Akkumulators der Inhalt des Speicherplatzes addiert wird, dessen Adresse im Register 89 steht.

2.2.5. Kopplung Rechner–Umwelt

Im Abschnitt 2.1. haben wir gesehen, daß bei Echtzeitsystemen an den Steuerrechner eine Vielzahl von Einrichtungen zur Bereitstellung der Eingangsgrößen (Meß- und Bedieneinrichtungen) und zur Übernahme der Ausgangsgrößen (Stell- und Anzeigeeinrichtungen) anzuschließen sind. Die technische Ausführung solcher Einrichtungen soll nicht Gegenstand dieses Buches sein, aber wir wollen die wichtigsten Probleme ansprechen, die bei der Kopplung zwischen Rechner und Umwelt auftreten:

Der Anschluß systemfremder Einrichtungen an das Eingabe/Ausgabe-System eines Rechners erfordert meist spezielle **Koppelelektronik** mit den im folgenden erklärten Grundfunktionen.

Anpassung der Signalpegel, Störsignalunterdrückung und Schutzmaßnahmen gegen bauelementezerstörende Überlastungen der Rechnereingänge und -ausgänge

Die Eingabe/Ausgabe-Signale eines Rechners sind von elektronischer Natur. Elektronisch anstelle von elektrisch soll dabei deutlich machen, daß die Signale im Bereich von wenigen Volt (meist zwischen 0 und 5 V) liegen und daß sie nur kleine Schaltleistungen übertragen können. Außerdem sind die Eingänge und Ausgänge der integrierten Schaltkreise empfindlich gegen Überlastungen

(Überspannung durch Störsignale), die zur bleibenden Funktionsstörung führen können.

Daraus resultieren die Nachteile elektronischer und mikroelektronischer Realisierungen gegenüber der klassischen Steuerungstechnik auf der Basis von Relais:

- größere Empfindlichkeit gegenüber Störspannungen, die als Folge elektromagnetischer Felder (z. B. durch Schalten starkstromtechnischer Anlagen) induziert werden und die die Signale mit kleinem Pegel leicht verfälschen können,
- Notwendigkeit des Schutzes der Elektronik vor Überspannungen (meist durch sog. Optokoppler realisiert),
- Notwendigkeit der Leistungsverstärkung der Ausgangssignale zur Betätigung der Stell- und Anzeigeeinrichtungen.

Diese Nachteile der Elektronik werden heute deshalb besonders spürbar, da sich die Art und Weise des Steuerungsentwurfs grundlegend verändert hat. In der vorherigen Entwicklungsetappe, in der jede zu realisierende Schaltfunktion eine zusätzliche Leiterplatte beim Aufbau einer verdrahteten Steuerung bedeutete, lag das Augenmerk beim Systementwurf auf einer Begrenzung des informationsverarbeitenden Teils und damit auch auf einer Begrenzung der Anzahl der Eingangs- und Ausgangsgrößen. Beim Mikrorechnereinsatz ergeben sich in dieser Hinsicht wesentlich geringere Beschränkungen, so daß sich als neuer Engpaß die Verfügbarkeit passender Meß- und Stelleinrichtungen und deren Koppelelektronik deutlicher herauskristallisiert. Die Folge ist, daß Platzbedarf und auch Kosten für die Koppeleinrichtungen, obwohl sie letztlich nur unbedeutende Hilfsfunktionen leisten, bei vielen Steuereinrichtungen die des eigentlichen Rechners um ein Mehrfaches übersteigen können.

Signalumwandlung

Als weitere Aufgabe ist die Anpassung der Signale an die Eingabe/Ausgabe-Möglichkeiten durch Signalwandlung vorzunehmen: Die rechnerinterne Informationsdarstellung basiert auf Binärwörtern. Dementsprechend ist auch die wortweise Eingabe/Ausgabe von Binärwörtern die rechnergemäße, also am einfachsten zu realisierende Form. Dem steht aber entgegen, daß die wenigsten Umweltsignale in dieser Form vorliegen. Grundsätzlich können folgende Signalarten unterschieden werden:

- analoge Signale
- binäre Signale
- digitale Signale.

Binäre Signale nehmen nur 2 Zustände ein und dienen folglich zur Übermittlung der Zustände EIN und AUS bei Schaltern, optischen und akustischen Anzeigeeinrichtungen bzw. bei allen Meß- und Stelleinrichtungen mit Zweipunktverhalten (Bild 2.17a). Außerdem tritt diese Signalart bei Takt-, Zähl- und Impulssystemen auf. Die Eingabe/Ausgabe von Binärsignalen erfordert keine Signalwandlung. Es ist nur zu entscheiden, ob einem Binärsignal ein E/A-Port und damit ein komplettes Rechnerwort (von dem dann nur ein Bit benötigt wird) zugeordnet wird oder ob mehrere Binärsignale willkürlich auf einem gemeinsamen E/A-Port zusammengefaßt werden.

Kann die Eingabe/Ausgabe-Einrichtung nur eine endliche Anzahl verschiedener Zustände einnehmen, dann liegt ein digitales System vor (z. B. Tastaturen, Positionsmeßsysteme mit einer endlichen Stufung usw.). Die Übermittlung der Zustandsinformation kann mit Hilfe eines mehrstufigen Signals (Bild 2.17b) erfolgen. Häufiger aber wird gleich die günstigere kodierte Übertragung durch parallele Binärsignale realisiert (Bild 2.17c). Hierbei gilt wiederum, daß durch

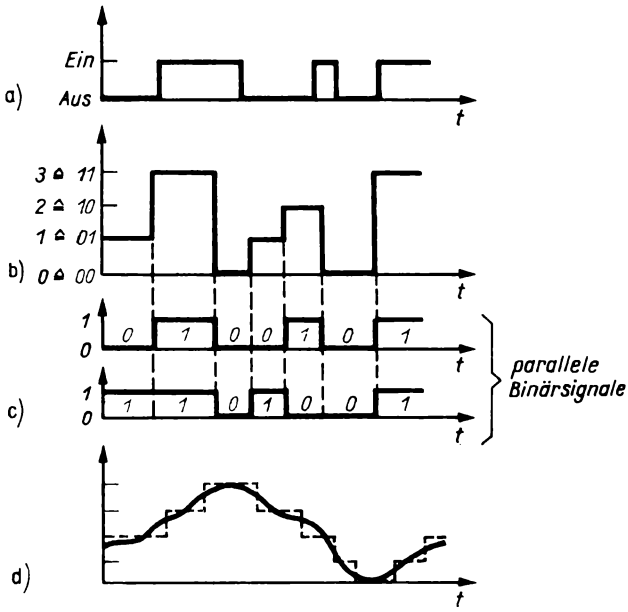


Bild 2.17. Signalarten

a) binäres Signal
b) digitales Signal — mehrstufig
c) digitales Signal — binärkodiert
d) Analogsignal und Approximation durch ein mehrstufiges Signal

n Binärsignale 2^n verschiedene Werte kodiert werden können. Die letztere Form entspricht damit direkt dem Eingabe/Ausgabe-Format des Rechners. Voraussetzung ist, daß n kleiner oder gleich dessen Wortbreite ist. Digitale Signale mit größerer Stufenanzahl sind auf mehrere E/A-Ports aufzuteilen und können folglich nicht mehr durch eine E/A-Operation behandelt werden.

Analoge Signale sind prinzipiell nicht mit der Informationsdarstellung und damit dem Eingabe/Ausgabe-Format von Rechnern verträglich. Ein analoges Signal (Bild 2.17d) kann unendlich viele Werte annehmen und müßte folglich durch ein Binärwort unendlicher Breite kodiert werden. Die Lösung kann nur in einer Informationsreduzierung bestehen, indem das analoge Signal durch ein digitales Signal angenähert wird. Diese Aufgabe wird durch Analog-Digital- bzw. Digital-Analog-Umsetzer realisiert, die inzwischen auch weitgehend als integrierte Schaltkreise verfügbar sind (s. Abschnitt 3.1.).

Anpassung der unterschiedlichen Arbeitsgeschwindigkeiten von Rechnern und Peripherie (Synchronisation)

Als weiteres Problem der Kopplung zwischen Rechner und Umwelt ist die Anpassung der unterschiedlichen Geschwindigkeiten durchzuführen. Aufgrund der seriellen Befehlsabarbeitung ist ein Rechner nicht in der Lage, kontinuierlich Information aufzunehmen bzw. abzugeben. Die Programmlaufzeiten für Eingabe/Ausgabe und für die dazwischen notwendigen Verarbeitungs- und Organisationsaufgaben führen zu einer zeitdiskreten Arbeitsweise, wobei die Zeitabstände zwischen Eingabe- und Ausgabeoperationen durchaus auch in Abhängigkeit von der konkreten Situation in weiten Grenzen schwanken können.

Bei zeitkontinuierlichen Signalen führt das zu einer weiteren Informationsreduzierung, indem aus dem Verlauf nur Proben entnommen bzw. ausgabeseitig nur stufenförmige Signale erzeugt werden können. Genau betrachtet, sind die zeitkontinuierlichen Signaldarstellungen ein zwar anschauliches, aber für diese Betrachtungen ungeeignetes und nicht reales Modell. Kein technisches System kann in unendlich dichter Folge Zustandsänderung erzeugen bzw. mit unendlicher

Geschwindigkeit Änderungen ausführen. Es ist also in jedem Fall nur abzuschätzen, ob die Reaktionsgeschwindigkeit des Rechners im Verhältnis zur Dynamik (Änderungsgeschwindigkeit) der Eingangs- und Ausgangssignale ausreichend ist. Am einfachsten ist dies bei binären bzw. digitalen Signalen zu erkennen. Obwohl auch hier eine zeitkontinuierliche Darstellung üblich ist (vgl. Bild 2.17a), sind doch in Wirklichkeit nur jene diskreten Zeitpunkte von Interesse, bei denen Zustandsänderungen auftreten. Dementsprechend braucht der Rechner auch nur darüber informiert zu werden.

Zum Erkennen von Zustandsänderungen eines Eingangssignals durch den Rechner stehen die beiden folgenden Verfahren zur Auswahl (Bild 2.18):

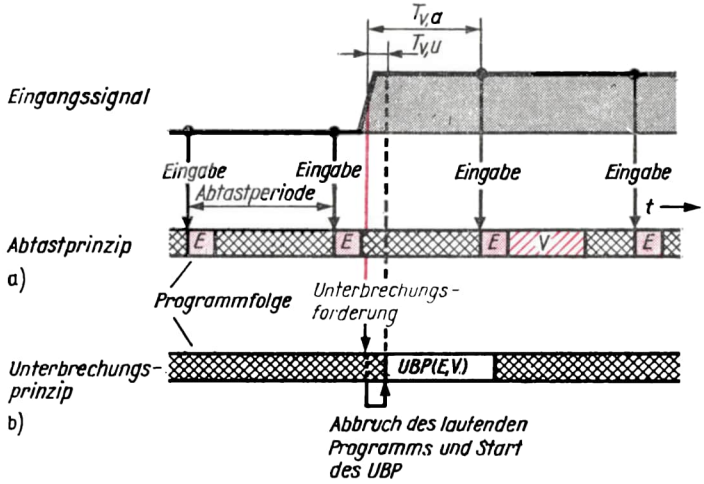


Bild 2.18. Erkennen von Zustandsänderungen an Rechnereingangsleitungen

a) Abfrageprinzip (polling)
b) Unterbrechungsprinzip

E Programm zur Signalabtastung der Eingänge; *V* Verarbeitungsprogramm bei Signaländerung; *UBP* Unterbrechungsbehandlungsprogramm; T_V Verzögerungszeit zwischen Signaländerung und Bearbeitungsbeginn im Rechner

1. **Abfrageprinzip (polling):** Bei diesem Verfahren erfolgt durch den Rechner eine laufende Probenentnahme aus dem Signal, indem zyklisch Eingabeoperationen mit anschließender Analyse (Eingabeprogramm *E* im Bild 2.18a) ausgeführt werden. Der Nachteil dieses Verfahrens ist, daß der Rechner mit dieser Beobachtung ständig belastet wird (Start von *E* auch wenn keine Zustandsänderung aufgetreten ist) und daß die Abtastperiode des Signals für den Anwendungsfall zu groß werden kann, wenn der Rechner zwischendurch noch weitere Aktivitäten ausführen muß (Verzögerungszeit $T_{V,a}$ im Bild 2.18).
2. **Anforderungsprinzip (Unterbrechungsverfahren):** Bei diesem Verfahren wird durch eine einfache elektronische Schaltung gewährleistet, daß jede Zustandsänderung des Eingangssignals ein Unterbrechungssignal für den Rechner erzeugt und diesen damit sofort informiert. Der Rechner unterbricht das laufende Programm und startet das Unterbrechungsbehandlungsprogramm, das Eingabe, Analyse und Verarbeitung des Zustandswechsels bewirkt. Bei diesem Verfahren wird damit gesichert, daß der Rechner nur bei Auftreten eines Zustandswechsels belastet wird und daß die Reaktion schneller erfolgen kann (Zeit $T_{V,u}$ im Bild 2.18).

Voraussetzung für beide Verfahren (und damit für den Rechnereinsatz überhaupt) ist, daß die Signaländerungen im Verhältnis zur Arbeitsgeschwindigkeit des Rechners nur selten erfolgen. (Als Richtwert kann gelten: Die Zeitintervalle müssen um den Faktor 10···100 größer sein als die Befehlsausführungszeiten.)

Bisher haben wir nur die Informationseingabe behandelt. Bei der Ausgabe ist zu unterscheiden zwischen Einrichtungen, die sich dem Zeitregime des Rechners beliebig unterordnen, also zu jedem Zeitpunkt und mit beliebiger Geschwindigkeit Information übernehmen können, und solchen, die den Zeitpunkt der Informationsübernahme und damit auch die Ausgabegeschwindigkeit selbst bestimmen. Im letzteren Fall ist mit dem Ausgabeprozeß immer ein Eingabevorgang gekoppelt, in dem der Rechner durch ein Signal über die **Übernahmebereitschaft** informiert werden muß. Bezüglich der Realisierung dieser Eingabe bestehen wiederum die beiden oben betrachteten Möglichkeiten: Das Bereitschaftssignal kann entweder vom Rechner laufend durch ein Eingabeprogramm beobachtet werden, oder es löst zum Zeitpunkt der Bereitschaft eine Unterbrechung aus.

2.3. Algorithmierung

Der Einsatz eines Rechners setzt voraus, daß die zu lösende Aufgabe in eine Folge solcher Teilfunktionen gegliedert wird, von denen die prinzipielle Ausführbarkeit mit Hilfe eines Rechners bekannt ist. Dieser Vorgang wird als **Algorithmierung** und das Ergebnis als **Algorithmus** bezeichnet. Unter Programmierung versteht man dagegen die Umsetzung eines Algorithmus in eine dem jeweils verwendeten Rechner verständliche Sprache. Der gesamte Entwurfsprozeß läßt sich damit wie folgt gliedern:

1. Fixierung der Aufgabenstellung: Neben einer möglichst umfassenden qualitativen Beschreibung des zu lösenden Problems müssen auch die quantitativen Bedingungen (z. B. kritische Echtzeitforderungen) festgelegt werden.
2. Systementwurf: Der Systementwurf besteht aus dem Entwurf der Hardware und Software. Dabei muß der Hardwareentwurf zuerst erfolgen. Darauf aufbauend ist das Problem in Teilaufgaben zu gliedern (**Strukturierung** der Aufgabe), für die jeweils Algorithmen zu finden sind.
3. Programmierung: In dieser Phase erfolgt die Umsetzung der Algorithmen in Programme. (Diese Phase erfordert nur dann noch wesentliche schöpferische Anteile, wenn es auf extreme Ausnutzung der Hardware ankommt, ansonsten lassen sich diese Aufgaben weitgehend schematisieren und damit durch Rechner ausführen.)

Zu dieser Grobgliederung des Entwurfsprozesses müssen noch 2 Anmerkungen gemacht werden:

– Die ersten beiden Phasen sind im Prinzip noch unabhängig von einem konkreten Rechnertyp ausführbar, d. h., Strukturieren und Algorithmieren setzen zwar die Annahme der prinzipiellen Hardwarekonfiguration und Kenntnisse über die Operationen voraus, die ein Rechner leisten kann, erfordern aber noch keine detaillierten Angaben über den Befehlssatz. Eine solche Aussage wird sicher von Programmierpraktikern nicht ohne Widerspruch hingenommen. Selbstverständlich bewirken Detailkenntnisse über die einsetzbare Hardware (z. B. Wortbreite, Befehlssatz, Speicherkapazität des Rechners) Rückkopplungen auf den Algorithmierungsprozeß und verhindern beispielsweise, daß Algorithmen entwickelt werden, die auf uneffektive Programmlösungen hinauslaufen. Zum anderen ist damit aber auch die Gefahr verbunden, daß von vornherein rechnerspezifische Einschränkungen vorgenommen werden, die eine Umsetzung des Algorithmus in Programme für andere Rechner unmöglich machen.

– Der Entwurfsprozeß wird in der Regel nicht nur einmal von oben nach unten (top-down-Methode) durchlaufen, sondern ist meist ein iterativer Vorgang. So kann ein wiederholter Ansatz für die Hardware und damit auch für die Programmstruktur erforderlich werden, wenn sich mit dem bisher Festgelegten nicht alle Forderungen erfüllen lassen. Als kritisch erweist sich bei Echtzeitsystemen die Einhaltung bestimmter Grenzen für die Programmlaufzeiten. Der Funktionstest offenbart auch häufig noch Entwurfsfehler, die zu einer wiederholten Abarbeitung einer oder mehrerer Entwurfsphasen zwingen.

Die Rationalisierung dieses Programmentwurfs durch Bereitstellung von Entwurfssystematiken, Entwurfssprachen bis hin zur Einbeziehung des Rechners selbst ist heute ein wichtiges Arbeitsgebiet der Rechentechnik. Trotz aller Fortschritte und teilweise auch euphorischen Darstellungen ist einzuschätzen, daß bei diesen Arbeiten auch zukünftig eine wesentliche intuitive und heuristische Komponente erforderlich bleiben wird. Die Darstellung des erreichten Standes auf diesem Gebiet kann nicht Gegenstand dieses Buches sein, zumal sich auch keine Besonderheiten durch die Mikrorechentechnik ergeben. Es sollen im folgenden nur eine Darstellungsform für Algorithmen eingeführt werden, auf die später im Abschnitt 4. Bezug genommen wird, und ein einfaches Entwurfsbeispiel bis zur Phase der Algorithmierung behandelt werden.

2.3.1. Darstellung von Algorithmen durch Ablaufpläne

Algorithmieren heißt, eine Aufgabe oder Teilaufgabe in eine Ablauffolge solcher Aktionen zu zerlegen, von denen bekannt ist, daß sie grundsätzlich durch einen Rechner realisierbar sind.

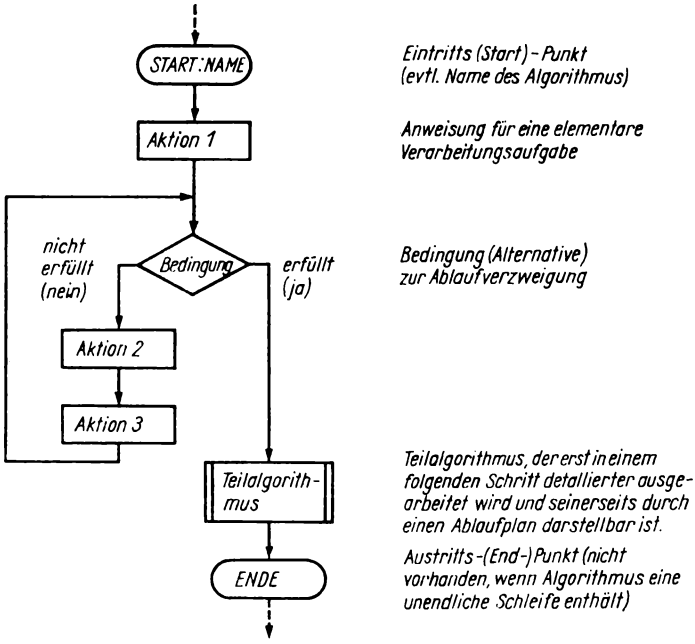


Bild 2.19. Ablaufplan zur Darstellung eines Algorithmus mit Hilfe von Sinnbildern

Zur Darstellung eines Algorithmus existieren verschiedene Verfahren. Am verbreitetsten ist dabei die Darstellung als Ablaufplan (auch: Programmablaufplan (PAP)). Bild 2.19 zeigt ein Beispiel, das alle verwendbaren Sinnbilder enthält und erläutert.

Dabei kann folgender Zusammenhang zwischen dem Sinnbild im Ablaufplan und dem danach daraus entstehenden Rechnerprogramm hergestellt werden: *Aktionen* werden durch einen Befehl oder durch mehrere nacheinander auszuführende Befehle vom Typ Verarbeitungs-, Transport- und Steueroperationen gebildet. *Alternativen* werden durch einen bedingten Sprungbefehl realisiert (oder durch eine Befehlsfolge, die mit einem solchen Befehl abschließt). Die Pfeile auf Folgeaktionen bedeuten eigentlich einen unbedingten Sprungbefehl, der aber immer dann entfallen kann, wenn der erste Befehl der Folgeaktion lückenlos im Programmspeicher an den vorhergehenden Befehl anschließt. [Im Beispiel Bild 2.19 ist damit nur ein solcher unbedingter Sprungbefehl erforderlich (Rücksprung nach Ausführung der Aktion 3).]

Ablaufpläne sind ein an das menschliche Informationseingabesystem angepaßtes Mittel (zumindest solange der Umfang eine Seite nicht übersteigt). Für die Kommunikation mit einem Rechner zum Zweck einer automatischen (oder rechnerunterstützten) Weiterverarbeitung bilden sie dagegen kein geeignetes Mittel. Aus diesem Grund wird zunehmend auf andere Methoden orientiert, z. B. auf die Darstellung eines Algorithmus durch formale Sprachen oder in Tabellenform. Wir werden im Abschnitt 4.2. bei der Erörterung höherer Programmiersprachen darauf zurückkommen.

2.3.2. Einfaches Entwurfsbeispiel

Als Beispiel soll der Entwurf der Steuerung für eine selbsttätige Türöffnung betrachtet werden.

Bild 2.20 zeigt die zu steuernde Einrichtung: Eine Lichtschranke *LS* zeigt das Betreten des Türbereiches an. Die Schiebetür *ST* läßt sich durch einen umsteuerbaren Motor *M* öffnen oder schließen, wobei das Erreichen der Endlagen durch Endschalter *ES 1* und *ES 2* signalisiert wird.

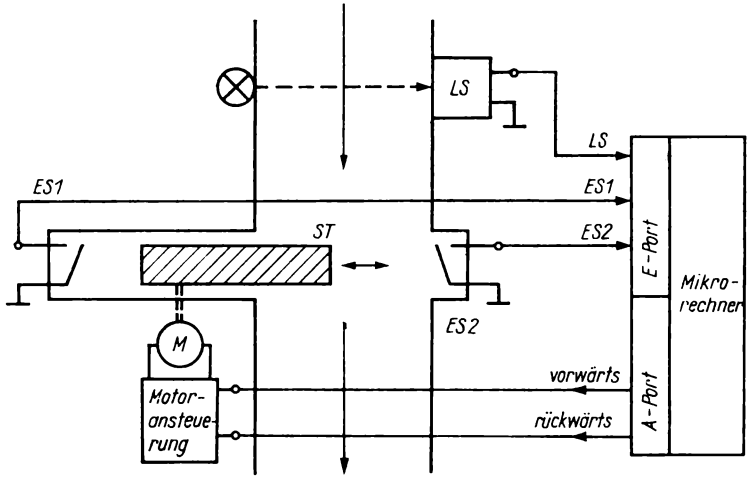


Bild 2.20. Automatische Tür (Hardwarestruktur)
LS Lichtschranke; *ES* Endschalter; *M* Motor; *ST* Schiebetür

Die Steuerung hat damit die folgende **Aufgabe** zu lösen: Bei Aktivierung der Lichtschranke ist die Tür zu öffnen, eine Zeit T_0 offen zu halten und danach zu schließen. Erfolgen während eines solchen Zyklus weitere Aktivierungen der Lichtschranke, dann ist die Öffnungszeit zu verlängern oder, falls bereits der Schließvorgang eingeleitet wurde, muß dieser gestoppt und ein neuer Öffnungszyklus begonnen werden. Die entscheidende Echtzeitforderung ist, daß nach Durchlaufen der Lichtschranke die Tür nach einer Zeit T geöffnet sein muß. (Alle diese qualitativen Angaben sind selbstverständlich noch durch quantitative Angaben zu präzisieren, worauf hier aber verzichtet werden soll.)

Im folgenden Schritt erfolgt der **Systementwurf,** beginnend mit einem 1. **Hardwarekonzept**: Die Schnittstelle Umwelt–Rechner wird durch 3 Eingangssignale (Lichtschranke *LS* und Endschalter *ES 1*, *ES 2*) und durch 2 Ausgangssignale (Motoransteuerung vorwärts und Motoransteuerung rückwärts) gebildet (Bild 2.20). Alle diese Signale sind binär. Folglich sind keine Signalwandlungen erforderlich, und es kann direkter Anschluß an den Eingabeport und an den Ausgabeport des Rechners erfolgen.

Für den Rechner genügt in diesem Fall ein Mindestumfang an Hardware. (Im Vorgriff auf die im folgenden Kapitel erfolgende Hardwarebeschreibung soll hier angenommen werden, daß diese Aufgabe von einem Einchipmikrorechner ab 4-bit-Wortbreite lösbar ist, d. h., die gesamte Steuereinrichtung besteht aus einem integrierten Schaltkreis.)

Für diese Hardware ist nun die **Software** zu entwerfen. Als erster Ansatz soll versucht werden, die Erfassung der Eingangssituation, also insbesondere das Erkennen der Lichtschrankenaktivierung, durch zyklische Abfrage (polling) zu realisieren (s. Abschnitt 2.2.5.).

Bild 2.21a zeigt die dazu erforderliche Grundstruktur des Programms: Nach Einschalten des Systems muß durch einen ersten Programmteil zunächst ein Grundzustand eingestellt werden (Tür schließen). Danach sind in unendlicher zyklischer Folge zwei Aktivitäten auszuführen:

1. Abtastung der Eingangssignale und
2. Berechnung der erforderlichen Reaktionen und Ausgabe der entsprechenden Stellsignale.

Das Polling-Prinzip und damit eine solche Programmstruktur, ist nur anwendbar, wenn die Programmschleife auch im ungünstigsten Fall so schnell durchlaufen wird, daß keine Lichtschrankenaktivierung unerkannt bleibt. Das Einhalten dieser Forderung kann erst geprüft werden, wenn das Programm erstellt ist und damit dessen Laufzeit bekannt ist.

Der Entwurf ist nun entsprechend durch eine schrittweise Verfeinerung der Programmblöcke bis hin zu eindeutigen Algorithmen fortzusetzen: Die Abtastung der Eingangssignale besteht aus einer einzelnen Aktion. Da alle Eingangssignale an einem Eingabeport angeschlossen sind, wird diese Teilaufgabe bereits durch einen Eingabebefehl übernommen. Das darauffolgende Teilprogramm erweist sich dagegen als komplexer. Es ist offensichtlich, daß die Reaktionen des Steuerrechners, die bei bestimmten Eingangssignalen erforderlich werden, vom Zustand abhängen, in dem sich das System gerade befindet. Folgende 4 Zustände sind zu unterscheiden: *1* – Tür geschlossen, *2* – Tür wird geöffnet, *3* – Tür offen, *4* – Tür wird geschlossen.

Damit ist es zunächst erforderlich, den aktuellen Systemzustand zu ermitteln. Zu diesem Zweck wird im Rechner der Zustand Z durch Abspeicherung einer Zahl $Z = 1, 2, 3$ oder 4 auf einem Platz im Datenspeicher (oder Register) aufbewahrt. (Dieser Platz wird beim Herstellen des Grundzustandes auf $Z = 1$ (Tür geschlossen) gesetzt.) Der Ablaufplan wird damit in einem ersten Schritt verfeinert und führt auf die Struktur gemäß Bild 2.21b.

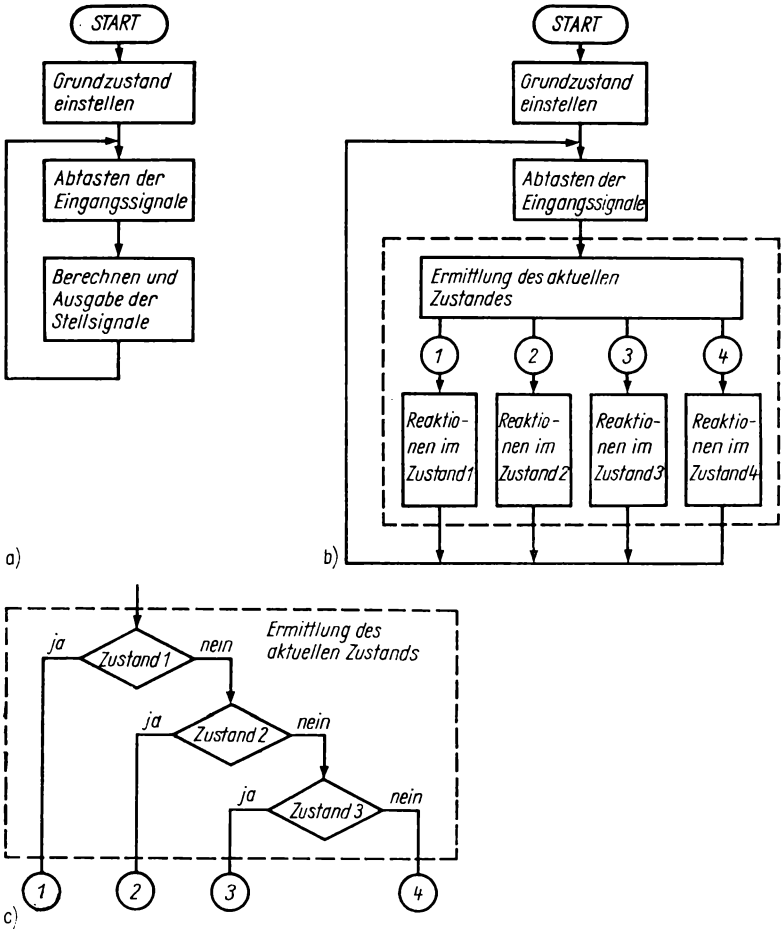


Bild 2.21. Schrittweiser Entwurf des Algorithmus
a) Grundstruktur; b) 1. Präzisierung; c) Teilalgorithmus

Die Ermittlung des aktuellen Zustandes, also die Verzweigung des Ablaufs in 4 Richtungen, kann durch eine Folge von 3 Alternativen realisiert werden (Bild 2.21c). Für die weitere Verfeinerung der Programmteile „Reaktionen im Zustand i" gilt es nun im einzelnen festzulegen, welche Wirkungen (Stellsignale) die Eingangssignale in jedem der 4 Zustände auslösen müssen. Wir wollen dies zunächst in Form einer Matrix darstellen (Tafel 2.2):

Anhand dieser Matrix lassen sich sofort die 4 Teilalgorithmen als Ablaufpläne ermitteln. Bild 2.22 zeigt den somit erhaltenen Gesamtalgorithmus.

Näher zu erläutern ist noch die Realisierung der Haltezeit T_0 für den Zustand „offen". Dazu wird im Rechner eine weitere Zahl T verwendet, die als Zeitzähler bezeichnet werden soll und für die ein weiterer Datenspeicherplatz oder ein Register reserviert wird. Diese Zahl wird mit Erreichen des Zustandes 3 gleich 0 gesetzt und danach mit jedem Schleifendurchlauf, falls keine weiteren Personen die Lichtschranke aktivieren, um 1 erhöht (Addition $+1$). Da die Schleifenlaufzeit des Programms exakt berechenbar ist, läßt sich dementsprechend ein End-

Tafel 2.2. Steuerreaktionen als Funktion der Eingangssituation und des Momentanzustandes für das Beispiel der selbsttätigen Türöffnung

ZUSTAND / EREIGNIS	1 Tür geschlossen	2 Tür wird geöffnet	3 Tür offen	4 Tür wird geschlossen
LICHTSCHRANKE LS = EIN	· Motor VORWÄRTS · Zustandsänderung: $Z = 2$		· Zeitzähler $T = 0$	· Motor AUS · Motor VORWÄRTS · Zustandsänderung: $Z = 2$
ENDSCHALTER 1 $ES\ 1$ = EIN		· Motor AUS · Zeitzähler $T = 0$ setzen · Zustandsänderung: $Z = 3$		
ENDSCHALTER 2 $ES\ 2$ = EIN				· Motor AUS · Zustandsänderung: $Z = 1$
Alle Eingänge AUS			· Zeitzähler erhöhen ($T + 1$) · Prüfen, ob Endwert des Zeitzählers (T_{max}) erreicht ist, wenn ja: · Motor RÜCKWÄRTS · Zustandsänderung: $Z = 4$	

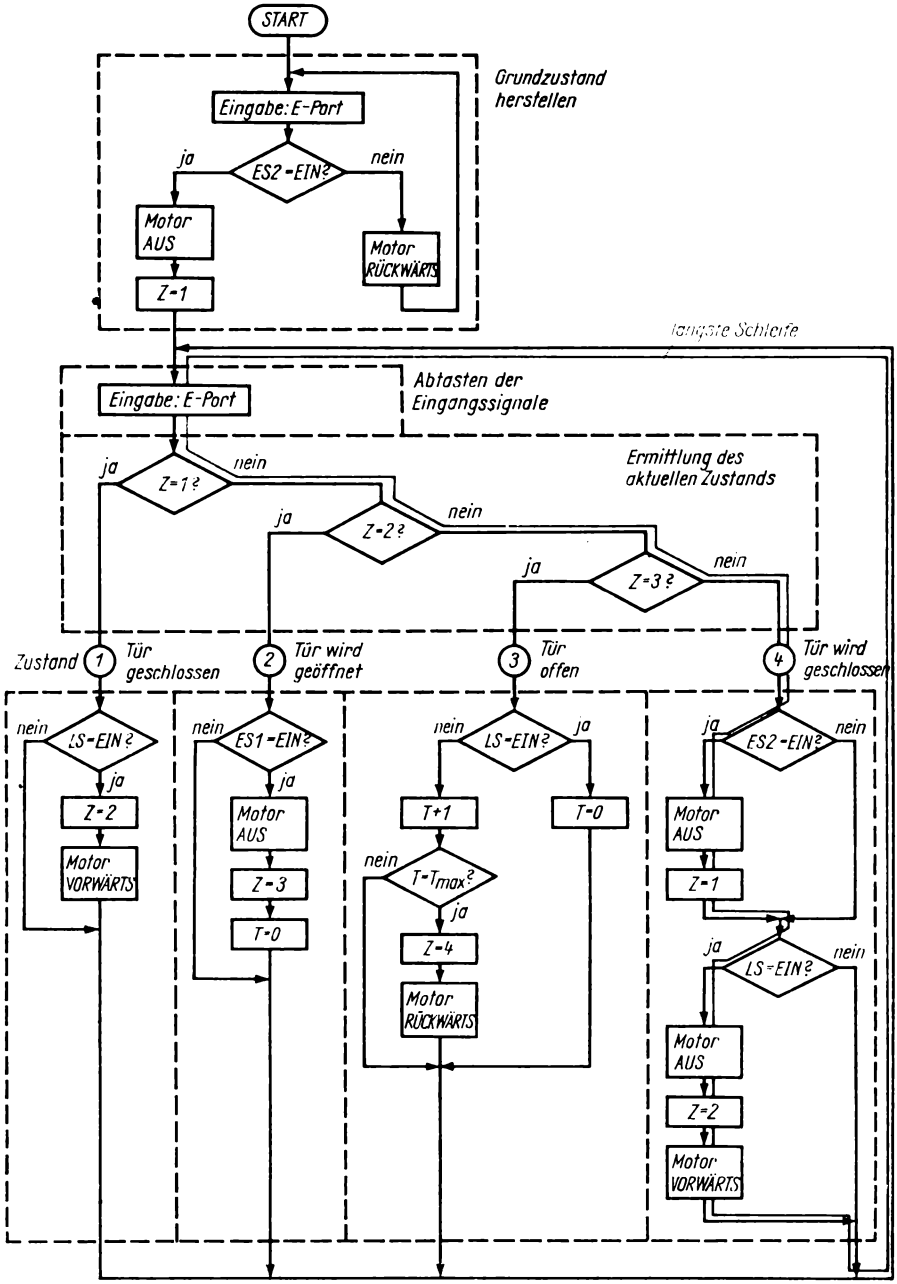


Bild 2.22. Gesamtalgorithmus zur Türsteuerung

wert T_{max} angeben, der der geforderten Haltezeit T_0 entspricht. Das Erreichen dieses Endwertes ist folglich das Signal für den Übergang in den Zustand 4 (Tür schließen). Sollten jedoch weitere Personen die Lichtschranke aktivieren, dann wird jeweils der Zeitzähler auf 0 zurückgesetzt und die Haltezeit entsprechend verlängert.

Der letzte Entwurfsschritt erfordert die Umsetzung des im Bild 2.22 dargestellten Algorithmus in ein Rechnerprogramm. Jede der dort dargestellten Aktionen läßt sich durch einen oder durch mehrere Befehle realisieren. Wir wollen diesen Schritt hier nicht vollziehen (erst im Abschnitt 4. soll diese nächste Ebene behandelt werden), aber es sollen einige Ergebnisse angegeben werden, die für die abschließende Eignungsprüfung des angewendeten Prinzips erforderlich sind: Die Umsetzung des Algorithmus in ein Programm für einen bestimmten Mikrorechner führt z. B. auf ein Programm mit insgesamt 64 Befehlen. Die längste Schleife (im Bild 2.22 rot angegeben) umfaßt 23 Befehle und benötigt eine Schleifenlaufzeit von 30 µs. Damit ist das längste Abtastintervall (ungünstiger Fall) bekannt.

Es gilt nun noch zu prüfen, ob diese Abtastfrequenz ausreicht, um keine Lichtschrankenaktivierungen unerkannt zu lassen. Dazu ist die kürzeste Impulsbreite, die durch die Lichtschranke erzeugt wird, abzuschätzen. Eine in schnellem Tempo laufende Person (30 km/h, 20 cm Breite) erzeugt einen Impuls von 24 ms, d. h., ein solcher Impuls würde durch den Rechner 800mal erkannt.

Somit ist nicht nur bestätigt, daß der gewählte Lösungsansatz für diese Steuerungsaufgabe ausreichend ist, sondern es ist auch offensichtlich, daß der Rechner mit dieser Aufgabe bei weitem nicht ausgelastet wird. Folgende ergänzende Anmerkungen seien deshalb noch hinzugefügt:

Es kann an dieser Stelle durchaus der Zweifel entstehen, ob es für eine solche Steuerungsaufgabe nicht einfachere elektronische Lösungen gibt; ob hierfür überhaupt ein Rechner erforderlich ist. Dazu ist zunächst die Bedeutung von einfach in diesem Zusammenhang zu klären. Entscheidend ist letztlich der ökonomische Gesamtaufwand, und dieser wird, was die Herstellungskosten betrifft, entscheidend durch die Anzahl der integrierten Bausteine und nicht durch deren innere Struktur bestimmt. Das heißt, es wird weniger von Bedeutung sein, eine Steuerung mit der einfachsten Schaltungsstruktur zu entwickeln, als vielmehr die Lösung mit der kleinsten Bausteinanzahl zu suchen. Daher gewinnen Lösungen mit Mikrorechnern selbst für einfachste Aufgaben so an Bedeutung.

Es ist zwar offensichtlich, daß bei dem betrachteten Beispiel der verwendete Mikrorechner nicht ausgelastet wird. Dies ist aber überhaupt kein Grund, dessen Einsatz in Frage zu stellen, sondern sollte vielmehr zu Überlegungen Anlaß geben, ob es nicht weitere nützliche Aufgaben gibt, die zusätzlich vom Mikrorechner übernommen werden können. Abgesehen davon, daß ein solcher Rechner mehrere Türen steuern könnte (falls diese Aufgabe vorliegt), sind durchaus auch bei einer Einzeltürsteuerung weitere Funktionen zweckmäßig. So wäre es z. B. sinnvoll zu prüfen, ob nach Einschalten des Motors (vorwärts oder rückwärts) die Endschalter wieder öffnen. Auf diese Weise könnten vom Rechner die Funktionstüchtigkeit des Motors und die der Bewegungseinrichtungen überprüft werden. Im Fehlerfall würden dann vom Rechner entsprechende Alarmsignale ausgelöst werden (optische oder akustische Anzeige bzw. Weiterleitung an einen übergeordneten Rechner oder Dispatcher).

Grundsätzlich sollten solche überschüssigen Verarbeitungskapazitäten neben einer Erweiterung des Steuerungskomforts insbesondere zur Erkennung von Fehlerzuständen dienen (Eigendiagnose).

3. Mikrorechner-Hardware

Im Abschnitt 2.2. haben wir die Arbeitsweise eines Rechners und die erforderlichen Funktionseinheiten (Hardware) kennengelernt. Ziel des vorliegenden Abschnittes soll es nun sein, die Möglichkeiten aufzuzeigen, die durch die Mikroelektronik zur Realisierung der Rechner-Hardware geboten werden, und die daraus resultierenden Besonderheiten der Mikrorechner abzuleiten.

Die Herstellungsverfahren für integrierte Schaltkreise ermöglichen, auf einem Halbleiterchip von wenigen Quadratmillimetern einige 10000 bis 100000 Transistoren aufzubringen. Damit ist es prinzipiell möglich, sämtliche Funktionseinheiten eines Rechners (unter bestimmten Einschränkungen seiner Leistungsparameter) auf einem großintegrierten Baustein (LSI-Baustein) anzuordnen. Von dieser Möglichkeit wird durchaus Gebrauch gemacht, aber das ist nicht der einzige und auch zukünftig nicht generell anzustrebende Weg. Wir können zwischen 3 Konzepten unterscheiden, die bei der Realisierung von Rechnern mit Hilfe von LSI-Bausteinen angewendet werden:

1. **Einchiprechner:** Alle Funktionseinheiten eines Rechners sind in einem Schaltkreis enthalten. Der Vorteil des geringen Platzbedarfs ist allerdings mit dem Nachteil verbunden, daß bei diesem Konzept die Struktur des Rechners und seine Leistungsparameter bereits vom Bauelementehersteller in relativ engen Grenzen vorgegeben sind. Dies trifft in erster Linie auf die Größe der Speicherbereiche und die Anzahl der E/A-Ports zu. Es wird auch auf längere Sicht nicht möglich sein, solche Speicherkapazitäten auf einem Chip unterzubringen, die für den überwiegenden Teil der Anwendungsfälle benötigt werden. Einchiprechner werden damit immer die unterste Klasse der Rechner bilden, aber aufgrund ihrer Wirtschaftlichkeit die zahlenmäßig am meisten eingesetzte.
2. **Mikroprozessoren und LSI-Bausteinsätze für Einkartenrechner (Mikroprozessorsysteme):** Dieses 2. Konzept geht davon aus, daß eine wesentlich größere Flexibilität beim Systementwurf erreicht wird, wenn auf einem LSI-Baustein nur Teilfunktionen eines Rechners zusammengefaßt werden und damit die Möglichkeit besteht, für den konkreten Anwendungsfall maßgeschneiderte Rechner zu entwerfen.
   Naheliegend war dabei, die gesamten Funktionseinheiten des Zentralprozessors auf einem Baustein anzuordnen, der dementsprechend als Mikroprozessor bezeichnet wird. Weiterhin werden Bausteine für Speicherbaugruppen und für das E/A-System bereitgestellt, so daß insgesamt ein Bausteinsatz entsteht, der den Aufbau von Rechnern in der Größe einer oder weniger Leiterplatten ermöglicht (Einkartenrechner/MR-Baugruppensysteme).
   Mit diesem Konzept können der Umfang des Speichers sowie die Anzahl und Art der E/A-Kanäle in weiten Grenzen dem Anwendungsfall angepaßt werden. Unveränderbar sind dagegen der Befehlssatz und die Wortbreite des Rechners, da diese Parameter von dem als LSI-Baustein vorgegebenen Zentralprozessor festgelegt werden.
3. **Bausteinsätze für Zentralprozessoren (Scheibenprozessoren, Slice-Prozessoren):** Bei diesem Konzept wird eine noch weitergehende Untergliederung der Rechnerstruktur vorgenommen, indem der Zentralprozessor in geeignete Teilkomponenten zerlegt und ein Satz von LSI-Bausteinen für den Aufbau von Pro-

zessoren bereitgestellt wird. Dadurch können nun auch der Befehlssatz und die Wortbreite des Rechners an den Anwendungsfall individuell angepaßt, also Spezialprozessoren vom Anwender realisiert werden. Die Zerlegung des Prozessors wird dabei so vorgenommen, daß meist 2 Bausteine für den steuernden Teil (mikroprogrammgesteuerter Steuerbaustein und Mikroprogrammspeicher) sowie ein Arithmetik-Logik-Baustein mit einer Wortbreite von 2 oder 4 bit vorhanden sind. Der Arithmetik-Logik-Baustein ist aber kaskadierbar, so daß scheibenweise eine beliebige Wortbreite realisierbar ist.

Es ist offensichtlich, daß in der oben angegebenen Reihenfolge die Flexibilität für den Rechnerentwurf zwar wächst und damit eine bessere Anpassungsfähigkeit an den Einsatzfall möglich wird, daß aber der Entwurfsaufwand und der Platzbedarf entsprechend größer werden.

Der Systementwurf bei Einchiprechnern beschränkt sich nahezu ausschließlich auf die Programmentwicklung. Beim Mikroprozessorkonzept liegt der Schwerpunkt ebenfalls bei der Programmentwicklung, aber es sind außerdem Hardwareentwurfsarbeiten mit geringem Schwierigkeitsgrad erforderlich. Die Anwendung des Scheibenkonzepts setzt dagegen ein tieferes Eindringen in das Wechselspiel zwischen Hardware und Software voraus.

Die Entwicklung der Mikrorechentechnik begann mit dem Mikroprozessorkonzept, und es ist anzunehmen, daß diese „mittlere" Variante auch auf längere Sicht eine Vorrangstellung behalten wird. Für Massenanwendungen mit geringeren Leistungsforderungen sind jedoch die Einchiprechner die wirtschaftlichste Lösung.

Im folgenden wollen wir daher zuerst die mikroprozessororientierten Baugruppensysteme für Einkartenrechner behandeln. Diese Kenntnisse bilden zugleich die Grundlage für die anschließende Beschreibung eines Einchiprechners. Die Problematik der Scheibenprozessoren überschreitet dagegen die Zielstellung dieses Buches.

3.1. Mikroprozessorsysteme (Bausteinsätze für Einkartenrechner)

3.1.1. Modularkonzept/Mikrorechnerbus

Im Abschnitt 2.2. haben wir die erforderlichen Baugruppen eines Rechners behandelt (Bild 2.15) und festgestellt, daß der Zentralprozessor den Kern eines Rechners bildet und von diesem Kommunikationswege zum Programm-, Daten- und Statusspeicher sowie zu den E/A-Ports führen. Soll nun diese Baugruppe auf einem LSI-Schaltkreis integriert, die Speicherblöcke und das E/A-System auf weitere Schaltkreise verteilt werden, dann ergeben sich 2 Forderungen:

1. Zusammenschaltbarkeit dieser Baugruppen in der Form, daß ein Modularkonzept gewährleistet ist, also verschiedene Ausbaustufen des Rechners möglich sind,
2. Reduzieren der Anzahl der Verbindungsleitungen zwischen den Schaltkreisen.

Diese letzte Forderung ist ein Grundproblem der integrierten Schaltkreistechnik, da von der Anzahl der erforderlichen Schaltkreisbeine (pins) entscheidend die Größe der Schaltkreise und deren Herstellungskosten abhängen. Eine direkte Umsetzung der im Bild 2.15 gezeigten Struktur scheidet damit aufgrund der vielen Kommunikationswege zwischen Zentralprozessor und den übrigen Baugruppen aus. Die Lösung des Problems besteht daher bei diesem Mikrorechner-

konzept darin, daß nur noch ein einziger interner Kommunikationsweg vorgesehen wird, der sog. Mikrorechnerbus (MR-Bus).

Der **Mikrorechnerbus** besteht aus einer Menge von Verbindungsleitungen, die das Rückgrat des Mikrorechners bilden. An diese einheitliche Schnittstelle werden alle Baugruppen angeschlossen und darüber alle rechnerinternen Kommunikationen abgewickelt. Damit ist zugleich eine beliebige Aneinanderreihung der Baugruppen und folglich der Aufbau an den jeweiligen Einsatzfall angepaßter Rechner aus vorgefertigten LSI-Schaltkreisen bzw. Baugruppen möglich. Die Struktur eines solchen Rechnerkonzepts zeigt Bild 3.1a. Zum Mindestumfang gehören:

- Zentralprozessormodul (Mikroprozessor)
- Speichermodule für Programm, Daten und Statusinformation
- E/A-Module.

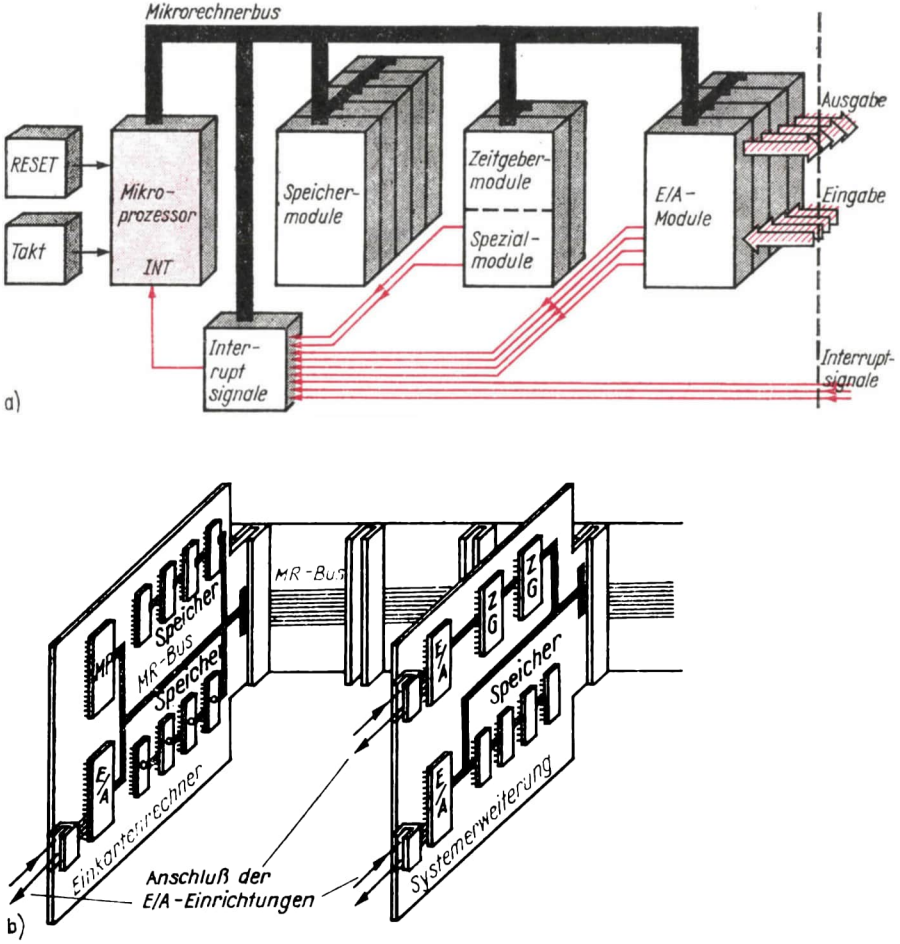


Bild 3.1. Mikroprozessorsystem (Modularkonzept für Mikrorechner)

a) Systemstruktur

b) technische Realisierungsform des Einkartenrechners mit Erweiterungsbaugruppen

Darüber hinaus werden solche Schaltkreissysteme oft durch spezielle Module erweitert, z. B.:

- Zähler/Zeitgeber-Module
- Arithmetikmodule
- Spezialprozessoren.

Die Beschränkung der Baugruppenkopplung auf ein einziges internes Transportsystem aufgrund der notwendigen Reduzierung der Anzahl der Schaltkreisbeine, ist das eigentliche Spezifikum der Mikrorechner-Hardware. Dadurch wird zwar das Modularkonzept vorteilhaft unterstützt, aber zugleich die Kommunikationsmöglichkeiten zwischen den Baugruppen in der Weise eingeschränkt, daß nicht mehrere Kommunikationen gleichzeitig ausführbar und folglich Verluste an Arbeitsgeschwindigkeit gegenüber anderen Realisierungsformen von Rechnern bedingt sind.

Bild 3.1a sagt dabei zunächst noch nichts über die konkrete Realisierung eines solchen Mikrorechners aus. Die an den Bus angeschlossenen Module können sowohl aus einem einzelnen LSI-Schaltkreis als auch aus mehreren Schaltkreisen und Bauelementen bestehen. Der Mikrorechnerbus kann dementsprechend eine Menge von Leiterbahnen auf einer gedruckten Leiterplatte oder eine gemeinsame Rückverdrahtung in einem Gestell für mehrere Leiterplatten sein. In der Regel ist es aber möglich, alle Baugruppen eines Rechners (in einem begrenzten Umfang) auf einer Leiterplatte unterzubringen (Einkartenrechner) und spezielle Erweiterungsleiterplatten für größere Ausbaustufen vorzusehen (Bild 3.1b).

Hinsichtlich ihrer Rolle, die die am Bus angeschlossenen Module bei den Kommunikationen spielen, müssen wir zwischen aktiven und passiven Modulen unterscheiden:

Aktive Module dürfen den Informationsaustausch mit anderen Modulen auslösen, während **passive Module** lediglich nach Aufforderung an der Kommunikation teilnehmen. Bisher haben wir nur den Zentralprozessor als aktiven Modul kennengelernt, der im Laufe eines Befehlszyklus immer mit dem Programmspeicher kommuniziert (Befehl holen) und, falls zur Ausführung des Befehls erforderlich, einen Informationsaustausch mit dem Daten- und Statusspeicher oder mit E/A-Ports anstößt. Aufgrund seiner aktiven Rolle bestimmt der Mikroprozessor damit die Arbeitsweise des Mikrorechnerbus und folglich die interne Schnittstelle zwischen allen Modulen des Mikrorechners.

Wir wollen im folgenden erläutern, wie ein Kommunikationsvorgang zwischen den Baugruppen abläuft und wie der Mikrorechnerbus organisiert ist:

Dabei soll zunächst vorausgesetzt werden, daß am MR-Bus nur ein aktiver Modul, der Mikroprozessor, und eine größere Anzahl passiver Module angeschlossen sind. Der aktive Modul muß immer den Kommunikationsvorgang mit der Ausgabe einer Adreßinformation beginnen, um den gewünschten Kommunikationspartner aus der Menge der passiven Module auszuwählen. Alle passiven Module verfolgen laufend die über den Bus ausgesendeten Adressen und schalten sich nur bei Auftreten des für sie gültigen Adreßwortes zu. Der ausgewählte passive Modul liefert daraufhin das unter dieser Adresse gespeicherte Datenwort bzw. übernimmt das vom Prozessor gesendete. Um z. B. die Richtung des Informationsaustausches und bestimmte Zeitpunkte für die Informationsübernahme zu markieren, werden vom aktiven Modul zusätzliche Steuersignale bereitgestellt.

Damit ergeben sich die im Bild 3.2 dargestellten Verhältnisse: Die Adressen- und Steuersignale sind immer vom aktiven zu allen passiven Modulen gerichtet, während der Transport des Datenworts in beiden Richtungen (bidirektional) möglich sein muß, obwohl durchaus Module existieren, die jeweils nur eine Transportrichtung benötigen.

Eine 1. Realisierungsform für den Mikrorechnerbus besteht darin, für alle diese Signale separate, also parallel angeordnete Leitungen zu verwenden. Der Bus

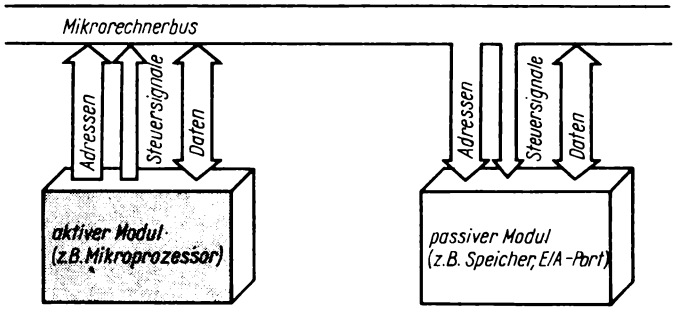


Bild 3.2. Mikrorechnerbus/Informationsarten und Modulankopplung

zerfällt dementsprechend in 3 Teile: in den **Adreß-**, in den **Daten-** und in den **Steuerbus** (Bild 3.3a). Die Gesamtanzahl der Busleitungen wird dabei hauptsächlich durch den Adreßumfang des Mikroprozessors (Breite des Adreßbus) und dessen Wortbreite (Breite des Datenbus) bestimmt.

Da, wie bereits erläutert, die Anschlußanzahl der LSI-Schaltkreise jedoch begrenzt ist (z. B. nicht größer als 40 sein sollte), ergeben sich sehr schnell Beschränkungen für diese Parameter. Bei 16-bit-Mikroprozessoren mit großem Adreßumfang muß deshalb von einer 2. Realisierungsform Gebrauch gemacht werden, nämlich Busleitungen mehrfach auszunutzen, indem z. B. über ein Leitungsbündel Adressen und Daten zeitlich nacheinander übertragen werden **(Zeitmultiplexbus)** (Bild 3.3b). Damit erhöht sich allerdings der organisatorische Auf-

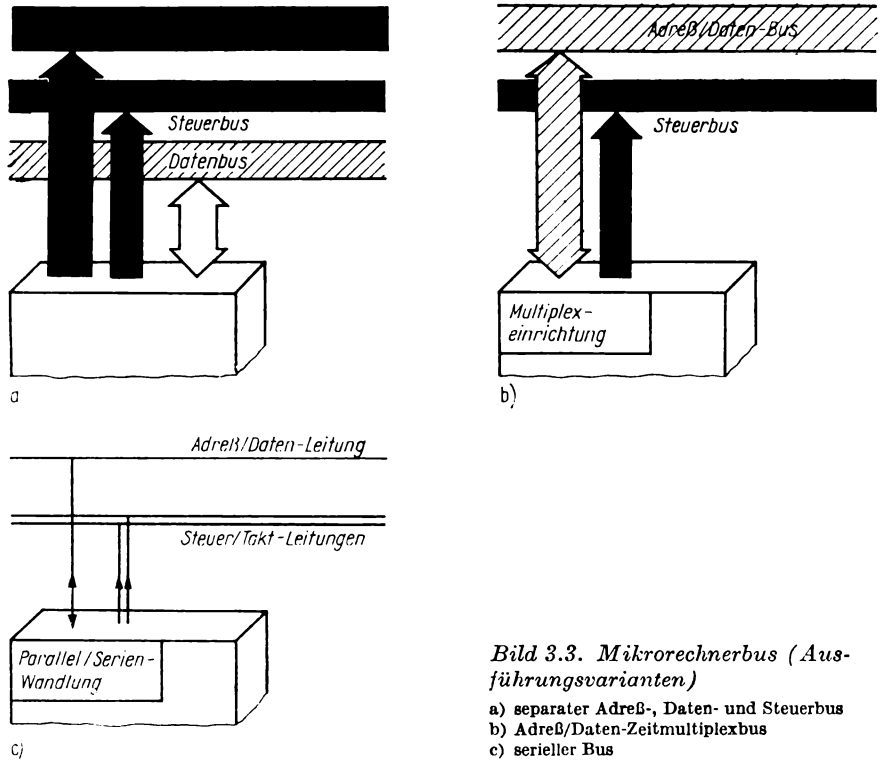


Bild 3.3. Mikrorechnerbus (Ausführungsvarianten)

a) separater Adreß-, Daten- und Steuerbus
b) Adreß/Daten-Zeitmultiplexbus
c) serieller Bus

wand bei der Kommunikation, d. h., es sind entsprechende zusätzliche Funktionen im Inneren der Schaltkreise zu realisieren.

Eine drastische Senkung der Pinanzahl der Schaltkreise (und damit ihrer Abmessungen) läßt sich durch eine 3. Realisierungsform erreichen. Dabei erfolgt die Kommunikation zwischen den Modulen durch **serielle Übertragung** (alle Bits werden zeitlich nacheinander übertragen). Der Mikrorechnerbus reduziert sich somit auf eine (oder wenige) Leitungen (Bild 3.3c). Allerdings sind dann noch aufwendigere Schaltungen zur Formatwandlung in allen Schaltkreisen erforderlich, da die modulinternen Verarbeitungs-, Speicher- und Transportprozesse parallel organisiert sind. Außerdem erhöht sich die erforderliche Zeit für einen Kommunikationsvorgang. Aus diesem Grund wird gegenwärtig bei den meisten Systemen von den ersten beiden Realisierungsformen Gebrauch gemacht.

Bild 3.4 zeigt den zeitlichen Signalablauf bei Kommunikationen über einen Mikrorechnerbus der 1. Art:

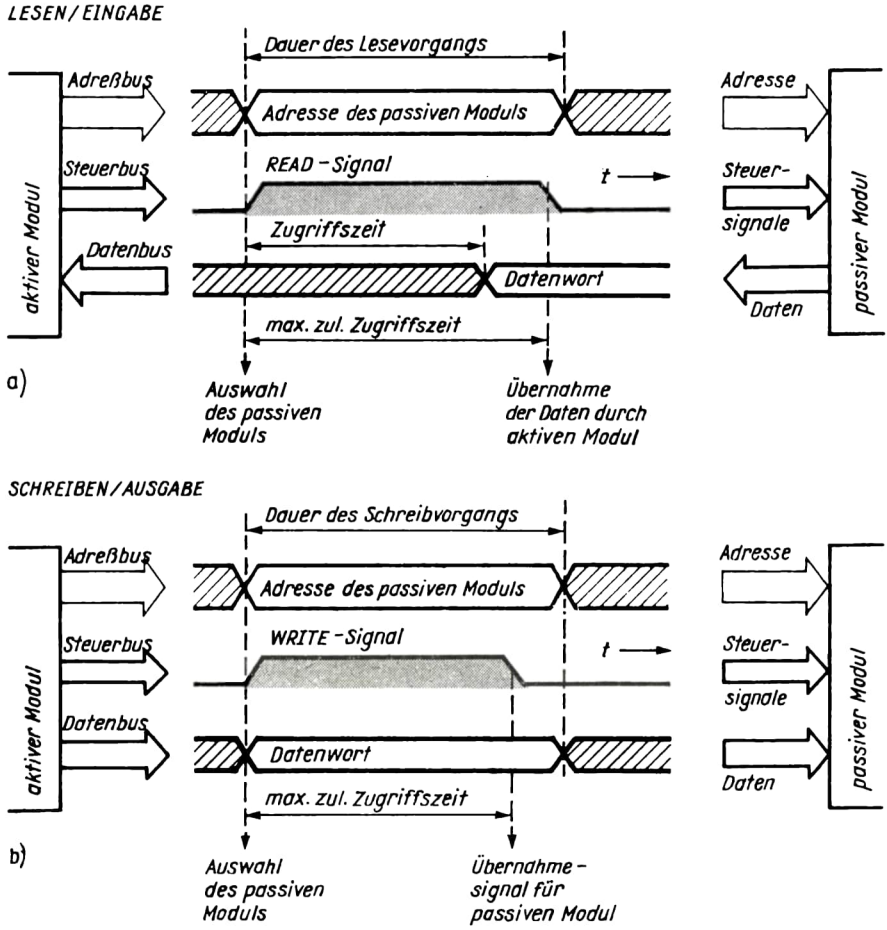


Bild 3.4. Zeitlicher Ablauf eines Kommunikationsvorganges

a) Lese- oder Eingabeoperation; b) Schreib- oder Ausgabeoperation

Bei einer Leseoperation (Information zum aktiven Modul gerichtet) werden die Adresse und das Steuersignal LESEN (READ-Signal) vom aktiven Modul über Adreß- und Steuerbus ausgegeben. Der angesprochene passive Modul wird nach einer für Adreßerkennung und interne Abläufe notwendigen Zeit (Zugriffszeit) die Information (Datenwort) auf den Datenbus abgeben. Diese Verzögerungszeit ist beim Entwurf des aktiven Moduls berücksichtigt worden, indem dieser erst nach einer etwas längeren Zeitpause den Datenbus abtastet und die Information übernimmt. Danach wird das READ-Signal zurückgenommen und der Bus damit für weitere Kommunikationen freigegeben. Ein solches Signalspiel benötigt 2 bis 3 Grundtaktperioden und dauert damit bei den in MOS-Technologie gefertigten Mikrorechnersystemen je nach zulässiger Taktfrequenz etwa 0,2···1 µs.

Bei einer Schreiboperation (Informationsübertragung zum passiven Modul) werden vom aktiven Modul Adresse, Datenwort und das Steuersignal SCHREIBEN ausgegeben. Der angesprochene passive Modul übernimmt nach einer Verzögerungszeit die Information. Der aktive Modul ist von vornherein wieder auf diese Trägheit eingestellt und hält seine Signale für eine entsprechende Zeit aktiv.

Das Merkmal dieser Kommunikationsform ist also, daß keine gegenseitige Quittierung über die Informationsübernahme erfolgt, daß vielmehr der aktive Modul die einzuhaltenden Verzögerungszeiten bei den Schreib- und Leseoperationen von vornherein berücksichtigt.

3.1.2. Speichermodule

Jeder Mikrorechner benötigt zumindest einen Speicher für das Programm. Darüber hinaus kann die Erweiterung des im Mikroprozessor vorhandenen Registersatzes durch einen Datenspeicher bzw. durch Speicherplätze für Statusinformation erforderlich sein.

Organisationsformen

Betrachten wir zunächst die prinzipiellen Organisationsformen bei elektronischen Speichern (Bild 3.5):

Der universelle Speichertyp ist der **RAM** (random access memory, Speicher mit wahlfreiem Zugriff). Er enthält eine Anzahl von Speicherplätzen, die durch Adressen einzeln ausgewählt und in die Daten eingeschrieben und danach beliebig oft zerstörungsfrei ausgelesen werden können. Die Richtung des Datenflusses wird durch ein Steuersignal (SCHREIBEN oder LESEN) festgelegt. Entscheidend ist, daß die Zugriffe zu diesem Speicher in beliebiger Adreßfolge (wahlfrei) erfolgen können. Mit diesem Speichertyp lassen sich alle weiteren Speichertypen realisieren.

Die anderen im Bild 3.5 angegebenen Speichertypen sind Spezialfälle, die entweder durch funktionelle Abrüstung oder Erweiterung entstehen. Der **ROM** (read only memory, Nur-Lese-Speicher) ist ein abgerüsteter RAM-Typ, bei dem ebenfalls wahlfreie Zugriffe, aber nur noch Leseoperationen möglich sind. Sein Inhalt muß folglich zum Zeitpunkt des Einsatzes im Mikrorechner bereits fest vorliegen. Die Speicher vom FIFO- und LIFO-Typ dagegen besitzen keinen wahlfreien Zugriff mehr. Die Reihenfolge der Adressen zum Schreiben und Lesen wird bei diesen Speichern intern erzeugt. Bei Schreib- und Leseoperationen brauchen also keine Adressen von außen angelegt zu werden. Der **FIFO-Speicher** (first in-first out) ermöglicht die Bildung von Warteschlangen, indem durch Schreiboperationen die Daten hintereinander eingespeichert werden und durch Leseoperationen das jeweils voranstehende Wort ausgelesen und wieder gelöscht wird. Er wird immer dann benötigt, wenn zwischen einer datenliefernden Baugruppe und einer

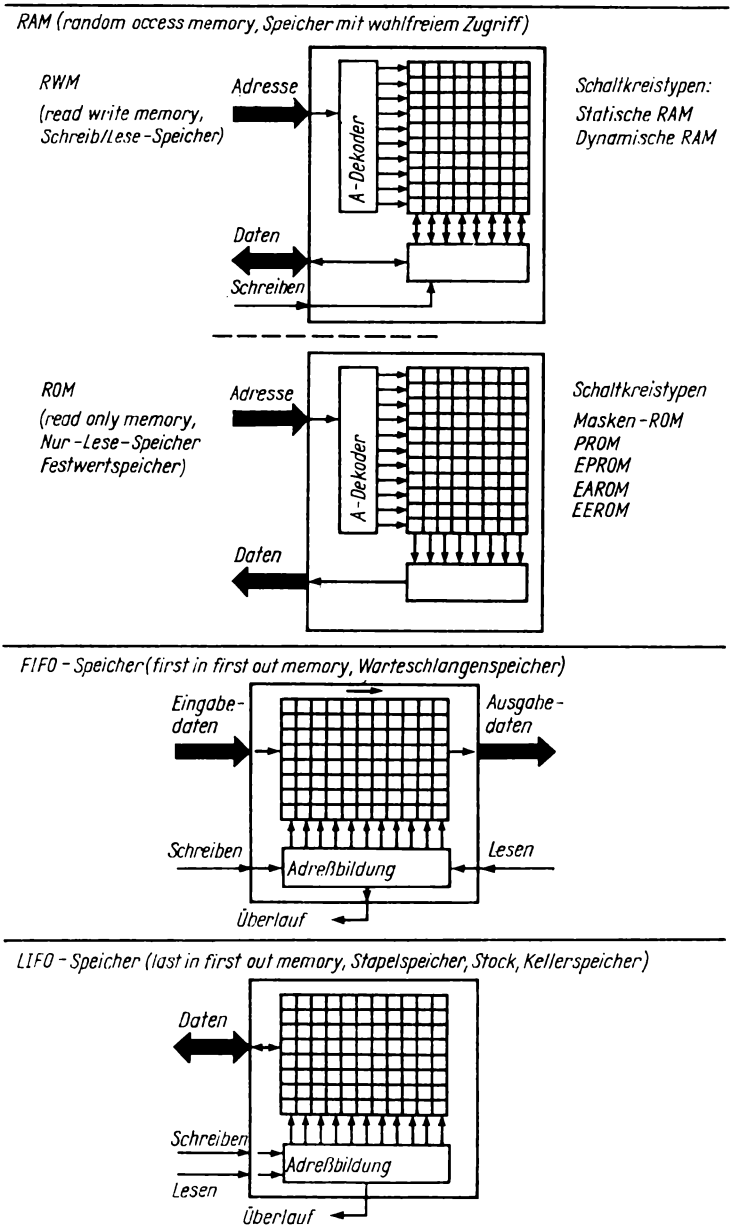


Bild 3.5. Zugriffsorganisationen verschiedener Speicher

datenempfangenden Baugruppe unterschiedliche Arbeitsgeschwindigkeiten vorliegen, die durch Zwischenpufferung in Speichern abgefangen werden müssen. Die Adressen für den jeweiligen ersten freien Speicherplatz zum Schreiben bzw. für den als nächsten zu lesenden Speicherplatz werden vom Speicherbaustein selbst gebildet; eine Überfüllung wird nach außen signalisiert. **LIFO-Speicher** (last in-first out) haben eine gegenüber den FIFO-Speichern umgekehrte Zugriffsorganisation, indem jeweils das zuletzt eingespeicherte Datenwort als erstes bei einer Leseanforderung wieder ausgegeben und gelöscht wird (deshalb auch die Bezeichnungen **Kellerspeicher** oder **Stapelspeicher (Stack)**). Selbstverständlich kann bei diesen Speichertypen aufgrund ihrer speziellen Zugriffsorganisation jedes eingespeicherte Wort nur einmal gelesen werden.

Speicher vom LIFO-Typ wären für den Statusspeicher geeignet. FIFO-Speicher sind für einige spezielle Aufgaben bei der Eingabe/Ausgabe bzw. bei der Rechnerkopplung notwendig. Für den Programm- und Datenspeicher, also für den überwiegenden Teil der Speicheraufgaben, werden jedoch grundsätzlich Speicher mit wahlfreiem Zugriff (RAM-Typ) eingesetzt.

Bisher ist es jedoch nicht gelungen, auf mikroelektronischer Basis RAM-Schaltkreise herzustellen, die alle an einen solchen Speicherbaustein gestellten Forderungen gleichzeitig erfüllen. Aus diesem Grund wird gegenwärtig eine Reihe verschiedener RAM-Typen produziert und in Mikrorechnern eingesetzt:

Am wirtschaftlichsten lassen sich die **dynamischen RAM-Schaltkreise** fertigen. Sie weisen jedoch den Nachteil auf, daß der Inhalt der Speicherzellen nur für wenige Millisekunden erhalten bleibt, aber bei einem Lesevorgang wieder aufgefrischt wird. Eine Speicherzelle darf folglich nicht über „lange" Zeiträume ohne Zugriff bleiben. (Ein scheinbar unpraktikables Verfahren. Hier gilt es wieder zu beachten, daß Millisekunden für elektronische Einrichtungen bereits lange Zeiten sind. Ein Mikroprozessor arbeitet in dieser Zeit mindestens 1000 Befehle ab.) Wird das Auffrischen nicht durch programmbedingte Zugriffe gesichert, dann ist dafür zu sorgen, daß alle Speicherzellen zyklisch (allein zum Zweck der Erhaltung ihres Inhaltes) gelesen werden. Die meisten Mikroprozessoren stellen dafür spezielle REFRESH-Signale bereit, so daß ohne wesentlichen Verlust an Arbeitsgeschwindigkeit diese Aufgabe „nebenbei" erledigt werden kann. Andernfalls sind spezielle Refresh-Baugruppen zu realisieren. Werden solche Baugruppen mit auf dem dynamischen Speicherschaltkreis integriert, dann verhalten sich diese Schaltkreise von außen betrachtet wie statische Speicher. Man bezeichnet sie entsprechend als **pseudostatische RAM-Schaltkreise.**

Statische RAM-Schaltkreise behalten ihren Inhalt, solange Betriebsspannung anliegt. Zu ihrer Herstellung sind allerdings mehr Transistoren je Speicherzelle erforderlich als bei dynamischen RAMs. Folglich bieten sie, bezogen auf die Schaltkreisfläche, geringere Speicherkapazitäten. Aber auch diese Speicher sind flüchtig, da mit Abschalten der Betriebsspannung ihr Inhalt verlorengeht.

Zum automatischen Anlauf eines Rechners ist es aber immer erforderlich, daß zumindest ein Startprogramm im Programmspeicher auch nach beliebigen Ausschaltzeiten enthalten ist. Bei vielen Mikrorechneranwendungen ist es sogar notwendig, das gesamte Programm permanent zu erhalten. Eine Lösung besteht deshalb darin, die RAM-Schaltkreise durch zusätzliche Spannungsquellen (Akkus) zu puffern, um bei Ausfall bzw. Abschalten der Netzspannung die Ruhestromversorgung zu übernehmen. Die Mikroelektronik bietet aber noch einen anderen Weg – den Einsatz von ROM-Speichern. **ROM-Schaltkreise** werden in mehreren Varianten produziert: Wird der Speicherinhalt bereits beim Herstellungsprozeß des Schaltkreises durch spezielle innere Verdrahtung vorgegeben, werden diese als **maskenprogrammierte** ROMs bezeichnet. Der Einsatz solcher ROMs setzt voraus, daß keine Änderungen des Inhalts notwendig und daß Bausteine gleichen Inhalts in großen Stückzahlen benötigt werden. Man wird diese Bausteine also bei allen Massenanwendungen von Mikrorechnern (z. B. für Konsumgüter) einsetzen.

Sind diese Bedingungen nicht erfüllt, dann stehen ROM-Bausteine zur Verfügung, die nach ihrer Herstellung einmalig oder wiederholt programmierbar sind (PROM-Bausteine, programmable ROM). Die bisher davon am meisten angewendete Version sind die **EPROM-Schaltkreise** (erasable PROM), die mit Hilfe elektrischer Signale beschrieben (programmiert) und durch UV-Strahlung wieder gelöscht werden können. Zu diesem Zweck sind diese Bausteine nicht wie üblich vollständig in Plast oder Keramik gekapselt, sondern mit einem Quarzglasfenster versehen. Schreiben (Programmieren) und Löschen sind beim Anwender ausführbar, erfordern jedoch spezielle Einrichtungen. Inzwischen werden auch EPROM-Bausteine produziert, die elektrisch programmier- und löschbar sind (EAROM, EEROM). Mit diesen Typen kommt man dem idealen Speicher langsam näher, denn der Speicherinhalt ist nicht flüchtig, und die Speicher können auf elektrischem Wege beliebig oft beschrieben und gelesen werden. Allerdings benötigt der Schreibvorgang (Löschen und Programmieren) beträchtlich mehr Zeit (Millisekundenbereich) als bei dynamischen und statischen RAMs, so daß sie als echte Schreib/Lese-Speicher noch ungeeignet sind und im System als Nur-Lese-Speicher eingesetzt werden.

Zusammenfassend soll noch einmal bemerkt werden: Die Vielfalt dieser gegenwärtig in der Mikrorechentechnik eingesetzten Speicherbausteine ist dadurch bedingt, daß die Mikroelektronik bisher keinen RAM-Speicher herstellen kann, der nichtflüchtig ist und große Speicherdichten bei niedrigen Kosten aufweist. (In der klassischen Rechentechnik wurde dieser Platz früher von den Magnetkernspeichern allein eingenommen, die aber aufgrund ihrer technologischen Basis und ihrer Parameter in der Mikrorechentechnik und inzwischen auch in der Großrechentechnik ihren Platz räumen mußten.)

Kennwerte

Neben der Angabe des Speichertyps sind die Speicherbausteine bzw. die daraus zusammengesetzten Speichermodule durch die folgenden Kennwerte charakterisiert:

1. **Speicherkapazität.** Mit N Adreßbits sind 2^N Speicherplätze auswählbar. Die Speicherkapazitäten der Schaltkreise sind daher immer Potenzen zur Basis 2. Die Entwicklung der Speicherkapazitäten ist in den letzten Jahren in rasantem Tempo vorangeschritten. Gegenwärtig ist folgender Stand erreicht:
   statische RAM-Schaltkreise: 1···8···16 Kbit[1])
   dynamische und pseudostatische
   RAM-Schaltkreise: 16, 64, 256 Kbit
   EPROM-Schaltkreise: 1, 2, 4, 8, Kbyte[2])
   EEROM-Schaltkreise: 8 Kbyte
2. **Zugriffszeit.** Zwischen Anlegen der Adresse und Abschluß des Lese- oder Schreibvorganges entsteht eine (von der Art der Herstellungstechnologie abhängige) Verzögerungszeit. Bei gegenwärtig am meisten eingesetzten Speicherbausteinen in MOS-Technologie liegen diese Zugriffszeiten im Bereich zwischen 50···500 ns.
3. **Aufrufbreite (Speicherwortbreite).** Die Speicherbausteine werden mit verschiedenen Wortbreiten gefertigt. Die durch eine Adresse aufrufbaren Speicherplätze besitzen typisch 1, 4 oder 8 bit (1 byte) Breite. Die Aufrufbreite des Speichermoduls muß gleich der Breite des Datenbus sein, wozu nötigenfalls eine parallele Anordnung mehrerer Speicherbausteine erforderlich ist.

[1]) 1 Kbit = 1024 bit

[2]) 1 Kbyte = 1024 byte
1 byte = 8 bit. Diese 8 bit werden gleichzeitig aufgerufen und parallel ausgegeben

Aufbau eines Speichermoduls

Bild 3.6 zeigt den Aufbau eines Programm- bzw. Datenspeichermoduls. Der Speicherschaltkreis (bzw. die bei nicht ausreichender Aufrufbreite parallel angeordneten Schaltkreise) können meist direkt an die Busleitungen angeschlossen werden. Da jedoch die Kapazität der Schaltkreise i. allg. kleiner als der maximal adressierbare Speicherbereich ist, können (wenn erforderlich) mehrere Speichermodule an den Bus angeschlossen werden. Dann ist aber ein Moduladreßdekoder notwendig, der jeweils erkennt, ob die anliegende Adresse innerhalb des eigenen Adreßbereichs liegt. Nur in diesem Fall aktiviert er über die Schaltkreis-Auswahlleitung (chip select) den eigenen Speicherschaltkreis.

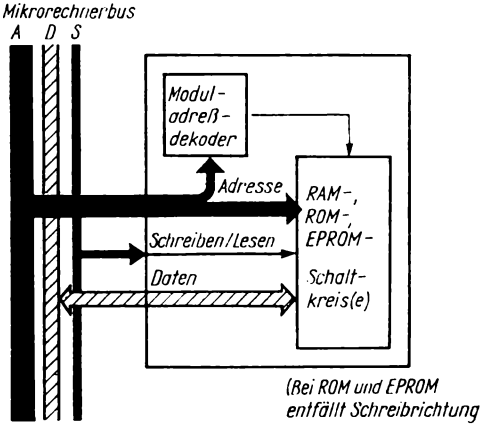


Bild 3.6. Speichermodul

Betrachten wir ein Beispiel: Besitzt der Mikroprozessor 16 Adreßleitungen (Adreßbus), dann sind 64 K ($= 2^{16} = 65536$) Speicherplätze aufrufbar. Werden 4-K-Speicherschaltkreise eingesetzt, so besteht die Möglichkeit, bis zu 16 Speichermodule an den Bus anzuschließen. Dabei sind die Speicherschaltkreise mit 12 Adreßleitungen ($2^{12} = 4$ K) zu versehen. Die restlichen 4 Adreßleitungen ($2^4 = 16$) müssen entsprechend an die Moduladreßdekoder angeschlossen werden. Anders betrachtet: Die über den Adreßbus gelieferte Information besteht aus 2 Adressen, der Moduladresse (4 bit) zur Auswahl des Moduls und der Adresse zur Auswahl des Speicherplatzes innerhalb des Moduls.

3.1.3. Zentralprozessormodul (Mikroprozessor)

Der Prozessormodul wird bei dem in diesem Abschnitt behandelten Mikrorechnerkonzept, abgesehen von einer geringfügigen Zusatzelektronik, durch einen großintegrierten Schaltkreis, den Mikroprozessor, gebildet. Da der Prozessor der aktive Kern eines Rechners ist und durch ihn viele Leistungsmerkmale festgelegt werden, kommt folglich bei einem derartigen Modularkonzept dem Mikroprozessor eine dominierende Stellung zu. Dies ist der Grund, warum häufig Begriffe, wie Mikroprozessortechnik, Mikroprozessorsysteme, Mikroprozessorsteuerung, verwendet werden, obwohl im Grunde genommen der Mikroprozessor allein kein funktionsfähiger Schaltkreis ist und erst durch weitere Baugruppen (zumindest durch einen Programmspeicher) zu einem Mikrorechner komplettiert werden muß.

Die innere Struktur eines Mikroprozessors unterscheidet sich nicht von dem im Abschnitt 2.2. behandelten Aufbau eines Zentralprozessors (Bild 2.15). Ledig-

lich die Vielzahl der Verbindungswege zwischen Prozessor und den übrigen Funktionseinheiten muß bei Mikroprozessoren auf einen einzigen, auf den Mikrorechnerbus, reduziert werden, um die Anzahl der Schaltkreisanschlüsse zu begrenzen.

Anschlußbelegungen

Bild 3.7 zeigt typische Anschlußbelegungen für Mikroprozessorschaltkreise. Die Anschlüsse lassen sich in 3 Gruppen unterteilen:

1. einige wenige Eingänge, die für die Betriebsfähigkeit des Schaltkreises notwendig sind (Betriebsspannung, Prozessortakt),
2. eine große Anzahl von (meist) Ausgängen, die den Mikrorechnerbus bilden (nur der Datenbus kann vom Prozessor auch als Eingang verwendet werden),
3. einige wenige Anschlüsse (meist Eingänge), die als einzige eine Beeinflussung des an sich starren (und vom angelegten Takt angetriebenen) Befehlszyklus ermöglichen. Auf diese Anschlüsse beschränkt sich auch bei einem solchen Mikrorechnerkonzept die noch erforderliche Hardware-Entwurfsarbeit.

Bevor wir diese Beeinflussungsmöglichkeiten näher betrachten, wollen wir die wichtigsten Kenngrößen eines Mikroprozessors zusammenstellen:

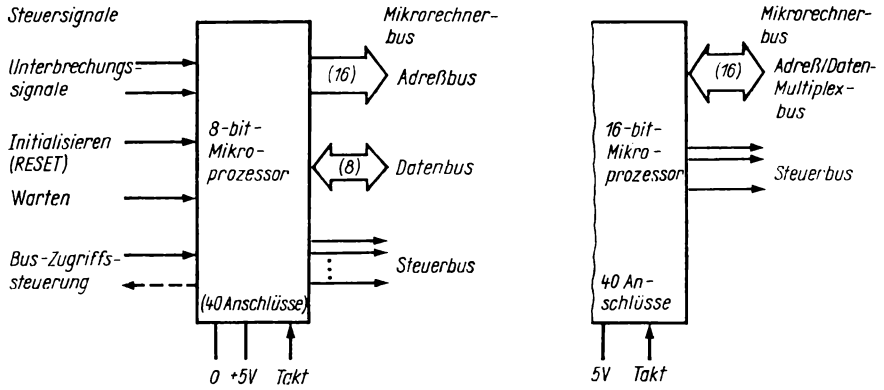


Bild 3.7. Typische Anschlußbilder für Mikroprozessoren

Kenngrößen

Verarbeitungsbreite und **Umfang des Befehlssatzes** (Was kann der Mikroprozessor?). Durch diese Parameter wird letztlich bestimmt, in wie viele Elementaroperationen eine Verarbeitungsaufgabe zerlegt werden muß (Algorithmierung, Programmierung). Je kleiner Verarbeitungsbreite und Befehlssatz, um so längere Befehlsfolgen werden erforderlich, sobald die Aufgaben bezüglich des Datenformates und der geforderten Verarbeitungsoperationen das vom Prozessor gebotene Niveau überschreiten. (Wir werden darauf im Abschnitt „Mikrorechner-Software" näher eingehen.)

Operationsgeschwindigkeit (Wie schnell arbeitet der Mikroprozessor?). Die Zeit, die der Mikroprozessor zur Ausführung eines Befehls benötigt, kann in Abhängigkeit von der Art des Befehls in einem großen Bereich schwanken. Der Grund liegt in dem Einbuskonzept. Jede Kommunikation mit einer anderen Rechnerbaugruppe benötigt eine bestimmte Anzahl von Grundtaktperioden. Damit ist für die Befehlsausführungszeit letztlich entscheidend, wie viele solcher Kommunikationen zusätzlich zum Befehlholen ablaufen müssen bzw. wie viele dafür selbst notwendig sind.

Adreßumfang für Speicher und Eingabe/Ausgabe (Wie viele Speicherzellen und E/A-Ports können direkt vom Mikroprozessor angesprochen werden?). Dieser Parameter bestimmt die Ausbaufähigkeit des Systems. Die 8-bit-Mikroprozessoren sind meist mit 16-bit-Adreßbreite ausgelegt, so daß sie 64 KByte (65536) Speicherplätze direkt adressieren können, die 16-bit-Mikroprozessoren weisen dagegen bereits Adreßbreiten von 20 bit (≙ 1 Mbyte Speicherplätze) bzw. von 23 bit (≙ 8 Mbyte Speicherplätze) auf.

Überschreitet die Anzahl der erforderlichen Speicherplätze den durch den Adreßumfang bestimmten Grenzwert, wird die Einsatzmöglichkeit dieses Mikroprozessortyps für Echtzeitsysteme mit niedrigen Reaktionszeiten bereits problematisch, da dann externe Zusatzspeicher (Massenspeicher wie Magnetplatten- oder Folienspeicher) notwendig werden. Damit sind aber zwangsläufig lange Zugriffszeiten, eine kompliziertere Organisation der Software und eine verringerte Zuverlässigkeit (elektromechanische Systeme!) verbunden.

Die Kennwerte einiger Mikroprozessoren sind aus Tafel 3.1 ersichtlich.

Tafel 3.1. Kennwerte einiger Mikroprozessoren

Wortbreite	Typ	Adreßumfang (max. Speichergröße)	Befehlsausführungszeiten in µs (kürzeste/längste)
8 bit	8080A	64 Kbyte	1,5/3,75
	8085		0,8/5,2
	Z80A		1,0/5,8
	Z80B		0,7/4,2
	U880		1,6/9,2
	M6800		1,0/2,5
	MCS 650X		0,5/3,5
16 bit	Z8002	384 Kbyte	0,3/51,5
	iAPX 86/10	1 Mbyte	0,2/19
	MC 68000	16 Mbyte	0,375/12
	iAPX 286/10	16 Mbyte	0,2/17,6
	Z8001	48 Mbyte	0,3/51,5
32 bit	iAPX 432	16 Mbyte	1,25/200
	MC 68020	4 Gbyte	0,12/5,8

1 Kbyte = 1024 byte, 1 Mbyte = 1048576 byte

Arbeitsweise

Die Arbeitsweise eines Mikroprozessors unterscheidet sich ebenfalls nicht von dem im Abschnitt 2.2. dargestellten Ablauf in einem Prozessor. Vom angelegten Takt angetrieben, erfolgt das ständige Durchlaufen von Befehlszyklen. Dieser Ablauf kann allerdings durch einige wenige Steuersignale beeinflußt werden, die wie folgt gruppiert werden können (Bild 3.7):

Unterbrechungssignale. Die Aktivierung der Unterbrechungseingänge bewirkt das Verlassen der programmgemäßen Abarbeitungsreihenfolge der Befehle und den Start eines speziellen Unterbrechungsbehandlungsprogramms. Das dazu notwendige Unterbrechungssystem ist aber meist nur teilweise im Mikroprozessor implementiert. Ergänzende Hardware ist separat zu realisieren bzw. in anderen Schaltkreisen untergebracht. Wir werden deshalb diese Problematik gesondert im Abschnitt 3.1.7. behandeln.

Steuersignal zur Initialisierung (Rücksetzen, RESET). Es sorgt für einen geord-

neten Anlauf des Mikroprozessors. Man kann davon ausgehen, daß bei Inbetriebnahme (Zuschalten der Betriebsspannung und Takt) sich zufällige Inhalte in den Speicherzellen des Mikroprozessorschaltkreises ergeben und somit auch im Programmzähler. Die Folge davon wäre, daß auch die Abarbeitung des Programms an einem zufälligen Platz des Speichers beginnen würde. Ein kurzzeitiges Aktivieren des RESET-Signals bewirkt nun, daß der Programmzähler und evtl. weitere innere Register in eine definierte Ausgangsstellung gebracht werden (z. B. Programmzähler = 0 und damit Start bei Speicherzelle 0). Dieses Signal kann manuell erzeugt werden (RESET-Taste). Um aber abzusichern, daß auf keinen Fall nach Zuschalten der Betriebsspannung erst zufällige Befehlsfolgen abgearbeitet werden, muß zusätzlich eine entsprechende elektronische Schaltung realisiert werden, die zwangsweise ein Rücksetzsignal beim Einschalten der Betriebsspannung erzeugt (automatisches Rücksetzen).

Signal zur zeitlichen Dehnung des Befehlszyklus (Warten, Schrittbetrieb). Es kann aus 2 Gründen erforderlich werden:

a) Die Kommunikationspartner des Mikroprozessors sind nicht in der Lage, innerhalb der vom Mikroprozessor vorgegebenen Zeiten zu reagieren. Wir hatten bei der Behandlung der Arbeitsweise des Mikrorechnerbus gesehen, daß in der Regel keine Quittierung der Informationsübernahme durch den passiven Modul existiert. Für die meisten Schaltkreise besteht dazu auch aufgrund ihrer schnellen Reaktionen keine Notwendigkeit. Für den Fall jedoch, daß passive Module angeschlossen werden müssen, die langsamer reagieren, besitzt der Mikroprozessor die Steuerleitung WARTEN, die, sobald sie vom passiven Modul als Notbremse aktiviert ist, dafür sorgt, daß der Kommunikationszyklus so lange angehalten wird, bis dieser Eingang wieder inaktiv wird.

b) Für Testzwecke wird eine schrittweise Abarbeitung des Programms gefordert. In diesem Fall wird die Aktivierung des Warteeingangs vom Bediener vorgenommen, und die Arbeitsweise des Systems kann in beliebiger Zeitdehnung beobachtet werden.

Steuersignale zur Buszugriffssteuerung. Sie bewirken ebenfalls einen Stop der Prozessorarbeit und darüber hinaus das Abschalten aller Sendeleitungen vom Mikrorechnerbus. Dies ist immer dann erforderlich, wenn mehrere aktive Module an einem Bus angeschlossen sind (Mehrprozessorsystem), die nur abwechselnd arbeiten können. Wir werden diese Problematik gesondert im Abschnitt 3.1.9. behandeln.

3.1.4. Eingabe/Ausgabe-Module

Der Informationsaustausch zwischen Mikrorechner und Umwelt ist eine vielschichtige Problematik. Dies resultiert aus der Vielfalt der Elemente, Baugruppen und Geräte, die als Eingabe- oder Ausgabeeinrichtungen bei den verschiedenen Einsatzfällen notwendig werden. In der Rechentechnik werden hierfür Sammelbegriffe wie Peripherie, E/A-Geräte, Umwelt verwendet. Dem Konstrukteur einer Verarbeitungsmaschine oder eines Industrieroboters bzw. dem Projektanten einer technologischen Anlage wird es sicher schwerfallen, diese funktionell komplexen und räumlich großen Objekte als Peripherie des (der) sie steuernden Mikrorechner(s) zu bezeichnen. Für die Analyse der Informationsflüsse und damit für den Entwurf der Steuerung dieser Anlagen ist diese Betrachtungsweise jedoch zwingend.

Eine Möglichkeit zur Klassifizierung der E/A-Einrichtungen für Mikrorechner bietet die im Abschnitt 2.1. eingeführte Grundstruktur bei Echtzeitsystemen (Bild 2.1). Danach können wir unterscheiden zwischen

a) **Anzeige- und Bedieneinrichtungen** für die Mensch(Bediener)-Rechner-Kommunikation:

Eingabe	Ausgabe
z. B. Taster, Schalter Tastaturen Lichtstift	Leuchtdioden (LEDs) bzw. damit realisierte Displayelemente, Bildschirmanzeigen, Drucker, Plotter usw.

b) **Meß- und Erfassungseinrichtungen** (Sensoren) und **Stelleinrichtungen** für die Objekt-Rechner-Kommunikation:

Eingabe	Ausgabe
z. B. analoge oder digitale Meßeinrichtungen für physikalische Größen (Druck, Temperatur, Kraft usw.) optische und akustische Erkennungssysteme	analoge und digitale Stelleinrichtungen für Stoff- und Energieströme (Ventile, Schütze, elektrische und hydraulische Antriebe)

Eine weitere Unterscheidung ist nach der Entfernung notwendig, da gleiche E/A-Einrichtungen unterschiedliche Kopplungslösungen bedingen, je nachdem, ob sie **lokal** oder **entfernt** angeordnet sind.

Den E/A-Systemen sind auch die **externen Speichereinrichtungen** (Massenspeicher) zuzuordnen, da die Zusammenarbeit zwischen Rechner und Massenspeicher nach den gleichen Anschlußbedingungen erfolgt wie für alle anderen obengenannten E/A-Systeme und die Stellung und Funktion der Massenspeicher im System nicht mit den Speichermodulen, die direkt an den Mikrorechnerbus angeschlossen sind, verwechselt werden darf.

E/A-Leistung des Mikroprozessors

Aus der oben genannten Typenvielfalt resultieren unterschiedliche Bedingungen, nach denen der Informationsaustausch zwischen Mikrorechner und E/A-System vollzogen werden muß. Betrachten wir zunächst, welchen Leistungsumfang der Mikroprozessor zur Eingabe und Ausgabe bietet:

Eingabe- und Ausgabeoperationen sind Kommunikationen zwischen dem Mikroprozessor als aktivem Modul und einem passiven E/A-Modul. Sie erfolgen daher ebenfalls nach dem im Bild 3.4 dargestellten Signalspiel. Im Programm enthaltene E/A-Befehle bewirken, daß der Mikroprozessor über den Adreßbus den entsprechenden Modul anspricht und über den Datenbus ein Binärwort von einem Register des Mikroprozessors zum E/A-Gerät bzw. umgekehrt transportiert. Damit gilt:

Die Eingabe/Ausgabe erfolgt wortweise (Wortbreite = Breite des Datenbus);

Der zeitliche Ablauf der Eingabe/Ausgabe wird vom Mikroprozessor bestimmt und vollzieht sich innerhalb weniger Taktperioden. Der Mikroprozessor ist damit auf der einen Seite in der Lage, mindestens 100000 E/A-Operationen je Sekunde auszuführen. Auf der anderen Seite liegen die Adreß- und Datensignale nur so kurzzeitig auf den Busleitungen an, daß die meisten externen Einrichtungen darauf nicht reagieren könnten.

Generell wird deshalb eine spezielle Kopplungsbaugruppe (E/A-Modul) zwi-

schen Mikrorechnerbus und dem E/A-Gerät zur Anpassung der unterschiedlichen Zeitverhältnisse und oft auch zur Signalwandlung notwendig (Bild 3.8a). Wir wollen uns deshalb zuerst der Frage zuwenden, welche Möglichkeiten durch mikroelektronische Schaltkreise geboten werden, um solche E/A-Module zu realisieren.

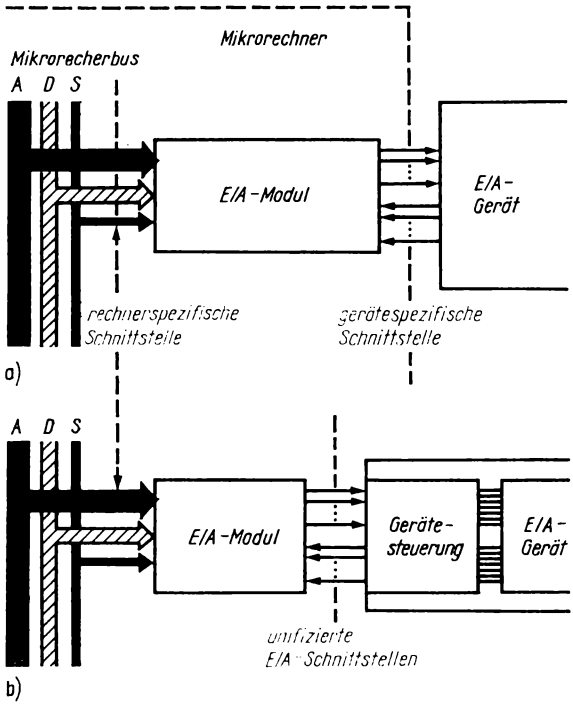


Bild 3.8. Kopplung Mikrorechner–Umwelt
a) gerätespezifischer E/A-Modul
b) E/A-Schnittstellen-Modul

LSI-Schaltkreiskonzepte für E/A-Module

Aus Bild 3.8a ist ersichtlich, daß aufgrund der beiderseitig vorhandenen individuellen Schnittstellen für jeden Mikrorechnerbus und für jede E/A-Einrichtung spezielle Schaltkreise zur Kopplung notwendig wären. Um die Diskrepanz zwischen dieser Vielfalt und der Notwendigkeit sehr großer Fertigungsstückzahlen zu lösen, werden 3 verschiedene Konzepte beim Entwurf und bei der Herstellung von E/A-Schaltkreisen angewendet:

1. **Universelle E/A-Schaltkreise** (UPI universal peripheral interface). Dieses Konzept geht davon aus, daß die größte Flexibilität dann erreichbar ist, wenn als Koppelelement wiederum ein Rechner (Einchiprechner) verwendet wird. Durch Entwicklung eines für jede E/A-Einrichtung spezifischen Programms lassen sich dementsprechend mit Hilfe eines Bausteins die unterschiedlichsten E/A-Einrichtungen ankoppeln.

2. **Gerätespezifische E/A-Schaltkreise.** Für E/A-Einrichtungen, die in großen Stückzahlen in Mikrorechnersystemen eingesetzt werden (z. B. Folienspeicher, Bildschirmanzeigen, Tastaturen usw.) wird es wirtschaftlich, spezielle E/A-Steuerschaltkreise zu fertigen (Floppy-disk-controller, CRT-Controller), die auf der einen Seite mit der Mikrorechnerbus-Schnittstelle und auf der anderen Seite mit der für ein konkretes E/A-Gerät notwendigen Schnittstelle ausgerüstet sind.

uns im folgenden mit dieser letzten Gruppe befassen und davon 4 Schnittstellentypen einführen:

1. **Quasiparallele Schnittstelle** (Bild 3.9). Diese Schnittstelle wird aus einer Menge von binären Eingabe- und Ausgabeleitungen gebildet, von denen jede eine eigene Funktion hat. Daraus folgt, daß die Signaländerungen auf diesen Leitungen in der Regel nicht gleichzeitig erfolgen. Als Beispiel wurde im Bild 3.9 die Steuerung eines Magnetbandkassettengerätes angenommen.

Der E/A-Modul muß bei diesem Typ ausgabeseitig nur eine Auffangfunktion erfüllen, indem das kurzzeitig auf dem Datenbus anliegende Ausgabewort in Speicherzellen (Auffangspeicher, Latch) übernommen wird und so lange als statisches Signalmuster anliegt, bis ein neues Wort (Signalmuster) ausgegeben wird.

Eingabeseitig hat dieser Schaltkreis im einfachsten Fall nur eine Schalterfunktion zu übernehmen. Durch einen Eingabebefehl werden die elektronischen Schalter (Bild 3.9) kurzzeitig auf den Datenbus geschaltet. Die Wirkung eines Eingabebefehls besteht damit in der Abtastung des in diesem Moment anliegenden Signalmusters aller Eingangsleitungen.

Eine modifizierte Form der Eingabe besteht darin, daß der Schaltkreis auch in dieser Richtung eine Auffangfunktion übernimmt. Der Zustand der Eingangsleitungen wird somit zu einem von der E/A-Einrichtung bestimmten Zeitpunkt übernommen und für einen später erfolgenden Eingabebefehl aufbewahrt. Der Zeitpunkt des Auffangens wird durch das zusätzliche Schnittstellensignal STROBE (Abtastzeitpunkt) getaktet. Der Zeitpunkt der Eingabe in den Rechner wird aber durch das laufende Programm bestimmt. Hierbei entsteht nun das Problem, den Rechner darauf „aufmerksam" zu machen, daß ein neues Eingabewort bereitsteht. Zu diesem Zweck sind derartige E/A-Schaltkreise mit einem speziellen Ausgang INT (Unterbrechung) ausgerüstet (Bild 3.9), der den Empfang eines neuen Eingabewortes anzeigt und damit als Unterbrechungssignal für den Zentralprozessor verwendet werden kann.

2. **Parallele Schnittstelle/Handshakeverfahren** (Bild 3.10). Parallele Schnittstellen übertragen digitale Signale in Form von Binärwörtern. Sie stellen damit, falls die E/A-Wortbreite kleiner oder gleich der Rechnerwortbreite ist, die direkte Verlängerung des Datenbus zu einer E/A-Einrichtung dar. Im Unterschied zu der oben betrachteten quasiparallelen Schnittstelle besitzen die einzelnen binären Eingangs- oder Ausgangsleitungen keine selbständige Funktion, und die Änderungen der Signalzustände erfolgen auf allen Leitungen zum gleichen Zeitpunkt.

Eine solche Schnittstelle setzt in der Regel eine Taktung des Empfängers voraus, d. h., von der Informationsquelle muß durch ein zusätzliches Taktsignal DAV (Daten vorhanden) die Übergabe eines neuen Zeichens (Binärworts, Signalmusters) angezeigt werden. Ist darüber hinaus noch Rücksicht auf die begrenzte Aufnahmefähigkeit des Empfängers zu nehmen, dann wird ein weiteres Signal (vom Empfänger zum Sender gerichtet) erforderlich, das die Übernahmebereitschaft READY (BEREIT) signalisiert.

Bild 3.10a zeigt als Beispiel einen Ausgabemodul, mit dem ein Drucker an den Mikrorechner gekoppelt wird. Mit Hilfe des Ausgabebefehls wird das zu druckende Zeichen als Binärwort kodiert übergeben und zunächst aufgefangen. Liegt Übernahmebereitschaft vom Drucker vor (READY = 1), dann erfolgt die Übergabe des Zeichenkodes auf dem parallelen Leitungsbündel, und der Drucker wird darüber durch Aktivieren von DAV informiert. Der Drucker quittiert die Zeichenübernahme durch Rücksetzen von READY. Der Ausgabeschaltkreis reagiert darauf durch Rücknahme von DAV. Die erneute Übernahmebereitschaft wird vom Drucker durch Aktivieren von READY angezeigt.

Dieses Wechselspiel der beiden entgegengesetzt gerichteten Signale wird als Handshakeverfahren bezeichnet und ermöglicht die Anpassung der unterschiedlichen Arbeitsgeschwindigkeiten zwischen dem Mikrorechner und der langsameren E/A-Einrichtung. Auf der Seite des Mikrorechners muß dann allerdings erkannt

(Dabei handelt es sich oftmals auch um die oben genannten universellen Schaltkreise (Einchiprechner), die dann allerdings bereits beim Herstelle einem speziellen (und nicht mehr änderbaren) Programm versehen werden.

3. **Schnittstellenspezifische E/A-Schaltkreise.** Bei diesem Konzept nutzt mai seit langem praktizierte Verfahren der Aufteilung der Kopplungselektron. 2 Baugruppen und einer damit zusätzlich entstehenden Zwischenschnittstelle (Bild 3.8b). Auf diese Weise läßt sich die Typenvielfalt auf einige Grundty von E/A-Schnittstellen reduzieren. Allerdings müssen die E/A-Einrichtungen einer entsprechenden Zusatzelektronik zur Realisierung einer ebensolc. Schnittstelle ausgestattet werden. Der Vorteil ist, daß nur noch wenige Sch kreistypen erforderlich sind, die die Kopplung zwischen dem Mikrorechnerk auf der einen und diesen Zwischenschnittstellen auf der anderen Seite bewirke

E/A-Schnittstellentypen

Der Standardisierungsgrad solcher E/A-Schnittstellen ist sehr unterschiedlich Es existieren sowohl internationale Standards (z. B. IEC-Bus bei rechner gesteuerten Meßsystemen), nationale und herstellertypische Schnittstellen al auch nichtstandardisierte, aber funktionell bedingte Schnittstellen. Wir wollen

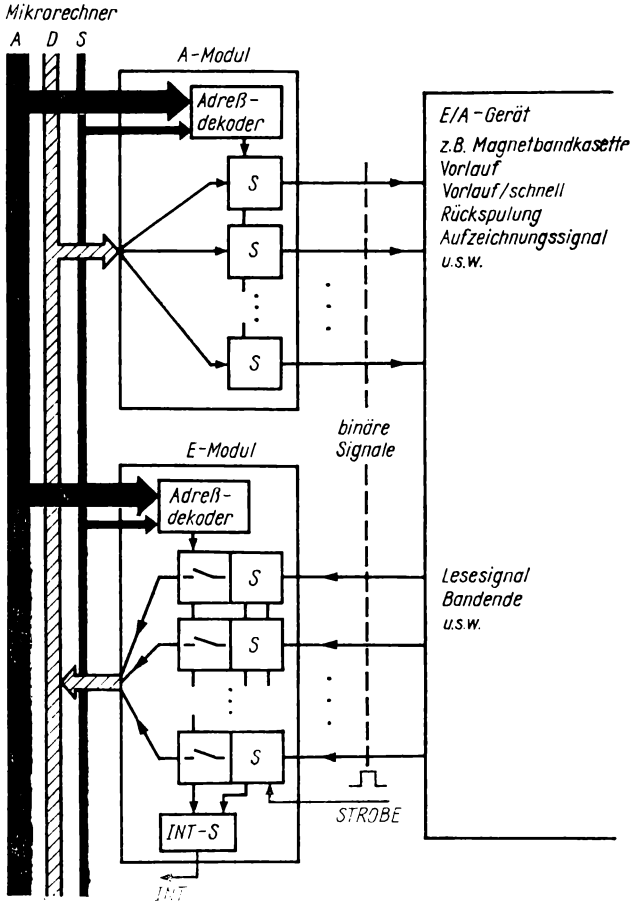


Bild 3.9. Modul für quasiparallele E/A-Schnittstelle
S Auffangspeicher, *INT* Unterbrechungsforderung

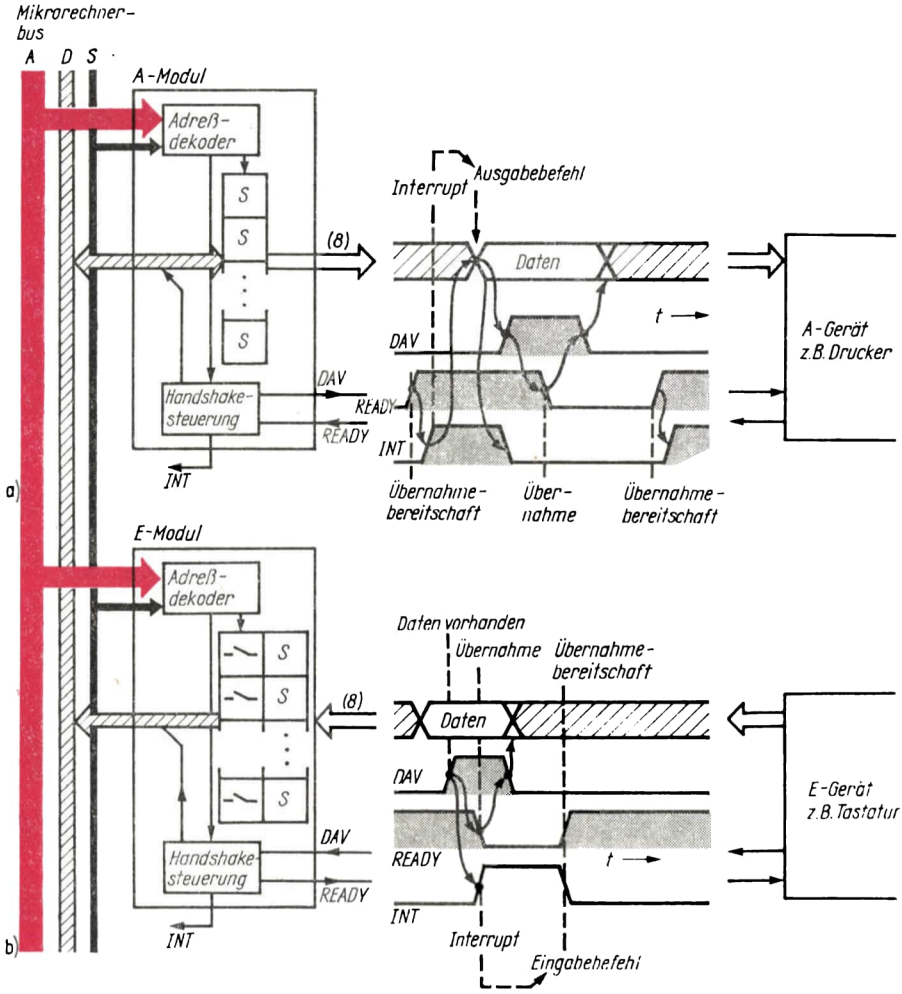


Bild 3.10. Parallele E/A-Schnittstelle (Handshakeverfahren)
a) Ausgabemodul; b) Eingabemodul

werden, wann erneute Übernahmebereitschaft besteht, damit ein weiterer Ausgabebefehl erfolgen kann. Zu diesem Zweck muß entweder der Zustand der Leitung READY vom Rechner jederzeit abtastbar sein (diese Leitung muß folglich auf einen Eingabeport geschaltet und durch einen Eingabebefehl lesbar sein), oder aber der Zustandswechsel auf dieser Leitung löst ein Unterbrechungssignal INT aus.

Ein Schaltkreis für Parallelausgabe hat also bereits einen breiteren Funktionsumfang zu realisieren: Neben der Auffangfunktion sind die Handshakesteuerung, die Bildung des INT-Signals und die Eingabefunktion für den Zustand der Handshakesteuerung zu gewährleisten.

Bild 3.10b zeigt einen parallelen Eingabemodul und das Handshake-Signalspiel am Beispiel einer Tastaturbaugruppe als Eingabeeinrichtung. Durch Betätigen

einer Taste wird ein Zeichenkode auf dem parallelen Leitungsbündel erzeugt. Darüber hinaus wird das Signal DAV (Daten vorhanden) aktiviert. Der Eingabemodul aktiviert daraufhin das Signal INT, um den Mikrorechner auf eine Eingabeforderung von der Tastatur aufmerksam zu machen. Nach Erkennen dieser Unterbrechung und Start des speziellen Unterbrechungsbehandlungsprogramms für die Tastatureingabe wird nach einer gewissen Zeit ein Eingabebefehl ausgeführt, der diesen E-Modul adressiert und das Zeichen abholt. Danach wird der E-Modul durch Aktivieren seiner READY-Leitung diese Übernahme der Tastatursignale quittieren und damit die erneute Bereitschaft (Aufforderung) zur Zeichenübernahme signalisieren.

3. Serielle Schnittstelle. Ist zwischen dem Mikrorechner und der E/A-Einrichtung eine größere Entfernung zu überbrücken, dann ist die Verwendung einer seriellen Schnittstelle zweckmäßig oder notwendig.

Notwendig ist eine solche Schnittstelle, wenn die Entfernung nur durch Übertragungswege der Fernmeldenetze überwunden werden kann. Das ist immer dann der Fall, wenn Mikrorechner und E/A-Einrichtung nicht im gleichen Gebäude

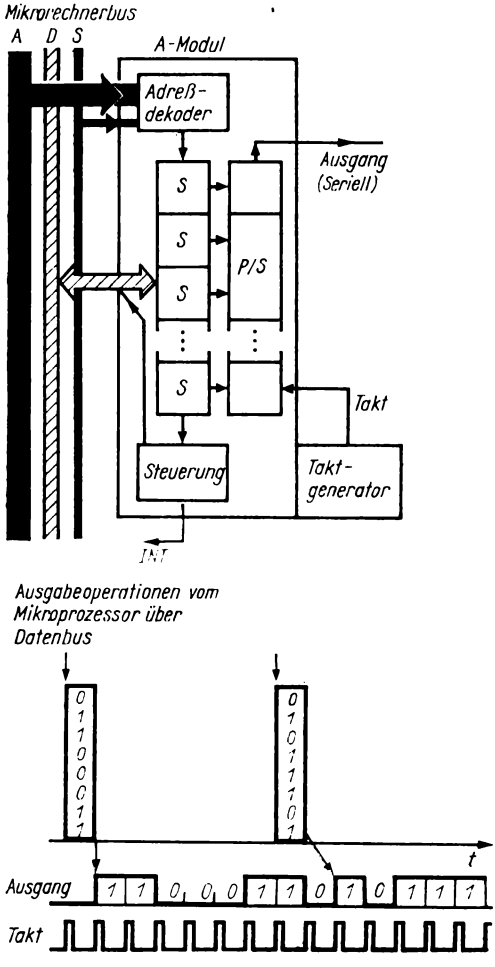


Bild 3.11. Serielle Schnittstelle (Ausgabe)

bzw. nicht innerhalb eines Betriebsgeländes stehen. Aber auch bei Kopplungen innerhalb von Gebäuden und Betriebsgeländen ist die serielle Schnittstelle zweckmäßig, da erhebliche Leitungskosten bei serieller Übertragung gespart werden können.

Allerdings erhöht sich der Aufwand bei der Schnittstellenanpassung erheblich. Betrachten wir zunächst einen seriellen Ausgabemodul (Bild 3.11): Bei einem Ausgabebefehl muß zunächst wiederum das über den Datenbus ausgegebene Wort aufgefangen werden und danach bitweise in einer festgelegten Reihenfolge und mit einem bestimmten Takt (seriell) auf eine Leitung ausgegeben werden. Auch hier stellt sich sofort die Frage, wie die Geschwindigkeitsanpassung durch den jeweils langsameren Kommunikationspartner gesteuert werden kann. Eine wortweise Steuerung nach dem Handshakeverfahren ist bei einer einzigen Schnittstellenleitung nicht realisierbar und selbst bei Einführung einer entgegengerichteten 2. Leitung nicht praktikabel. Die übliche Lösung besteht darin, daß eine Reihe von Übertragungsgeschwindigkeiten standardisiert wurde (z. B. 300, 600, 1 200, 2 400 bit/s), wovon für den konkreten Fall eine geeignete Geschwindigkeit ausgewählt werden muß. Eigene Taktgeneratoren im E/A-Modul und in der entfernten E/A-Einrichtung sorgen für die Einhaltung dieser Übertragungsgeschwindigkeiten. Die Übertragungsrate wird auf den langsameren Kommunikationspartner ausgerichtet bzw. ist oftmals auch durch den Übertragungsweg begrenzt. Auf jeden Fall ist sie in der Regel wesentlich kleiner als die Ausgabegeschwindigkeit des Mikrorechners. Entsprechend ist auch hier eine Meldung an den Prozessor erforderlich, wenn ein neues Ausgabewort nachfolgen darf. Diese Meldung wird aber im Unterschied zur parallelen Schnittstelle nicht vom eigentlichen Empfänger, sondern vom Schnittstellenmodul selbst erzeugt. Dabei stehen wiederum die beiden Möglichkeiten zur Auswahl: Erzeugung eines Unterbrechungssignals oder Abfrage des Zustandes durch einen Eingabebefehl. Die serielle Übertragung erfordert aber noch die Lösung weiterer Probleme, z. B.:

- Synchronisation zwischen Sender und Empfänger (Angleichung der entfernt stehenden Taktgeneratoren und Erkennen von Anfang und Ende eines seriell übertragenen Binärwortes)
- Erkennen von Übertragungsfehlern und entsprechende Maßnahmen zur Beseitigung dieser Fehler.

Dazu sind eine Reihe weiterer Festlegungen über die Form des Informationsaustausches zu treffen, die als **Datenübertragungsprozeduren** bzw. **-protokolle** bezeichnet werden. Das historisch 1. und einfachste Verfahren ist dabei das **Start-Stop-Verfahren** (Bild 3.12a). Bei diesem Verfahren wird Anfang bzw. Ende eines Zeichens durch 1 Startbit (0-Zustand) bzw. durch 1 oder 2 Stopbits (1-Zustand) gekennzeichnet. Damit ist die Synchronisation zwischen Sender und Empfänger gewährleistet, und die Zeichen können in asynchroner Folge (beliebige Lücken zulässig) übertragen werden.

Den Nachteil dieses Verfahrens – nämlich die Notwendigkeit der vielen Bits für Synchronisierzwecke (vor und nach jedem Zeichen) – vermeidet das **Synchronverfahren** (BSC Bisync-Verfahren) (Bild 3.12b). Die Bits werden hierbei lückenlos übertragen, und damit existiert keine Kennung von Anfang bzw. Ende eines jeden Wortes (Zeichen) mehr. Es wird deshalb ein spezielles Zeichen, das Synchronzeichen, vereinbart, das zum Einrasten des Empfängers an den Anfang der Übertragung gestellt wird und außerdem dazwischen zum Auffüllen von Lücken benutzt wird.

Die bisher betrachteten Verfahren beruhen auf einer zeichenweisen Übertragung. Eine Bildung von größeren Dateneinheiten (Blöcken) und deren gesicherte Übertragung muß durch weitere Steuerzeichen (Blockanfang, Blockende usw.) vorgenommen werden. Zukünftig werden sich jedoch Prozeduren durchsetzen, die von vornherein blockorientiert sind, z. B. das HDLC- oder SDLC-Verfahren

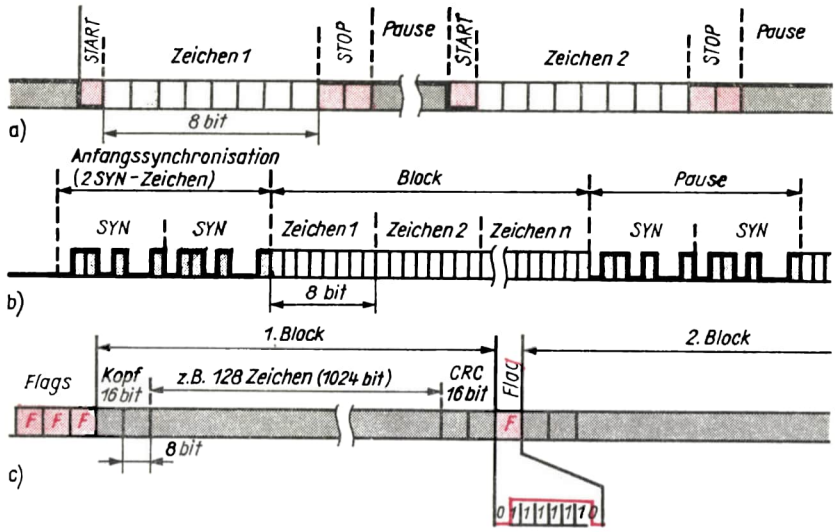


Bild 3.12. Datenübertragungsverfahren

a) Start-Stop-Verfahren; b) Synchronverfahren; c) HDLC-(SDLC-)-Verfahren
***CRC* Prüfinformation; *SYN* Synchronisierzeichen**

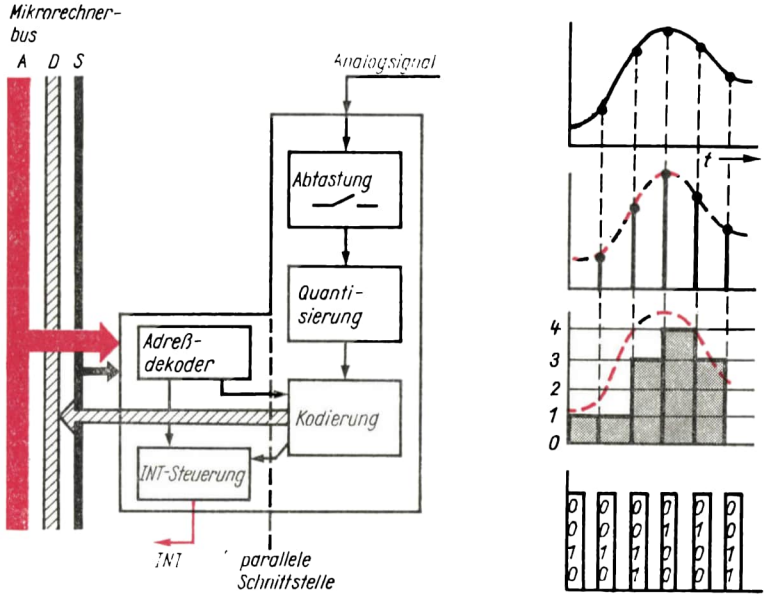


Bild 3.13. Analoge Schnittstelle (Eingabemodul)

(Bild 3.12c). Ein einziges Steuerzeichen, das Flag, wird bei diesem Verfahren vereinbart. Es dient als Kennung für Anfang und Ende eines Blockes und als Lückenfüller zwischen den Blöcken. Die Blöcke bestehen aus etwa 1000 Bits und werden mit Sicherungsinformation versehen.

4. Analoge Schnittstelle. Analoge Eingangs- oder Ausgangssignale sind mit der rechnerinternen Informationsdarstellung unverträglich und bedürfen einer Analog-Digital- bzw. Digital-Analog-Umsetzung.

Betrachten wir zuerst den prinzipiellen Aufbau eines Eingabemoduls für analoge Signale (Bild 3.13):

Das Analogsignal weist in unendlich dichter Folge Signalwerte mit (zumindest theoretisch) unendlich feiner Abstufung auf. Die im **Analog-Digital-Umsetzer** erfolgende Wandlung in ein digitales Signal kann nur in einer Informationsreduktion in beiden Koordinaten bestehen. Aus dem Signalverlauf werden im Abstand ΔT Proben entnommen und die dabei gemessenen Signalwerte quantisiert, d. h. einem mehr oder weniger groben Raster mit N unterschiedlichen Stufen zugeordnet. Aus dem analogen Signal entsteht damit scheinbar intern ein stufenförmiges Zwischensignal, das in einer nachfolgenden Kodierstufe in ein Kodewort mit n Bitstellen umgesetzt wird und das eigentliche Ausgangssignal des AD-Umsetzers bildet. Dabei gilt wiederum, daß mit n bit 2^n verschiedene Stufen kodiert werden können (z. B. mit einem 8-bit-Wort 256 Amplitudenstufen). Der AD-Umsetzer benötigt dazu, abhängig von seiner elektronischen Realisierung, eine bestimmte Zeit (Transversionszeit).

Die Kenngrößen eines AD-Umsetzers sind damit:

Transversionszeit: Bestimmt die obere Grenze der Abtastdichte des Analogsignals und damit die erreichbare Genauigkeit der Erfassung der Signaldynamik.
Wortbreite (Genauigkeit): Bestimmt die Anzahl der Quantisierungsstufen des Analogsignals und damit die Genauigkeit bei der Umsetzung der Signalwerte.

Beim Systementwurf ist es daher erforderlich, die jeweils notwendigen Genauigkeiten abzuschätzen und danach die geeigneten AD-Umsetzer auszuwählen. Es ist selbstverständlich, daß der technische Aufwand und die Kosten dieser Umsetzer mit sinkender Transversionszeit und steigender Stufenanzahl wachsen und für diese Kennwerte auch technische Grenzen existieren. Inzwischen wird jedoch ein Sortiment solcher Schaltkreise produziert, das für die meisten Anwendungsfälle genügt. Nur für extreme Forderungen werden Sonderanfertigungen erforderlich. Dabei muß natürlich beachtet werden, daß der Mikrorechner auch nur mit begrenzter Geschwindigkeit die vom AD-Umsetzer produzierten Kodewörter übernehmen kann und auch von dieser Seite Auflösungsgrenzen gesetzt werden.

Die Funktion des AD-Umsetzers besteht also letztlich darin, dem Mikrorechner anstelle der analogen eine parallele Schnittstelle vorzutäuschen (Bild 3.13).

Einfacher ist die inverse Signalwandlung, die Erzeugung eines analogen Ausgangssignals aus einer Folge vom Rechner ausgegebener Binärwörter. Der entsprechende Ausgabemodul enthält einen **Digital-Analog-Umsetzer** (Bild 3.14). Das durch einen Ausgabebefehl übergebene Binärwort wird aufgefangen und auf elektronischem Wege in ein N-stufig quantisiertes Digitalsignal umgewandelt. Da die Ausgabedichte durch das Ausgabeprogramm und letztlich durch die endliche Arbeitsgeschwindigkeit des Rechners begrenzt ist, entsteht ein in beiden Koordinaten diskretes Signal. (Die Bezeichnung Digital-Analog-Umsetzer ist damit strenggenommen falsch, da die Funktion eigentlich nur auf die Dekodierung, also auf die Umsetzung des parallelen Binärwortes in ein mehrstufiges Signal, begrenzt ist.) Erst durch ein nachgeschaltetes Filter läßt sich dieses Signal glätten und damit ein analoges Signal vortäuschen. Oftmals ist dies jedoch nicht erforderlich, da die nachfolgende Ausgabeeinrichtung ohnehin eine solche Glättung aufgrund ihrer Trägheit vornimmt.

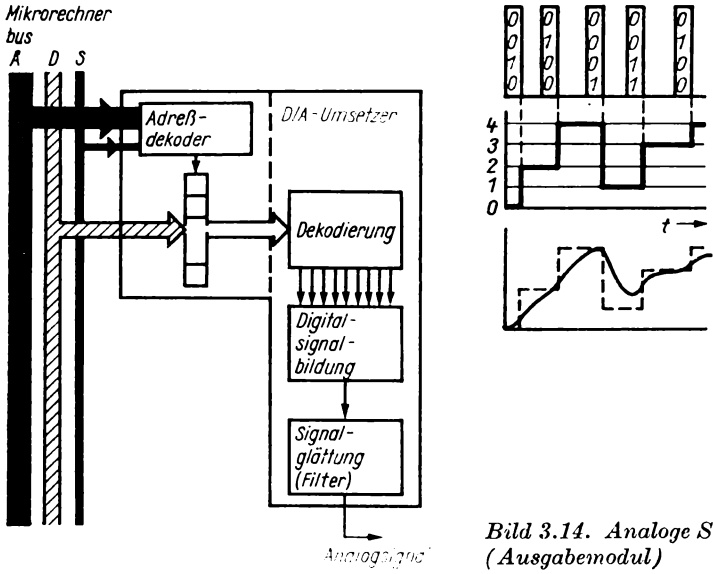


Bild 3.14. Analoge Schnittstelle (Ausgabemodul)

Programmierbare E/A-Module

Die bisherigen Betrachtungen haben gezeigt, daß trotz Standardisierungsbemühungen eine beträchtliche Anzahl unterschiedlicher Forderungen bei der Kopplung Rechner–Umwelt zu beachten sind und damit von den Schaltkreisherstellern der Kompromiß zwischen Sortiments- und Funktionsumfang zu schließen ist. Eine häufig praktizierte Lösung besteht darin, E/A-Schaltkreise herzustellen, die zwar einen bestimmten Grundtyp repräsentieren (z. B. parallele oder serielle Schnittstelle), aber noch auf verschiedene Betriebsarten einstellbar (programmierbar) sind. Diese Einstellung wird dabei so realisiert, daß der Modul intern Steuerregister besitzt, die erst durch Ausgabebefehle geladen werden müssen, um den Schaltkreis auf eine konkrete Funktion festzulegen. Auf diese Weise wird dem Systementwerfer die Möglichkeit geboten, mit einem begrenzten Schaltkreissortiment einen breiteren Anwendungskreis zu erfassen und auch eine Funktionsumschaltung während der Programmabarbeitung zu ermöglichen. Vom Rechner aus gesehen, haben diese E/A-Module damit 2 „Gesichter" (dementsprechend auch 2 verschiedene Adressen), zum einen die normale Adresse für die E/A-Funktion und zum anderen eine Adresse für das (die) Steuerregister zur Schaltkreisprogrammierung.

3.1.5. Zähler/Zeitgeber-Modul

Der Einsatz zur Steuerung von Echtzeitsystemen erfordert nahezu immer, daß der Steuerrechner über „Zeitgefühl" verfügt, d. h., daß er beispielsweise in der Lage ist:

- einmalig zu einem bestimmten Zeitpunkt oder periodisch in definierten Zeitabständen Reaktionen zu veranlassen (Start spezieller Programme);
- den zeitlichen Abstand zwischen 2 äußeren Ereignissen zu messen usw.

Einfach ausgedrückt, benötigt der Rechner je nach Einsatzfall eine interne Uhr, auf der er zu jedem Zeitpunkt die Tageszeit „ablesen" kann, ein oder mehrere

Stoppuhren, mit denen er Zeitintervalle messen kann, und ein oder mehrere Wecker, die er nach Bedarf stellen kann und die ihn nach Ablauf der Zeitintervalle „wecken", um den Start bestimmter Programmteile zu veranlassen.
Die **Erzeugung von Zeitdifferenzen** (Totzeiten, Wartezeiten) läßt sich softwaremäßig realisieren. Man nutzt dabei den Umstand, daß die Ausführungszeiten der Befehle exakt bekannt und aufgrund des frequenzstabilisierten Prozessorgrundtaktes hochstabil sind. Es lassen sich nun solche Befehlsfolgen geringen Umfangs programmieren, mit denen in weiten Grenzen variierbare Programmlaufzeiten (vom Mikrosekunden- bis Stunden- und bis Tagebereich) eingestellt werden können. Gelangt der Rechner bei der Abarbeitung seines Programms an ein solches Zeitprogramm, dann wird er für eine bestimmte Zeit mit dessen Ausführung „beschäftigt"; legt also scheinbar eine Pause ein, bevor er das Teilprogramm verläßt und die Programmabarbeitung fortsetzt. Nachteilig ist, daß der Rechner während dieser Zeit keinerlei andere Aufgaben ausführen kann und darf (also beispielsweise auch nicht auf Unterbrechungen reagieren darf!), da sonst keine exakte Zeitdauer eingehalten wird. Weiterhin kann auf diese Weise jeweils nur eine Totzeit erzeugt werden.

Die **Realisierung von Zeitmessungen,** also sowohl die Messung von Zeitintervallen (Stoppuhr) als auch die Messung der absoluten Tageszeit, erfordert den Einsatz zusätzlicher Hardware. Eine mit geringem Hardwareaufwand verbundene Methode sieht wie folgt aus: Ein separater externer Taktgenerator liefert einen frequenzstabilisierten Basistakt. Die Taktrate wird dabei gerade so hoch gewählt, wie es die geforderte Zeitauflösung erfordert (also z. B. Millisekundentakt). Dieser Takt bildet für den Rechner ein Eingangssignal (Unterbrechungssignal). Im Abstand der Taktflanken wird die Arbeit des Rechners unterbrochen und der Sprung in das spezielle Unterbrechungsbehandlungsprogramm UHRPROGRAMM veranlaßt. Dieses Programm bewirkt, daß der Inhalt von ausgewählten, für die Aufbewahrung der Uhrzeit vorgeschenen Speicherplätzen weitergezählt wird. Für dieses Weiterrücken der Uhr sind nur wenige Befehle erforderlich, so daß damit der Rechner kaum belastet wird, vorausgesetzt, die verwendete Zeittaktperiode ist mindestens 100···1000mal größer als die des Prozessortaktes. Auf diese Weise ist sichergestellt, daß auf festen Speicherplätzen im Datenspeicher oder Registersatz des Prozessors jederzeit durch andere Programme die genaue Zeit „abgelesen" werden kann.

Um den Rechner nun noch weiter von diesen Hilfsarbeiten zu befreien, kann die Hardware durch spezielle **Zeitgeber** erweitert werden. Diese Einrichtungen bestehen im wesentlichen aus einem **Zählerbaustein,** also einer elektronischen Einrichtung, die selbständig den Inhalt eines Speicherplatzes um 1 weiterzählt, wenn ein Taktimpuls auftritt. Wird nun ein solcher Zählerbaustein dahingehend erweitert, daß er an den MR-Bus anschließbar und folglich wie ein Speicherplatz jederzeit schreib- und lesbar ist, so steht damit eine Uhr zur Verfügung, deren Fortbewegung den Rechner überhaupt nicht mehr belastet (Bild 3.15a). (Der Unterschied zu oben besteht also nur darin, daß der Zeittakt nicht mehr eine Unterbrechung des Rechners auslöst, sondern direkt den Inhalt des Zeitregisters verändert, und daß dieses Zählregister nicht im Datenspeicher, sondern in einem speziellen Zeitgebermodul angeordnet ist, der wie ein E/A-Port adressiert und gelesen werden kann.)

Über das betrachtete Grundprinzip hinaus, verfügen nun die mikroelektronisch realisierten **Zähler/Zeitgeber-Schaltkreise (Counter/Timer)** über eine Reihe zusätzlicher Leistungsmerkmale, die bei der Systeminitialisierung durch spezielle Steuerworte jeweils festgelegt (programmiert) werden können. Bild 3.15b zeigt als Beispiel einen solchen Schaltkreis, der folgende Merkmale aufweist:

– Als Taktquelle kann wahlweise ein externer Takt oder der durch einen Vorteiler reduzierte Prozessorgrundtakt verwendet werden, wobei der Vorteiler auf verschiedene Werte einstellbar ist.

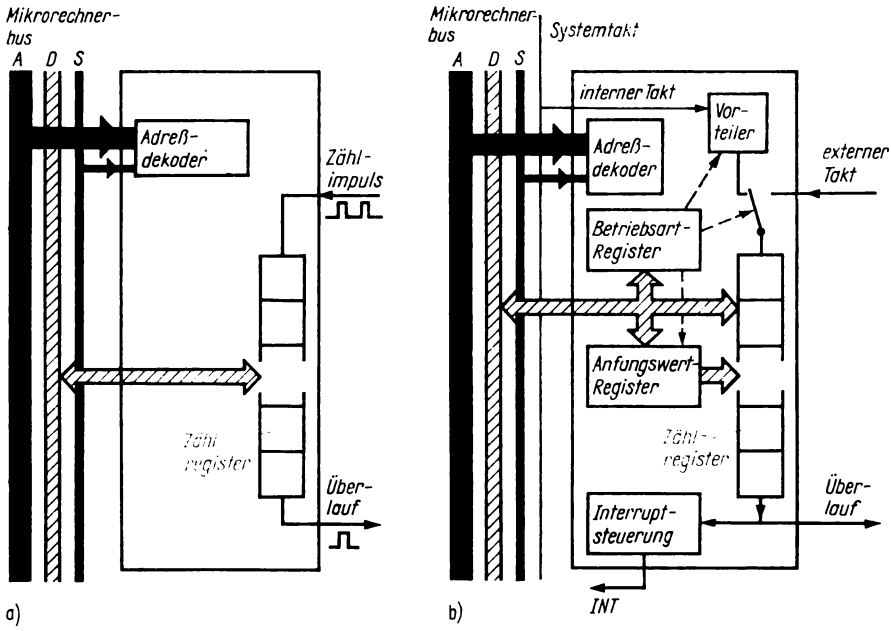


Bild 3.15. Zähler/Zeitgeber-Modul
a) Prinzip; b) Zähler/Zeitgeber-Schaltkreis (Beispiel)

– Erreicht der Zähler seinen Endstand, so gibt er einen Impuls nach außen ab, erzeugt ein Unterbrechungssignal und beginnt danach wahlweise wieder mit einem vereinbarten Anfangswert oder bleibt stehen.
– Der Anfangswert kann beliebig vorgegeben (eingeschrieben) werden.

Mit Hilfe eines solchen Moduls läßt sich eine Vielfalt unterschiedlicher Aufgaben lösen. Betrachten wir einige Beispiele:

1. *Weckerfunktion*: Durch einen Ausgabebefehl wird das Zählregister mit einem Anfangswert geladen und danach für die Taktung mit dem durch den Vorteiler reduzierten Systemtakt freigegeben. Nach einer vom Anfangswert und von der Vorteilereinstellung abhängigen Zeit (Weckzeit) erreicht der Zähler seinen Endwert und informiert darüber den Mikroprozessor durch ein Unterbrechungssignal.
2. *Taktgenerator*: Häufig werden im Rechner- bzw. im Gesamtsystem noch weitere Takte benötigt (z. B. Takt für serielle Schnittstellen). Um dafür separate Generatoren einzusparen, können die Ausgänge der Zeitgebermodule verwendet werden. Die Programmierung erfolgt wie bei 1., nur das INT-Signal wird nicht angeschlossen und der Baustein so programmiert, daß bei Überlauf des Zählers der Anfangswert erneut geladen wird. Der Zeitgeber arbeitet damit nach seinem Start vollkommen autonom und erzeugt eine periodische Impulsfolge, deren Frequenz aber jederzeit durch Laden anderer Parameter in den Zeitgeberschaltkreis verändert werden kann.
3. *Stoppuhr*: Der Zeitgeber wird auf 0 gesetzt und gestartet. Bei Eintreten des zu stoppenden Ereignisses wird durch einen Lesebefehl der Inhalt des Zählers gelesen, der dementsprechend ein Maß für die inzwischen abgelaufene Zeit ist.

Der Zählbereich dieser Zählregister ist oft der Datenbusbreite angepaßt und damit eng begrenzt (z. B. bis 256 bei 8-bit-Systemen). Die Erweiterung ist jedoch einfach durch Kaskadierung möglich, indem der Ausgang des vorhergehenden Zeitgebers den Takteingang eines folgenden bildet.

3.1.6. Spezialmodule

Das gemeinsame Merkmal aller bisher betrachteten passiven Module ist, daß sie für den Mikroprozessor Kommunikationspartner sind, die mit Adressen angesprochen und mit denen Datenwörter ausgetauscht werden können. Die Unterschiede bestehen in den Operationen, die mit diesen Daten erfolgen. In den Speichermodulen werden sie unverändert aufbewahrt, in den A-Ports werden sie aufgefangen und zugleich auf nach außen führende Leitungen gelegt, oder sie werden in den Modulen in spezielle Steuerregister geladen, deren Ausgangsleitungen die interne Arbeitsweise des Moduls festlegen, um eine von mehreren Varianten einzustellen (programmierbare Module).

Es stellt sich nun die Frage, ob es weitere nützliche Operationen gibt, für die spezielle passive Module entwickelt und eingesetzt werden können. Ein naheliegendes Problem ist die Ausführung von arithmetischen Operationen, z. B. das Berechnen von Funktionen mit einem Operanden (e^x, sin x usw.) bzw. das Ausführen von höheren Rechenoperationen (Multiplikation, Division, Potenzfunktion usw.). All diese Aufgaben könnten prinzipiell durch Programme mit der Arithmetik/Logik-Einheit des Zentralprozessors realisiert werden (indem die Funktionen auf die elementaren, im Befehlssatz des Prozessors enthaltenen Befehle, z. B. Addition und Subtraktion, zurückgeführt werden). Dann ist aber mit beträchtlichen Ausführungszeiten zu rechnen. Wird dagegen ein spezieller Modul eingesetzt, der eine bestimmte arithmetische Funktion (z. B. sin x) schnell ausführen kann, dann ist folgende Arbeitsweise möglich: Der Operand x wird durch einen Schreibbefehl diesem „Sinusmodul" übergeben, der den Funktionswert sin x errechnet und in einem Ergebnisregister bereitstellt. Durch einen darauf folgenden Lesebefehl wird das Ergebnis in den Mikroprozessor geholt. Anstatt die Sinusfunktion durch Reihenentwicklung im Prozessor selbst zu berechnen (Sinusprogramm!), besteht das Programm nun nur noch aus 2 Transportbefehlen (Schreiben und Lesen des Spezialmoduls). Hinsichtlich der Rechenzeit, die dieser Modul benötigt, sind 2 Fälle zu unterscheiden: Wird die Funktionsermittlung hardwaremäßig gelöst, dann steht das Ergebnis sofort zur Verfügung. Wird dagegen innerhalb des Moduls wiederum ein Rechner (spezialisierter Einchiprechner) eingesetzt, dann sind mehrere Schritte erforderlich, bis der Funktionswert bereitsteht. In diesem Fall kann das Ergebnis nicht sofort gelesen werden. Es muß erst auf eine Fertigmeldung des Moduls gewartet werden (Abfragen durch Warteschleife bzw. Unterbrechungssignal).

Zum Standardumfang einiger Mikrorechner-Schaltkreissätze gehören inzwischen die **Arithmetikprozessoren,** die für diesen Aufgabenkreis entwickelt wurden. Der Unterschied zu dem zuvor betrachteten Sinusprozessor besteht nur darin, daß sie für die Ausführung einer Menge von Operationen ausgelegt sind und deshalb vom Mikroprozessor außer dem (den) Operanden noch zuvor ein Steuerwort übergeben werden muß, mit dem die auszuführende Operation ausgewählt wird (Bild 3.16). Mit einem solchen passiven Modul können damit die für Zahlenverarbeitung schlecht ausgerüsteten Mikroprozessoren ergänzt und die Leistungsfähigkeit des Mikrorechners auf dieser Ebene beträchtlich gesteigert werden.

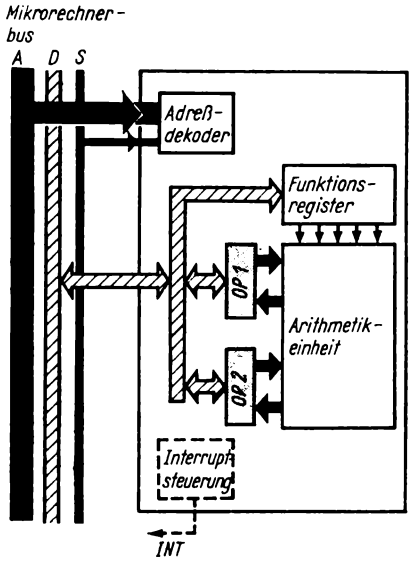


Bild 3.16. Arithmetikprozessor OP 1,2 Operanden- und/oder Resultatregister

3.1.7. Unterbrechungssystem

Die Steuerung von Echtzeitsystemen erfordert vom Steuerrechner, daß bei Eintreten bestimmter Ereignisse in der Umwelt Sofortreaktionen ausgelöst werden. Dazu wäre notwendig, eine Menge von Meldeleitungen ständig zu überwachen. Wir haben bereits gesehen, daß eine solche Überwachung durch periodische Abfrage (polling) mit Hilfe von Eingabebefehlen erfolgen kann. Es ergeben sich dann aber zwangsweise nur endliche Abfrageabstände, die insbesondere dann zu groß werden können, wenn ein Ereignis erkannt wird, der Rechner die erforderliche Reaktion bearbeitet und damit für eine bestimmte Zeit die Abfrage unterbricht. Auf jeden Fall wird für diese Abfragen u. U. ein nicht unbeträchtlicher Teil der Prozessorleistung verbraucht. Eine günstigere Lösung kann daher nur darin bestehen, diese Aufgabe hardwaremäßig zu implementieren. Der dazu benötigte Teil des Zentralprozessors wird dementsprechend als Unterbrechungssystem und die Signale auf den überwachten Leitungen als Unterbrechungssignale (Interruptsignale) bezeichnet.

Betrachten wir zunächst, wodurch Unterbrechungsforderungen entstehen können und welche prinzipielle Reaktion daraufhin erfolgen muß. Entsprechend der Quellen für Unterbrechungssignale unterscheidet man:

1. **externe Unterbrechungen**: direkte Mitteilung über binäre Ereignisse aus der Umwelt (Alarme)

2. **interne Unterbrechungen**: Fertigmeldung bzw. Bedienanforderungen der passiven Module (E/A-Modul, Zeitgeber usw.)

3. **Ausnahmezustände** innerhalb des Prozessors (traps): falsche Befehle, unzulässige Operanden (Division durch 0) usw.

Die unter 1. und 2. genannten Unterbrechungen entstehen i. allg. asynchron zum Rechnertakt und bilden letztlich eine Menge von Eingangssignalen für den Prozessor (Bild 3.1).

Als Reaktion auf eine Unterbrechungsforderung muß das Unterbrechungssystem prinzipiell folgendes veranlassen:

- schnellstmögliche Unterbrechung der laufenden Programmbearbeitung so, daß dieses Programm später ohne Informationsverlust fortgesetzt werden kann;
- Start eines für die Unterbrechungsursache spezifischen Unterbrechungsbehandlungsprogramms (UBP).

Betrachten wir nun, wie diese Aufgaben bei Mikroprozessoren gelöst werden:

Als Nadelöhr bei der Realisierung des kompletten Unterbrechungssystems auf dem Mikroprozessorschaltkreis erweist sich wiederum die Begrenzung der Anzahl der Anschlüsse. So ist es nicht möglich, ausreichend viele Eingänge vorzusehen, um jede Unterbrechungsquelle direkt anzuschließen.

Als Kompromiß verfügt der Mikroprozessor deshalb meist nur über 2 oder 3 Unterbrechungseingänge, wovon immer ein Eingang ein sog. **Sammelunterbrechungseingang** (vektorisierter Interrupt) ist. Das Prinzip der Sammelunterbrechung besteht darin, daß alle Unterbrechungsquellen auf diesen einen Eingang letztlich zusammengeführt werden. Der Mikroprozessor wird damit bei Auftreten einer Unterbrechungsforderung seine laufende Programmabarbeitung nach kompletter Ausführung des laufenden Befehls beenden, hat aber zu diesem Zeitpunkt noch keine Information über die konkrete Unterbrechungsursache und kann folglich auch nicht sofort ein UBP starten. Der Mikroprozessor muß sich vielmehr zunächst nach der Ursache erkundigen, indem er eine Kommunikation mit einem speziellen Modul, dem **Unterbrechungssteuerungsmodul** (interrupt controller), erzwingt. Der Mikroprozessor gibt zu diesem Zweck eine spezielle Adreßkombination (Unterbrechungsbestätigung, interrupt acknowledge) aus und erhält daraufhin vom Interruptsteuermodul über den Datenbus ein Kodewort (sog. **Unterbrechungsvektor**) übertragen, das eindeutig die Unterbrechungsquelle kennzeichnet. Damit kann der Mikroprozessor nun die Startadresse des zutreffenden UBP ermitteln, in seinen Programmzähler eintragen und dieses Programm starten.

Für den Sammelunterbrechungseingang existiert im Mikroprozessor ein Hauptschalter, mit dem es möglich ist, Unterbrechungen generell zu ignorieren. Die Betätigung dieses Schalters erfolgt durch 2 Befehle (s. Musterbefehlssatz, Tafel 2.1d).

Wir können damit feststellen, daß bei diesen mikroprozessororientierten Systemen das Unterbrechungssystem nur teilweise in den Mikroprozessorschaltkreis integriert werden kann. Sobald in dem entworfenen Mikrorechner mehr Unterbrechungsursachen als Unterbrechungseingänge vorhanden sind, muß ein Interruptsteuermodul mit folgendem Funktionsumfang eingefügt werden (Bild 3.17):

1. Bildung des Sammelunterbrechungssignals
2. Bereitstellung des Unterbrechungsvektors und Ausgabe an den Datenbus, wenn der Mikroprozessor diesen nach Erkennen der Sammelunterbrechung abfragt.
3. Bestimmung der Rangfolge (Priorität), wenn mehr als eine Unterbrechungsforderung gleichzeitig anliegt.
   (Tritt dieser Fall auf, muß von dieser Baugruppe entschieden werden, welcher Vektor zuerst übergeben wird. Die Rangfolge kann fest eingestellt (verdrahtet) sein, es gibt aber auch Lösungen, bei denen wiederum über Steuerregister diese Rangfolge eingestellt wird und damit im laufenden Programm veränderbar ist.)
4. Maskierung der Unterbrechungseingänge, d. h., die Möglichkeit, einzeln diese Eingänge abschalten zu können. (Zu diesem Zweck ist ein Register erforderlich, das vom Programm mit der **Unterbrechungsmaske** geladen wird, von der jeweils eine Bitstelle für einen Eingang zuständig ist und über Zuschaltung und Abschaltung entscheidet. Damit ist die selektive Auswahl der im konkre-

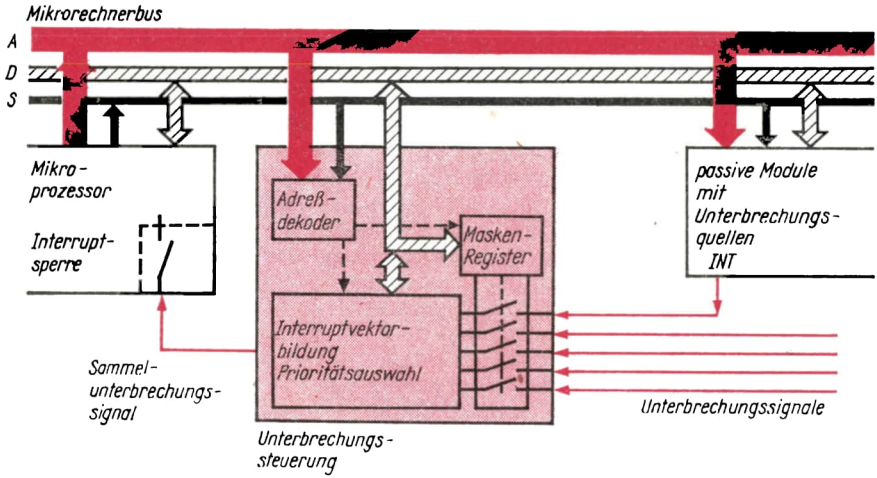


Bild 3.17. Unterbrechungssystem/Realisierung mit Hilfe eines Unterbrechungssteuermoduls

ten Zeitpunkt zugelassenen Unterbrechungssignale durch das Programm möglich.)

Die Realisierung einer solchen Baugruppe für die Unterbrechungssteuerung wird seitens der Mikrorechnerschaltkreise durch 2 unterschiedliche Konzepte unterstützt:

1. separater Schaltkreis (interrupt controller)
2. verteilte Unterbrechungssteuerung (weitestgehende Integration der notwendigen Funktionen in die Unterbrechungsquellen).

Die 1. Form ist im Bild 3.17 gezeigt. Die Unterbrechungssteuerung bildet damit einen separaten passiven Modul, der durch einen Schaltkreis realisiert wird. Der Nachteil dieses Konzepts ist, daß ein weiterer Schaltkreis angeordnet werden muß und daß wiederum nur eine begrenzte Anzahl von Unterbrechungsquellen anschaltbar sind.

Bei der 2. Variante wird das vermieden, indem von vornherein auf jedem Schaltkreis, der Unterbrechungsquellen enthält (z. B. E/A-Schaltkreise, Zeitgeber usw.), jeweils eine eigene Baugruppe zur Unterbrechungssteuerung integriert wird. Funktionsmäßig bedingt können aber nur die Bildung des Unterbrechungsvektors und die Maskierung unmittelbar an die Quelle verlagert werden, während die Prioritätsauswahl und die Bildung der Sammelunterbrechung nur übergeordnet gelöst werden können.

Das heißt, es wird auf jeden Fall eine Zusammenschaltung dieser einzelnen verteilten Unterbrechungssteuerungen notwendig.

Bild 3.18 zeigt das für diesen Zweck häufig verwendete Prinzip der starren **Prioritätskette (Daisy-chain-Prinzip)**. Jede verteilte Unterbrechungssteuerung verfügt dabei über 3 Anschlüsse. Ein Ausgang (INT) ist die eigentliche Unterbrechungsmeldung, die entsprechend mit allen weiteren zum Sammelunterbrechungssignal vereinigt wird (logische ODER-Verknüpfung). Je ein Eingang *IE* und ein Ausgang *IA* dienen der Bildung einer Rangfolge nach folgendem Prinzip:

Bei Auftreten einer Unterbrechungsforderung erfolgt das Aktivieren der Ausgänge *INT* und *IA*. Der Mikroprozessor wird sofort über seinen Sammelunterbrechungseingang informiert und, falls keine Unterbrechungssperre eingeschaltet ist, seine Arbeit unterbrechen. Danach wird durch den Mikroprozessor eine Leseoperation für den Unterbrechungsvektor ausgelöst. Dieser Lesevorgang kann

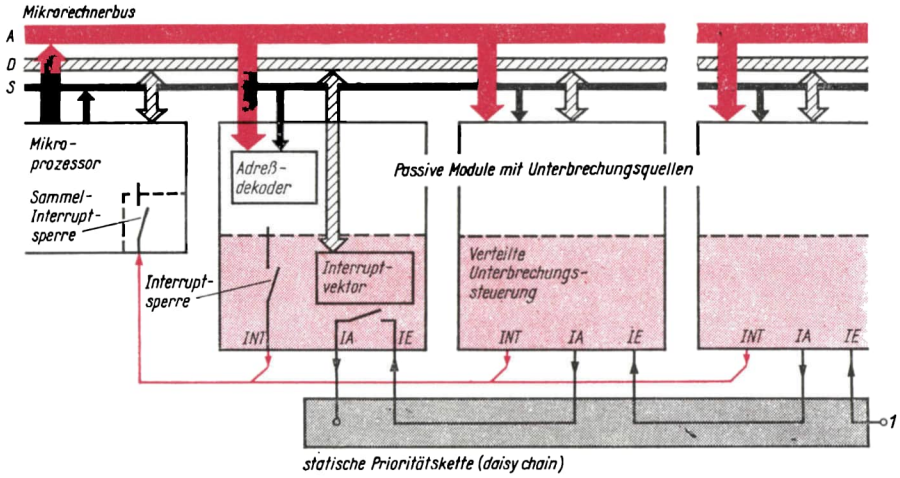


Bild 3.18. Unterbrechungssystem mit verteilter Unterbrechungssteuerung

aber nun nicht einem bestimmten Modul gelten, sondern muß eine allgemeine Aufforderung an alle Unterbrechungsquellen sein. Zu diesem Zweck ist eine Adresse (Unterbrechungsbestätigungsadresse) vereinbart, auf die alle Module reagieren können. Damit nun sichergestellt wird, daß nur ein, und zwar der ranghöchste Modul auf diese Leseoperation antwortet, werden bei diesem Daisy-chain-Prinzip alle Module gesperrt, deren Eingang *IE* aktiv ist. Solange noch keine Forderungen entstanden sind, sind alle Eingänge *IE* inaktiv. Sobald jedoch ein Modul Unterbrechung fordert, aktiviert er seinen Ausgang *IA*, und wenn dieser gemäß Bild 3.18 mit dem Eingang des folgenden verschaltet ist (daisy-chain Gänseblümchenkette!), sperrt er damit alle rangmäßig unter ihm liegenden Module. Auf diese Weise wird eine verdrahtete Rangfolge realisiert. Nach kompletter Ausführung des UBP sorgt am Ende dieses Programms ein spezieller Rückkehrbefehl dafür, daß diese Sperre der darunterliegenden Bausteine wieder aufgehoben wird (der Ausgang *IA* wird wieder inaktiv).

3.1.8. Realisierung des Statusspeichers

Der Start eines für jede Unterbrechungsursache spezifischen Unterbrechungsbehandlungsprogramms (UBP) ist nur die eine Hälfte der zu leistenden Arbeit. Um das unterbrochene Programm später fortsetzen zu können, ist die Rettung des Prozessorzustandes vorzunehmen.

Unter **Prozessorzustand (Status)** wollen wir die Gesamtheit der Registerinhalte verstehen, die zum Zeitpunkt der Unterbrechung vorliegt. Durch den Programmwechsel muß der Inhalt des Programmzählers und durch die nachfolgende Abarbeitung des UBP können die Inhalte der übrigen Register verändert werden.

Ideal wäre es, wenn die Rettung des Prozessorstatus vom Unterbrechungssystem vollständig mit übernommen würde. Meist wird jedoch nur ein Teil hardwaremäßig realisiert (z. B. nur Rettung des Programmzählers). Die übrigen Register müssen durch spezielle Transportbefehle gerettet werden, die damit am Anfang jedes UBP wiederholt zu programmieren sind.

Unabhängig von der hardwareseitigen Ausbaustufe dieses „Rettungsmechanismus" ist immer ein entsprechender Speicherraum erforderlich. Funktionell ist hierfür ein Speicher vom LIFO-Typ am besten geeignet, da nur die Aufgabe besteht, einen Stapel von Binärwörtern (Registerinhalte) für eine kurze Zeit „zur Seite zu legen (einzukellern)", um ihn danach geschlossen wieder in die Register einzuordnen. Ein LIFO-Speicher (Kellerspeicher, **Stapelspeicher, Stack**) übernimmt genau diese Aufgabe, indem bei Schreiboperationen die Daten in der Reihenfolge ihres Eintreffens gespeichert und bei Leseoperationen das zuletzt geschriebene Wort ausgelesen und die übrigen nachgerückt werden. Es gilt nur zu beachten, daß die Reihenfolge des Stapels beim Lesen umgekehrt wird.

Im Bild 3.5 haben wir bereits den prinzipiellen Aufbau eines LIFO-Speichers gesehen: Er besteht aus einer begrenzten Anzahl von Speicherplätzen und einer Zugriffssteuerung, die die jeweilige nächste Schreib- bzw. Leseadresse in einem internen Adreßregister bereitstellt. Bei einer Schreiboperation wird zuerst der Inhalt dieses Registers um 1 erhöht und danach das Datenwort in diesen Speicherplatz eingereiht; bei einer Leseoperation wird der durch dieses Register adressierte Platz gelesen. Das Adreßregister zeigt damit auf den zuletzt beschriebenen Platz (nächste Leseposition) und wird entsprechend auch als **Stapelzeiger (Stackpointer)** bezeichnet.

Bevor wir uns ansehen, wie ein solcher Statusspeicher realisiert werden kann, gilt es noch zu klären, welche Speicherkapazität („Tiefe" des Stapels) notwendig ist. Zunächst könnte man annehmen, daß die Stapeltiefe gleich der Anzahl der zu rettenden Register sein muß. Das ist aber nur dann ausreichend, wenn jeweils nur eine Unterbrechungsebene zugelassen wird. Bei vielen Echtzeitsystemen ist es durchaus erforderlich, daß eine mehrfache Verschachtelung der Unterbrechungen (nested interrupts) zu gewährleisten ist. Das heißt, während der Abarbeitung eines UBP kann die Unterbrechungssperre aufgehoben werden, und damit kann eine weitere Unterbrechungsforderung dieses UBP unterbrechen und ein weiterer UBP starten. Es müssen also mehrere Prozessorzustände im Statusspeicher gleichzeitig aufbewahrt werden. Die exakte Ermittlung der Stapeltiefe ist aufgrund des Zufallcharakters der Unterbrechungen nicht möglich. Es muß vielmehr der ungünstigste Fall zugrunde gelegt werden.

Die Realisierung des Statusspeichers kann nach folgenden Varianten erfolgen (Bild 3.19):

1. **Verwendung eines LIFO-Speicherschaltkreises:** Diese zunächst naheliegende Variante wird kaum eingesetzt. Von Vorteil wäre zwar die einfache Zugriffsorganisation (keine Adreßangaben erforderlich). Nachteilig ist aber, daß ein weiterer Schaltkreis eingesetzt werden muß.
2. **Interner Stapelspeicher:** Wird der LIFO-Speicher in den Mikroprozessor integriert, dann ergeben sich kurze und damit schnelle Transportwege. Aber die Speichertiefe muß zwangsläufig begrenzt werden. Dieses Prinzip wird bei einigen Mikroprozessoren angewendet, häufiger ist jedoch die folgende Variante.
3. **Externer, in den RAM-Bereich verlagerter Stapelspeicher:** Dies ist eine Kompromißlösung. Zur Vermeidung gesonderter LIFO-Schaltkreise werden die für den Datenspeicher erforderlichen RAM-Schaltkreise mitbenutzt und nur die Zugriffssteuerung (Stapelzeiger und die zum Weiterzählen notwendige Logik) wird innerhalb des Mikroprozessors angeordnet. Der Vorteil dieses Konzepts ist, daß neben der Reduzierung des Schaltkreissortiments die Speicherkapazität für diesen Statusspeicher durch entsprechende Größe des RAM-Bereichs an den Einsatzfall leicht angepaßt werden kann. Nachteilig sind allerdings die zeitaufwendigen Transportprozesse, wodurch die Programmumschlagzeiten beträchtlich erhöht werden und die Leistungsfähigkeit solcher Mikrorechner bei Echtzeitanwendungen merklich begrenzt wird.

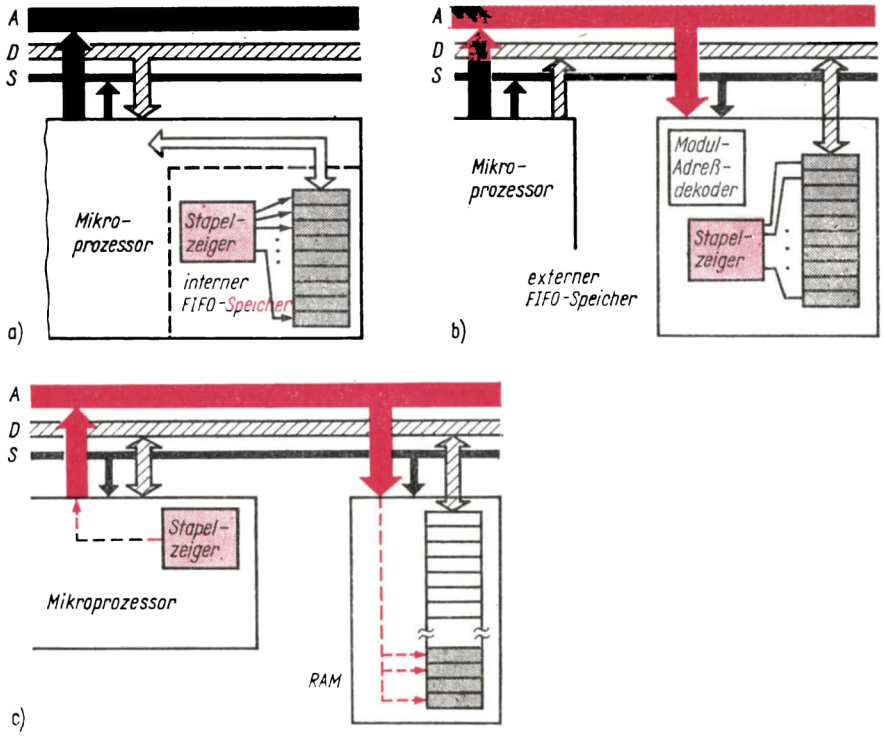


Bild 3.19. Realisierungsvarianten für Statusspeicher

a) interner Stapelspeicher; b) Anwendung eines separaten FIFO-Speichers; c) Mitbenutzung von RAM-Modulen

3.1.9. Mehrprozessorsysteme

Bisher sind wir davon ausgegangen, daß am Mikrorechnerbus nur ein aktiver Modul, der Mikroprozessor, angeschlossen ist.

Zur Erhöhung der Leistungsfähigkeit kann es durchaus zweckmäßig sein, weitere aktive Module (entweder weitere Mikroprozessoren bzw. Spezialprozessoren) an den gleichen Bus anzuschließen, damit einige oder alle passiven Module (Speicher, E/A-Ports usw.) gemeinsam genutzt werden können. Dabei gilt, daß zwar beliebig viele aktive Module anschließbar sind, daß aber immer nur einer davon arbeiten darf. Genauer formuliert: Es darf nur ein aktiver Modul den Bus mit einem Kommunikationszyklus belegen. Folglich müssen sich die Module den Bus „teilen" und zeitlich nacheinander ihre Kommunikationen ausführen (zeitmultiplex).

Dazu ist es jedoch notwendig, alle aktiven Module mit der Fähigkeit des Zuschaltens und Abschaltens ihrer Ausgangsleitungen auszurüsten und eine gewisse Rangordnung einzuführen. Diesbezüglich müssen die aktiven Module nochmals genauer in Master- und in Slave-Module unterschieden werden.

Master-Module sind die Zentralprozessoren, die einen eigenen Befehlszyklus ausführen und damit laufend Kommunikationsforderungen erzeugen. **Slave-Module** nehmen eigentlich eine Zwischenstellung zwischen passiven und aktiven

Modulen ein, indem sie erst vom Master aktiviert werden und danach „unter Aufsicht" des Masters ihre aktive Rolle am Bus für eine begrenzte Zeit einnehmen.

Bild 3.20 zeigt die prinzipielle Struktur eines **Master-Slave-Mehrprozessorsystems.** Der Mikroprozessor verfügt als Master über Steuerleitungen für die Buszugriffssteuerung (s. auch Bild 3.7). Eine Eingangsleitung, meist mit BUSRQ (busrequest, Busanforderung) oder HOLD bezeichnet, veranlaßt ihn, nach Beendigung der laufenden Kommunikation in einen Wartezustand überzugehen und sich zugleich vom Bus abzutrennen. (Dies wird durch spezielle Sendestufen erreicht, die insgesamt 3 Zustände (**Tristate**-Ausgänge) annehmen können: Die binären Zustände 0 und 1 sowie einen hochohmigen AUS-Zustand.) Die erfolgte

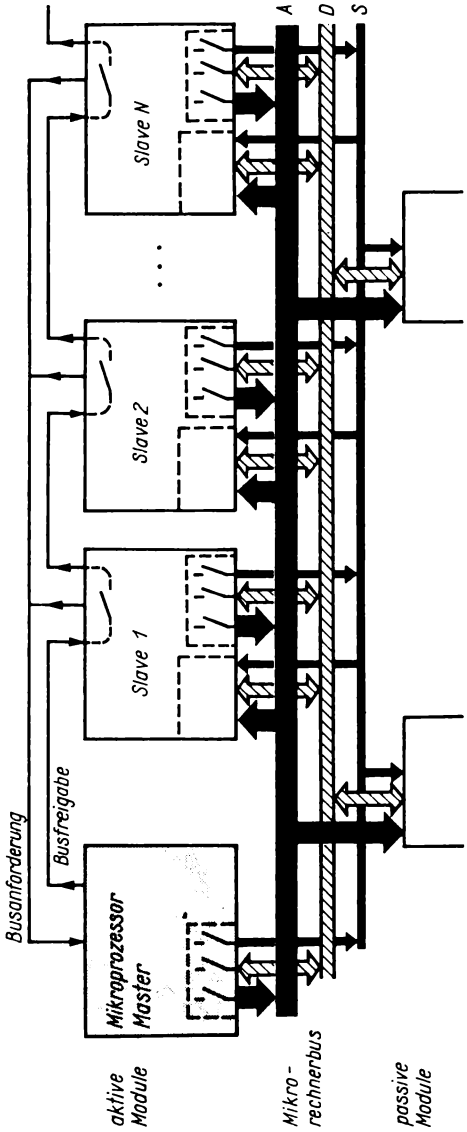


Bild 3.20. Master-Slave-Mehrprozessorsystem

Abschaltung wird durch das Quittungssignal BUSACK (busacknowledgement, Busbestätigung) angezeigt. Der Mikroprozessor bleibt nun in diesem Wartezustand, solange das Forderungssignal anliegt. Die Slave-Module besitzen dagegen eine Forderungsleitung, mit der sie bei Bedarf den Bus anfordern, und schalten erst bei einer erhaltenen Bestätigung ihre Sendestufen zu. Dabei wird durch Kettenschaltung (daisy chain) wiederum eine verdrahtete Rangfolge der Slave-Module untereinander gewährleistet. Ein solches Buszugriffsverfahren wird auch als **Zyklusstehlen** (cycle stealing) bezeichnet, da die Slave-Module bei Bedarf jederzeit den Mikroprozessor abschalten und ihm Kommunikationszyklen stehlen können. Es ist offensichtlich, daß dieses Verfahren nur dann anwendbar ist, wenn die Anzahl der gestohlenen Zyklen im Mittel relativ klein bleibt und die Arbeit des Mikroprozessors nicht merklich behindert.

Sollen dagegen mehrere Master-Module an einem Bus angeschlossen werden **(Multi-Master-Bus)**, dann besteht keine gegenseitige Kontrolle der aktiven Module und alle stellen unabhängige Forderungen nach Busherrschaft. Hier kann die Lösung nur in der Einrichtung einer übergeordneten „Instanz", dem **Busverwalter** (busarbiter) bestehen, der alle Anforderungen erhält und nach bestimmten Entscheidungsregeln jeweils für einen Master die Zuschalterlaubnis zum Bus erteilt (Bild 3.21).

Wenden wir uns nun aber der Frage zu, worin eigentlich die Leistungssteigerung des Mikrorechners begründet ist, wenn jeweils nur ein Prozessor aktiv sein kann und alle anderen pausieren müssen:

Werden gleiche Mikroprozessoren derart angeordnet, dann ist in der Tat keine bedeutende Steigerung möglich. Der einzige Vorteil besteht darin, daß durch Umschalten der Prozessoren mehrere Programme quasi nebeneinander (in Wirklichkeit zeitlich nacheinander, aber ineinander verzahnt) abgearbeitet werden können, ohne daß erst der Prozessorzustand (Programmzählerinhalt, Registerinhalte) in den Stapelspeicher gerettet werden muß.

Der Einsatz von Spezialprozessoren dagegen kann die Leistung erheblich steigern. Für welche Aufgaben lohnt es sich aber, spezielle aktive Module an den Bus anzuschließen? Die Analyse der Rechnerprogramme läßt schnell erkennen, daß zum Beispiel die Eingabe bzw. Ausgabe größerer Datenmengen zu bzw. von langsamen peripheren Geräten eine immer wiederkehrende Aufgabe ist, die mit dem wortweisen Transport, wie ihn der Mikroprozessor bietet, und dem Polling- bzw. Unterbrechungsverfahren einen beträchtlichen Programm- und Zeitaufwand

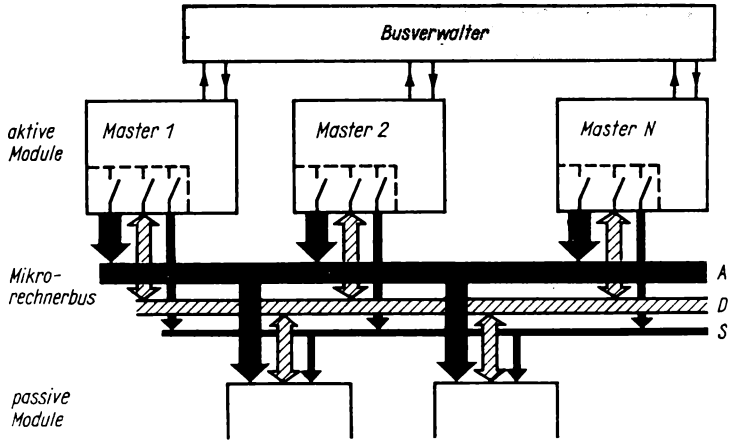


Bild 3.21. Multi-Master-Mehrprozessorsystem

ausmacht. Aus diesem Grund besteht der erste Schritt zur Leistungssteigerung bei Rechnern im Einsatz separater **Eingabe/Ausgabe-Prozessoren** (I/O-Prozessoren, DMA-Einrichtungen; bei EDV-Anlagen als Kanäle bzw. als Kanalsteuereinheiten bezeichnet).

Betrachten wir als Beispiel einen **DMA-Schaltkreis** (direct memory access, direkter Speicherzugriff). Dieser Schaltkreis bildet einen aktiven Modul mit Slave-Funktion (Bild 3.22), d. h., er kann vom Mikroprozessor im passiven Zustand mit Steuerinformation geladen und danach aktiviert werden, um selbständig eine begrenzte Menge von Aufgaben auszuführen, z. B.:

– Transport eines Datenblocks vom Speicher zu einem A-Port bzw. von einem E-Port in den Datenspeicher;
– Transport eines Datenblocks von einem Speicherbereich in einen anderen.

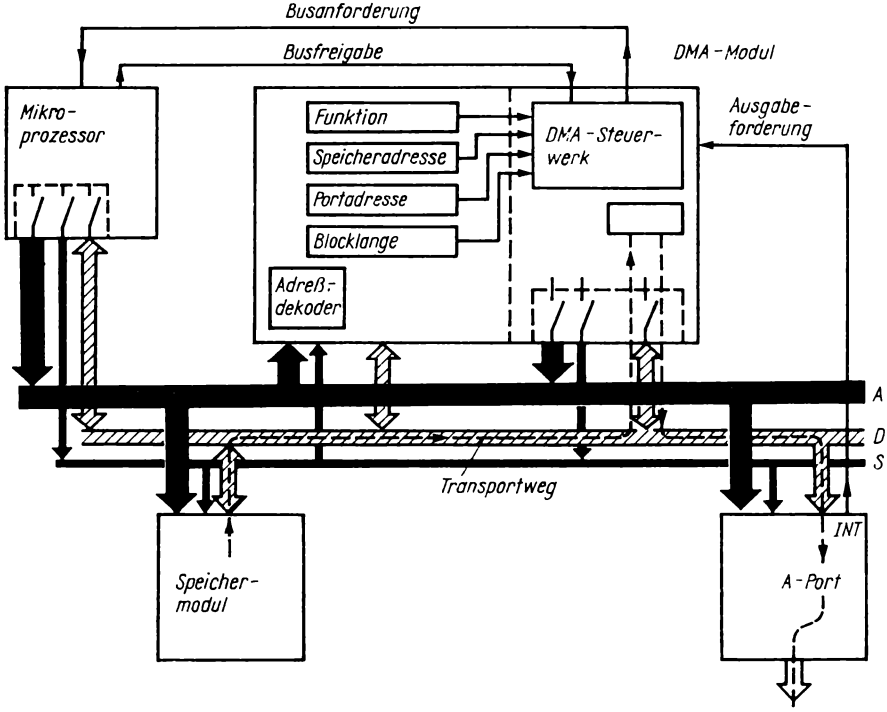


Bild 3.22. DMA-Modul

Im Fall eines Blocktransports vom Datenspeicher zu einer Ausgabeeinrichtung (z. B. Drucker) muß diesem DMA-Modul vom Mikroprozessor die Adresse des A-Ports, die Anfangsadresse des Speicherbereichs, die Länge des Blocks und die Art der auszuführenden Funktion durch mehrere Steuerworte übergeben werden. Diese Parameter werden in spezielle Steuerregister des Schaltkreises geladen. Nach Übergabe eines weiteren Steuerwortes zur Aktivierung wird der DMA-Modul aktiv, indem er zunächst die Forderung nach Busbelegung stellt und dadurch den Mikroprozessor zum Rücktritt auffordert. Nach Bestätigung der Busfreigabe durch den Mikroprozessor löst der DMA-Schaltkreis 2 eigene Kommunikationen aus, adressiert zuerst den Speicher und holt das Datenwort, um danach den A-Port zu adressieren und das Datenwort auszugeben. Nach Stehlen

dieser 2 Zyklen gibt er den Bus wieden frei und wiederholt diesen Vorgang erst, wenn die Ausgabeeinrichtung zur Übernahme eines weiteren Wortes bereit ist. Auf diese Weise wird der Mikroprozessor nicht mehr durch Abfragen oder Interruptbehandlung und durch die wortweisen Ausgaben belastet, nur einige kurze Wartezeiten durch das Zyklusstahlen des DMA-Schaltkreises entstahen die aber nur einen Bruchtnil der sonst notwendigen Bearbeitungszeit ausmachen.

3.2. Einchiprechner

Das mikroprozessororientierte Modularkonzept besticht durch sein hohes Maß an Flexibilität beim Hardware-Entwurf, besitzt jedoch den Nachteil, daß selbst bei einem sehr kleinen Leistungsumfang des Rechners eine Reihe von Schaltkreisen (Mikroprozessor, Programm- und Datenspeicher, E/A-Ports usw.) erforderlich ist und damit in der Regel eine separate Leiterkarte für den Mikrorechner gebraucht wird.

Ein Weg, den Platzbedarf solcher modularen Systeme weiter zu reduzieren, besteht darin, kombinierte Schaltkreise zu produzieren, indem beispielsweise Datenspeicher und E/A-Ports bzw. Festwertspeicher und Zeitgeber auf einem Schaltkreis integriert werden. Damit bleibt das Modularkonzept jedoch gewahrt, aber die Minimalkonfiguration ist bereits mit 2 oder 3 Schaltkreisen realisierbar.

Die Integration einer Grundkonfiguration eines Rechners auf einen Schaltkreis bedeutet dagegen, daß der extrem niedrige Platzbedarf mit einem merklichen Verlust an Flexibilität bezahlt werden muß.

Die Einchiprechner zielen daher in erster Linie in das Feld der Massenanwendungen, bei denen Rechner als Steuerelemente und „Intelligenzlieferanten" in Geräte integriert werden sollen. Die typische Anschlußbelegung und innere Struktur eines Einchiprechners zeigt Bild 3.23. Funktionell bestehen damit keine Unterschiede zu den im vorhergehenden Abschnitt behandelten mikroprozessororientierten Systemen.

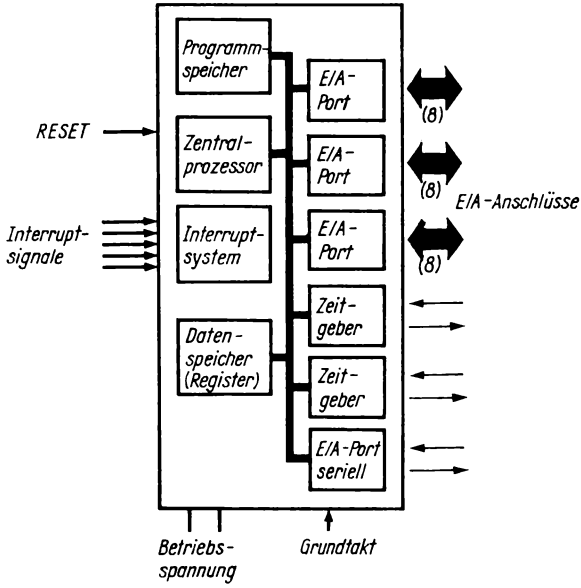


Bild 3.23. Einchip-mikrorechner

Kenngrößen

Aufgrund der festgefügten inneren Struktur wird ein Einchiprechner durch folgende Kenngrößen charakterisiert:

Programmspeicherkapazität. Der Programmspeicher wird durch einen maskenprogrammierten ROM gebildet, d. h., das gesamte Rechnerprogramm muß bereits vor Herstellung des Schaltkreises bekannt sein. Daraus folgt die Notwendigkeit des Einsatzes in großen Stückzahlen. Für Anwendungen in kleinen Stückzahlen und zur Programmentwicklung existieren auch teilweise Einchiprechner mit EPROM-Programmspeicher. Die Speichergröße liegt gegenwärtig im Bereich zwischen 1 und 4 KByte und wird sich sicher zukünftig noch erhöhen.

Datenspeicherkapazität. Der Umfang des Datenspeichers (Registersatz) ist meist sehr gering und beträgt oft nur 128 byte.

Anzahl und **Typ** der **E/A-Ports.** Entscheidend für die Einsatzbreite sind die Anzahl und Art der E/A-Schnittstellen. Da keine einfache Erweiterungsmöglichkeit besteht, wird bei den Einchiprechnern versucht, durch Programmierbarkeit der vorhandenen E/A-Ports ein gewisses Maß an Flexibilität zu erreichen. So ist es in der Regel möglich, die Ports in verschiedenen Arbeitsweisen zu betreiben, indem spezielle interne Steuerregister vom Programm mit Steuerworten geladen werden. Auf diese Weise kann der Einchiprechner an unterschiedliche Einsatzfälle angepaßt werden. Zur Standardkonfiguration gehören parallele E/A-Ports mit und ohne Handshake, meist eine serielle E/A-Schnittstelle und ein oder mehrere Zeitgeber. Darüber hinaus werden zur direkten Ankopplung an die Umwelt auch Einchiprechner mit analogen Schnittstellen (AD- und DA-Umsetzer) hergestellt.

Befehlssatz und **Operationsgeschwindigkeit.** Diese durch den internen Zentralprozessor bestimmten Parameter unterscheiden sich kaum von denen der Mikroprozessorschaltkreise. Meist liegt die Operationsgeschwindigkeit noch geringfügig höher, bedingt durch die ausschließlich innerhalb eines Schaltkreises verlaufenden Transporte.

Daraus folgt, daß auf keinen Fall diese Einchiprechner als leistungsmäßig schwächer einzuordnen sind, ihre Beschränkungen ergeben sich allein aus der begrenzten Speichergröße und der begrenzten Anzahl der E/A-Anschlüsse. Die typischen Operationsgeschwindigkeiten liegen gegenwärtig bei 500 000 bis 1 Million Operationen/Sekunde bei einer Wortbreite von 8 bit.

Steuersignale. Die Steuermöglichkeiten beschränken sich meist auf ein Signal zur Initialisierung (RESET), wobei eine automatische Initialisierung beim Zuschalten der Betriebsspannung in der Regel bereits auf dem Schaltkreis realisiert ist, und auf einige Interrupteingänge. Im Gegensatz zum Mikroprozessorkonzept existiert jedoch im Normalfall kein Sammelinterrupt, da keine Möglichkeit zur Eingabe des Interruptvektors vorhanden ist.

Einsatzvarianten

Obwohl diese Einchiprechner vorwiegend für solche Einsatzfälle vorgesehen sind, die mit diesem einen Schaltkreis auskommen, werden doch vom Schaltkreishersteller oft auch darüber hinausgehende Möglichkeiten vorgesehen. Wir wollen deshalb die folgenden 3 Einsatzvarianten für Einchiprechner unterscheiden:

1. Einchiprechner in der **Großserienfertigung:** Bei extrem hohen jährlichen Fertigungsstückzahlen (Fahrzeug- und Konsumgüterindustrie) wird die Herstellung spezieller (endproduktgebundener) Einchiprechner wirtschaftlich. Diese Schaltkreise brauchen folglich über keinerlei Anpaßfunktionen zu verfügen.
2. Einchiprechner als **universelles Zwischenprodukt:** Bei dieser Einsatzzielstellung muß seitens des Schaltkreisherstellers entweder durch Programmierbarkeit der E/A-Ports oder durch Bereitstellung einer Anzahl verschiedener

Typen und jeweils spezifischer Umwelt-Schnittstellen eine begrenzte Anpassungsfähigkeit an typische Einsatzfelder gesichert werden.

3. Einchiprechner als **Grundbaustein für Mikrorechnersysteme**: Bei vielen Einchiprechnern wurde beim Systementwurf eine über das Ein-Schaltkreis-Konzept hinausgehende Struktur vorgesehen, indem diese Schaltkreise als Grundbaustein dienen und wiederum modular erweiterbar sind.

   Dazu ist es erforderlich, die Schaltkreise mit einer Mikrorechnerbus-Schnittstelle auszurüsten (Verlängerung des internen Bus), an die weitere Speicher, E/A- und Spezialmodule angeschlossen werden können. Da die dafür notwendige Anzahl von Anschlüssen nicht zusätzlich vorhanden ist, kann die Lösung nur in einer Umschaltung der E/A-Port-Funktion des Einchiprechners bestehen. In der Regel werden 2 E/A-Ports durch ein spezielles internes Steuerwort in einen Adreß- bzw. Datenbus verwandelt (Bild 3.24). Dann lassen sich mit einem Einchiprechner und wenigen Ergänzungsschaltkreisen wiederum leistungsfähige, angepaßte Rechnerkonfigurationen aufbauen.

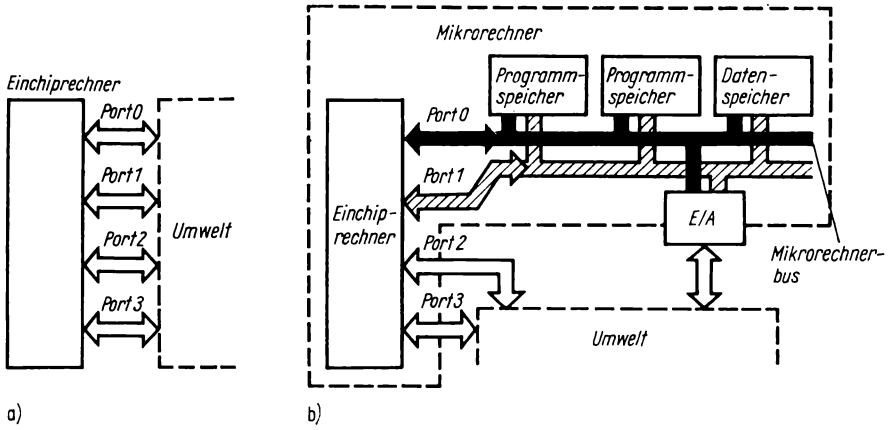


Bild 3.24. Einsatzvarianten für Einchiprechner

a) Ein-Baustein-Rechner; b) Einchiprechner als Grundbaustein eines modular erweiterbaren Mikrorechners

4. Mikrorechner-Software

Nach dem Entwurf der für den vorgegebenen Einsatzfall notwendigen Hardwarestruktur des Mikrorechners schließt sich die Softwareentwicklung an.

Durch ein aus dem Programmspeicher geholtes Befehlswort wird die Funktion des Rechners für die Dauer eines Befehlszyklus bestimmt (also bei Mikrorechnern für eine Zeitdauer von einer oder wenigen Mikrosekunden). Durch Einspeichern einer geeigneten Befehlsfolge in diesen Programmspeicher wird dementsprechend die Arbeitsweise des Rechners insgesamt festgelegt.

Die für eine bestimmte Anwendung des Rechners erforderliche Befehlsfolge wird als dessen **Programm** bezeichnet. Bei umfangreichen Programmen werden die Ausdrücke **Programmsystem, Gesamtprogramm** verwendet, um noch Gliederungsmöglichkeiten in Programmodule (Teilprogramme) zu haben.

Unter **Programmierung** wird dementsprechend der Vorgang der Programmentwicklung verstanden, der vor der ersten Inbetriebnahme des Rechners ausgeführt werden muß.

Wir müssen uns an dieser Stelle noch einmal verdeutlichen, daß die Begriffe Programm und Programmierung selbst im technischen Sprachgebrauch mit unterschiedlicher Bedeutung verwendet werden. So wird z. B. von der Programmwahl bei einer Haushaltwaschmaschine, von der Programmierung einer numerisch gesteuerten Werkzeugmaschine oder von einem freiprogrammierbaren Industrieroboter gesprochen. Hiermit ist die Auswahl einer bestimmten Betriebsweise eines Gerätes von vielen (auch unendlich vielen) Möglichkeiten gemeint, d. h., der Programmiervorgang ist im Sinne der im Abschnitt 2.1. eingeführten Grundstruktur eines Echtzeitsystems in Wirklichkeit die Eingabe von Bedienkommandos. Eine solche **Geräteprogrammierung** darf nicht mit der Programmierung der in diesen Geräten integrierten Rechner verwechselt werden.

Um den Unterschied zu verdeutlichen, soll eine Darstellung herangezogen werden, die allgemein als „Schalenmodell" bezeichnet wird (Bild 4.1). Der Kern eines rechnergesteuerten Systems (Gerätes) wird durch die Hardware eines Rechners gebildet, der aufgrund seiner Struktur und Arbeitsweise grundsätzlich für eine Vielzahl von Anwendungen geeignet ist. Zur Realisierung eines bestimmten Echtzeitsystems wird dieser Rechner mit einer speziellen Peripherie – den Meß- und Stelleinrichtungen sowie den Bedien- und Anzeigeelementen – und mit einem Programmsystem „umhüllt". Diese beiden Komponenten bilden gleichsam eine „Schale" um den Hardware-Kern. Das Modell soll verdeutlichen, daß damit aus dem universell verwendbaren Rechner ein spezialisiertes Gerät geworden ist und der eigentliche Rechnerkern dem Zugriff des Gerätebedieners entzogen, d. h. Hardware und Software des Rechners nicht mehr manipulierbar sind. Nur über die Bedienperipherie besteht noch eine Einwirkungsmöglichkeit. Diese Bedienkommandos sind aber vom Programmierer des Rechners im Programmablauf bereits vorgeplant worden. Sie werden zu bestimmten Zeitpunkten erwartet und nach Eingabe durch ein darauffolgendes Teilprogramm analysiert (**interpretiert**). Als Folge werden spezifische Aktionen ausgeführt, die selbstverständlich sämtlich ebenfalls als Teilprogramme im Programmspeicher des Rechners vorhanden sind. Es gilt also, daß die Geräteprogrammierbarkeit durch die Rechnerprogrammierung festgelegt wird.

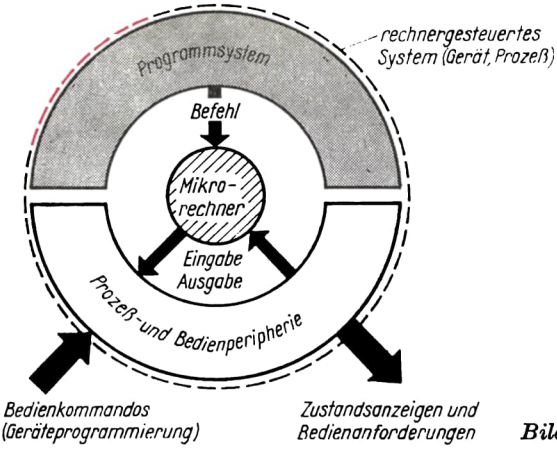


Bild 4.1. Schalenmodell

Wenn im folgenden die Begriffe Programm und Programmierung verwendet werden, dann ist damit immer die Ebene der Rechnerprogrammierung gemeint.

Oben haben wir vorausgesetzt, daß beim Systementwurf die Trennung in Hardware- und Softwareentwicklung möglich ist und die Software nach Festlegung der Hardware erstellt wird. Bei komplexen Systemen kann dieser Prozeß durchaus mehrfach durchlaufen werden. Prinzipiell gilt, daß die Software von der Hardware abhängig ist. Oftmals wird jedoch während oder nach der Softwareentwicklung erst ersichtlich, daß sich die anwendungsseitigen Forderungen nicht realisieren lassen (z. B. zu lange Programmlaufzeiten und dadurch Verletzung von Echtzeitbedingungen). Folglich werden Hardwareänderungen und nachfolgend wiederum Softwareänderungen erforderlich.

Dieses Wechselspiel zwischen Hardware und Software ist nun zwar keine neue, aber durch die Mikrorechentechnik wiederbelebte Erkenntnis. Die Entwicklung der Rechentechnik hatte zuvor zu einer vorwiegend starren Hardware geführt, indem Rechner als komplette Endprodukte eines Herstellers produziert wurden. Damit war die Anpassung an konkrete Einsatzfälle beim Anwender in erster Linie auf die Softwareseite beschränkt, was auch zu einer personellen Trennung zwischen Hardware- und Softwareentwicklung geführt hat. Die Grenze der Anwendbarkeit einer Rechnersteuerung wurde damit ebenfalls wesentlich durch die Fähigkeiten des Programmierers bestimmt. Die Modularkonzepte der Mikrorechner-Hardware bieten dagegen einen hohen Grad an Flexibilität und erzwingen die Einheit von Hardware- und Softwareentwurf. Optimale Systemlösungen lassen sich nur dann finden, wenn Kenntnisse sowohl auf der Hardware- als auch der Softwareseite vorhanden sind.

Als Folge der sinkenden Hardwarekosten nimmt dabei die Softwareentwicklung eine immer bedeutendere Position in der Kostenbilanz ein. So wird in einschlägigen Veröffentlichungen oft angeführt, daß zukünftig die Softwarekosten überhaupt die Ökonomie eines Rechnereinsatzes bestimmen und daraus die Forderung nach verstärkten Anstrengungen zur Weiterentwicklung der Software-Technologie abgeleitet. Diese Folgerung ist in jedem Fall richtig. Die erste Aussage ist jedoch in der Mikrorechentechnik differenzierter zu sehen.

Es lassen sich 3 Einsatzfelder unterscheiden:

1. Bei Produkten mit hoher Fertigungsstückzahl werden die Kosten nach wie vor wesentlich durch den Hardwareaufwand bestimmt, da sich die einmaligen Softwarekosten auf die Gesamtstückzahl verteilen;
2. Bei Kleinserien und Einzelfertigungen (z. B. Geräte zur innerbetrieblichen

Rationalisierung) gilt das Primat der Software. Hier kann es durchaus wirtschaftlich sein, durch Überdimensionierung der Hardware für möglichst geringen Softwareaufwand bzw. schnelle Programmentwicklungszeiten zu sorgen;
3. Eine 3. Gruppe bilden solche Anwendungen mit komplexer Struktur (große Systeme). Hier muß an erster Stelle die Übersichtlichkeit während und nach der Entwicklungsphase stehen, d. h., solche Probleme wie arbeitsteilige Entwicklung, einfache Testbarkeit und Korrigierbarkeit sowie leichte Wartungs- und Servicedienste stehen im Vordergrund. Es ist einzuschätzen, daß dabei Hardware- und Softwareaufwand gleichermaßen eingehen.

4.1. Maschinenprogramm

Das im Programmspeicher befindliche Programm kann nur aus Kodewörtern (Befehlen) bestehen, die der Zentralprozessor (Mikroprozessor) versteht. Das heißt, der Befehlssatz dieses Prozessors bestimmt die Art der verwendbaren Befehle und ihre genaue Kodierung. Diese Kodewörter werden als Maschinenbefehle bezeichnet. Wie auch immer die Erstellung des Programms durch den Programmierer erfolgt, das Endprodukt der Softwareentwicklung muß in jedem Fall eine Folge von solchen Maschinenbefehlen sein. Dementsprechend wird dieses Programm auch als Maschinenprogramm bezeichnet, da es von dem jeweils eingesetzten Maschinen-(Prozessor-)Typ abhängt.

4.1.1. Programmbeispiele

Im folgenden wollen wir einige Beispiele für Maschinenprogramme betrachten. Es soll dabei nicht Ziel dieses Abschnittes sein, die Programmerstellung auf Maschinenniveau im Detail kennenzulernen oder zu üben. Die Programmierung in der **Maschinensprache** ist zeitaufwendig und verlangt hohe Konzentration. Da der Rechner aber auf diesem Niveau arbeitet, läßt sich nur so dessen Arbeitsweise kennenlernen. Mit den angeführten Beispielen soll gezeigt werden, in welche Elementarschritte letztlich ein Problem zerlegt werden muß, damit es ein Rechner lösen kann, bzw. wie viele Befehle ein Prozessor nacheinander ausführen muß, um selbst einfachste Aufgaben zu vollziehen. Zum anderen sollen die Beispiele einige für Steuerrechner typische Aufgaben und ihre Programmierung veranschaulichen.

Da die Programmierung in Maschinensprache vom jeweils verwendeten Prozessor abhängt, wird für die nachfolgenden Beispiele folgendes vorausgesetzt:

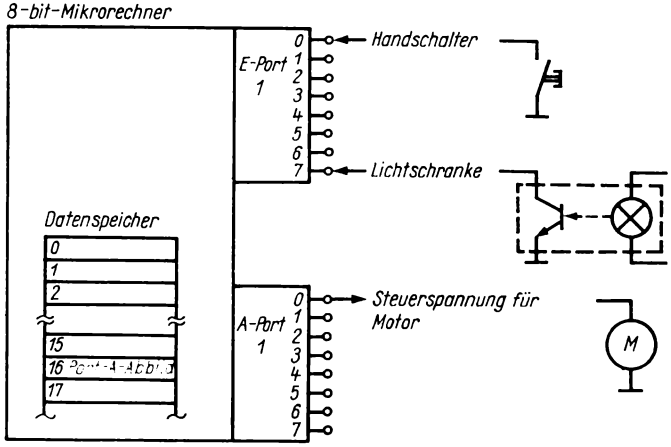
a) Der verwendete Mikrorechner hat eine Verarbeitungsbreite von 8 bit.
b) Der Befehlssatz entspricht dem im Abschnitt 2.2. eingeführten Musterbefehlssatz (Tafel 2.1). Demnach liegt ein Einadreßprozessor vor, d. h., Ergebnis und 1. Operand stehen immer in einem bestimmten Register, dem Akkumulator.
c) Die Befehlsworte sollen eine konstante Wortbreite von 16 bit aufweisen und damit immer 2 aufeinanderfolgende Speicherplätze im Programmspeicher belegen. (Diese Voraussetzung ist eine Vereinfachung. Im realen Fall werden die Befehlsworte unterschiedliche Länge aufweisen.)

Außerdem ist zu beachten, daß die in den folgenden Beispielen entwickelten Programme immer nur Teile aus einem Gesamtprogramm bilden. Folglich ist auch zum Zeitpunkt der Programmerstellung noch nicht bekannt, auf welchem Platz des Programmspeichers deren Abspeicherung beginnt. Erst wenn alle Teilprogramme vorliegen, kann das Gesamtprogramm zusammengestellt werden, und damit ergeben sich die konkreten Werte für die Programmspeicheradressen. Für quantitative Betrachtungen sind für die einzelnen Befehle Ausführungszeiten

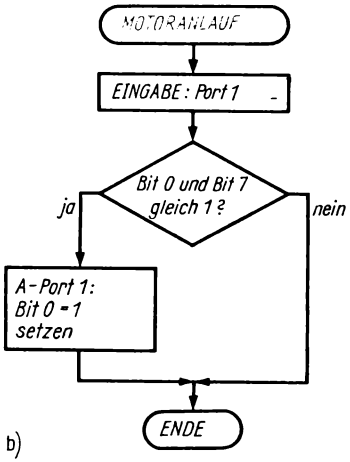
angenommen worden, die etwa den realen Werten der vorwiegend eingesetzten Mikrorechner bzw. Mikroprozessoren in MOS-Technologie entsprechen.

Beispiel 1

Es soll die programmäßige Lösung jenes Beispiels betrachtet werden, das bereits im Abschnitt 2.1. zur Gegenüberstellung der prinzipiellen Realisierungsformen elektronischer Steuerungen herangezogen wurde. Der Antrieb einer Bearbeitungsmaschine sollte nur unter der Bedingung manuell einschaltbar sein, daß eine Schutzvorrichtung, deren Lage durch eine Lichtschranke signalisiert wird, zur Absicherung des Arbeitsraumes eingerichtet worden ist (Bild 2.9). Bild 4.2a zeigt die zugrunde gelegte Hardwarestruktur. Handschalter und der elektronische Schalter der Lichtschranke sind an Leitung 0 und 7 des Eingabeports geschaltet. Die Steuerspannung für den Anlauf des Antriebsmotors wird auf Leitung 0 des Ausgabeports bereitgestellt. Die restlichen Leitungen des Eingabe- bzw. Ausgabeports werden zum Teil noch für andere Eingangs- bzw. Ausgangssignale benötigt, die aber für den Anlaufvorgang keine Rolle spielen. Auf Platz 16 des Daten-



a)



b)

Bild 4. 2. Motoranlauf (Beispiel 1)
a) Hardwarestruktur; b) Programmablaufplan

speichers soll ständig der Zustand des Ausgabeports gespeichert sein. Diese Maßnahme ist deshalb notwendig, da dieser Port nicht vom Prozessor gelesen werden kann und folglich immer ein „Abbild (Kopie)" seines aktuellen Zustands im Datenspeicher aufbewahrt werden muß, das dann jederzeit lesbar ist. Bild 4.2b zeigt als Ablaufplan die von diesem Teilprogramm „Motoranlauf" auszuführenden Schritte. Mit den in Tafel 2.1 eingeführten Befehlen ergibt sich das in Tafel 4.1 aufgeführte Maschinenprogramm:

Wenn diese Befehlsfolge erstellt worden ist, besteht nun der letzte Arbeitsschritt bei der Programmentwicklung noch darin, anhand einer Kodetabelle die den verwendeten Befehlen entsprechenden Kodewörter zu ermitteln. Diese Kodewörter sind dann in den Programmspeicher einzugeben.

Als Ergebnis können wir feststellen, daß das Programm 9 Befehle umfaßt (natürlich handelt es sich dabei nur um ein Teilprogramm der Steuerungsaufgabe). Mit den zugrunde gelegten Befehlsausführungszeiten ergibt sich, daß bei nichterfüllter Anlaufbedingung (Handschalter oder Schutzvorrichtung nicht geschlossen) das Programm 8 µs und anderenfalls 17 µs benötigt.

Dieses Beispiel zeigt bereits, daß die Programmierung in Maschinensprache ein hohes Maß an Konzentration vom Programmierer verlangt. So müssen beispielsweise die Adressen des Programmspeichers laufend fortgeschrieben werden, damit die jeweiligen Sprungadressen ablesbar sind. Dabei ist es sogar notwendig, Sprungadressen zunächst offen zu lassen, da sie sich erst im weiteren Ablauf der Programmerstellung ergeben und entsprechend erst dann in die zurückliegenden Befehle eingetragen werden können.

Tafel 4.1. Maschinenprogramm zu Beispiel 1 (Motoranlauf)

Adresse	Befehl	Erläuterungen der Wirkung
X + 0	**EINGABE** | **Port 1** (2 µs)	Der aktuelle Zustand aller Signale am E-Port 1 wird in den Akkumulator transportiert. (Als Programmspeicheradresse wird zunächst die Adresse X angenommen.)
X + 2	**LADE** | **Reg 1** | m. Binärwort **1000 0001** (2 µs)	Register 1 wird mit dem Binärwort 10000001 geladen.
X + 4	**UND** | **Reg 1** (1 µs)	Der Akkumulatorinhalt (Zustand des E-Ports) wird mit dem Binärwort von Register 1 nach der logischen UND-Funktion verarbeitet. Der Inhalt von Register 1 dient dabei als Maske, d. h., es enthält an den Bitstellen eine 1, die für die weitere Verarbeitung von Interesse sind. In unserem Fall sind das die Positionen 0 und 7, da dort die beiden Leitungen angeschlossen sind. Das Ergebnis steht nach Ausführung dieses Befehls im Akkumulator und ist nur dann von 0 verschieden, wenn beide Eingangsleitungen zum Zeitpunkt des Eingabebefehls den Zustand 1 besaßen (also sowohl Handschalter betätigt als auch Schutzvorrichtung eingerückt).

Fortsetzung von Tafel 4.1

Adresse	Befehl	Erläuterungen der Wirkung
X + 6	SPRUNG | wenn NULL | Y (3 µs)	Ist das Ergebnis gleich 0 (entweder Handschalter nicht betätigt oder Schutzvorrichtung nicht eingerückt), werden die folgenden Befehle übersprungen und das Programm ab Adresse Y fortgesetzt. (Diese Adresse ist beim Schreiben dieser Zeile noch nicht bekannt, muß also zunächst offen bleiben und kann erst nachträglich eingetragen werden.)
X + 8	TRANSPORT | nach Akku | von Sp 16 (2 µs)	Die folgenden Befehle bewirken den Anlauf des Motors. Zunächst wird aus dem Datenspeicher vom Platz 16 der zuletzt gestellte Zustand des A-Ports in den Akkumulator geholt.
X + 10	LADE | Reg 1 | m. Binärwort 0000 0001 (2 µs)	Register 1 wird mit dem Binärwort 00000001 geladen.
X + 12	ODER | Reg 1 (1 µs)	Durch die ODER-Verknüpfung des Akkumulators mit dem Wert im Register 1 werden alle Bitstellen außer der 1. Stelle unverändert gelassen. Nur die 1. Bitstelle wird 1 gesetzt.
X + 14	AUSGABE | Port 1 (2 µs)	Dieser Befehl transportiert den Akku-Inhalt zum A-Port 1. Damit wird dort die Steuerleitung für den Motor aktiviert.
X + 16	TRANSPORT | nach Sp 16 | von Akku (2 µs)	Durch diesen Befehl wird der neue Zustand des A-Ports wieder auf den Platz 16 des Datenspeichers transportiert. Damit ist das Programm beendet.
Y = X + 18	*Folgebefehl*	Auf diesem Programmspeicherplatz steht der 1. Befehl eines weiteren Teilprogramms, das eine nächste Aufgabe im Rahmen des Gesamtprogramms zu lösen hat. Auf diesen Befehl muß auch der oben programmierte bedingte Sprung gelangen, d. h., erst jetzt kann dort die Sprungadresse X + 18 eingetragen werden.

Werden nach einer ersten Fertigstellung eines Programms noch Fehler festgestellt und sind demzufolge Befehle zu ändern, zu ergänzen oder zu streichen, so kann sich die Adreßfolge im Programmspeicher ändern, und alle Sprungadressen sind zu korrigieren.

Zur Erleichterung der Programmierarbeit ist es daher günstig, zunächst nur **symbolische Adressen** einzuführen. Anstelle der konkreten Zahlenwerte werden nur Symbole (Buchstabenfolgen oder Kombinationen aus Buchstaben und Ziffern)

für diejenigen Adressen des Programmspeichers eingeführt, die Einsprungstellen bilden. Genauso kann mit den im Programm erforderlichen Konstanten verfahren werden. Das Programm von Tafel 4.1 kann damit im ersten Entwurfsschritt auch entsprechend Tafel 4.2 geschrieben werden.

Diese Methode vereinfacht die Programmierarbeit. Wir werden sie deshalb bei den folgenden Beispielen von vornherein anwenden. Damit sind jedoch vor der Einspeicherung des Programms in den Programmspeicher nun 2 Schritte erforderlich. Neben dem Aufsuchen der Kodewörter für die Befehle sind auch noch alle symbolischen Adressen und Konstanten durch die realen Werte zu ersetzen.

Beispiel 2

Die Aufgabenstellung von Beispiel 1 soll dahingehend erweitert werden, daß 200 ms nach Anlauf des 1. Antriebs ein 2. Motor anlaufen soll.

Ohne Verwendung eines Zeitgebermoduls läßt sich diese Aufgabe dadurch lösen, daß nach Ausgabe des Anlaufbefehls für den 1. Antrieb ein Programmstück folgt, das genau 200 ms Laufzeit benötigt, ohne dabei äußere Wirkungen zu erzeugen. Danach erfolgt der Stellbefehl für den 2. Antrieb.

Die Aufgabe besteht also darin, eine möglichst kurze Befehlsfolge zu finden (Platzbedarf im Programmspeicher!), die aber den Rechner über längere Zeiträume „beschäftigt". (Wir erinnern uns, daß ein Mikrorechner in 200 ms 100 000 bis 200 000 Befehle abarbeitet!) Hierfür sind Programmschleifen geeignet (Bild 4.3). Das Programm für eine einfache Schleife ist in Tafel 4.3 beschrieben.

Die Laufzeit dieses Programms beträgt

$$T = (2 + 4 \cdot ZEITKONST)\ \mu s.$$

Die maximal programmierbare Verzögerungszeit hängt nun vom Zahlenbereich ab, der für ZEITKONST verwendet werden kann (bei 8-bit-Rechnern enthält

Tafel 4.2. Programm mit symbolischen Adressen (Beispiel 1)

Adresse	Befehl	Erläuterung
	EINGABE **Port 1**	Anstelle der laufenden Fortschreibung der Programmspeicheradressen wird nur noch die eine Einsprungstelle mit ADR 1 markiert. Außerdem werden die Konstanten allgemein als MASKE und MOTORANLAUF bezeichnet und ihre konkreten Werte in dieser 1. Programmierphase noch nicht festgelegt.
	LADE **Reg 1** mit MASKE	
	UND **Reg 1**	
	SPRUNG wenn **NULL** **ADR 1**	
	TRANSPORT nach **Akku** von **Sp 16**	
	LADE **Reg 1** MOTORANLAUF	
	ODER **Reg 1**	
	AUSGABE **Port 1**	
	TRANSPORT nach **Sp 16** von **Akku**	
ADR 1	*Folgebefehl*	

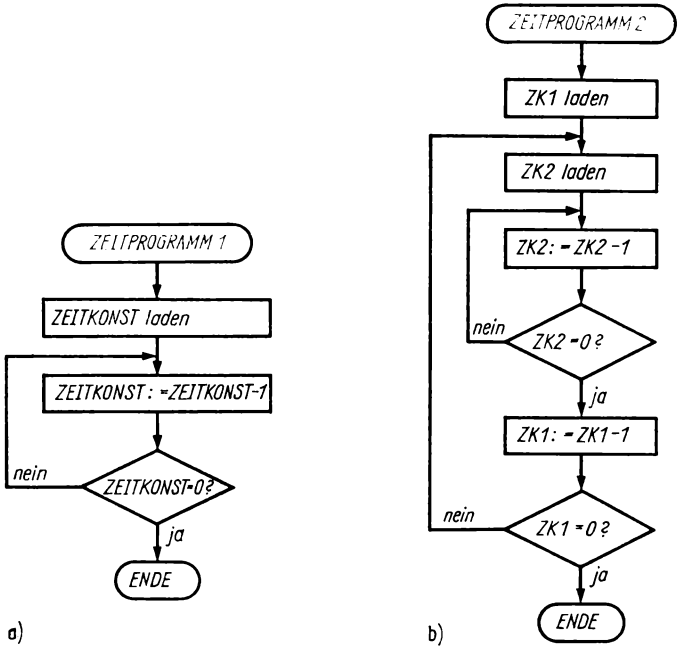


Bild 4.3. Zeitschleifen (Beispiel 2)
a) einfache Zeitschleife; b) Doppelschleife

Tafel 4.3. Programm für eine einfache Zeitschleife

Adresse	Befehl	Erläuterung
	LADE | **Reg 1** | *ZEITKONST* (2 µs)	Register 1 wird mit Binärwort *ZEITKONST* geladen.
ADR 2	**DEC** | **Reg 1** (1 µs)	Der Zahlenwert *ZEITKONST* wird um 1 verringert.
	SPRUNG | wenn **NICHT NULL** | *ADR 2* (3 µs)	Solange der Inhalt von Register 1 ungleich 0 ist, wird durch den bedingten Sprung zu ADR 2 zurückgesprungen. Damit werden die beiden Befehle so oft wiederholt, bis der Inhalt von Register 1 0 geworden ist. Danach wird mit dem Folgebefehl fortgesetzt.
	Folgebefehl	

dieser Zahlenbereich die ganzen Zahlen von 0 bis 255, bei 16-bit-Rechnern die ganzen Zahlen von 0 bis 65535). Mit den in Tafel 4.3 angegebenen Befehlsausführungszeiten ergeben sich damit die folgenden Zeitbereiche:

8-bit-Mikrorechner 6 µs···1026 µs

16-bit-Mikrorechner 6 µs···262146 ms

(Das Einstellintervall beträgt dabei 4 µs.)

Die erforderliche Verzögerungszeit von 200 ms wäre also mit dieser einfachen Programmschleife bei einem 8-bit-Rechner nicht realisierbar. Mit der im Bild 4.3b gezeigten Doppelschleife läßt sich die Laufzeit wesentlich erhöhen. Das Programm besteht dann aus den 6 Befehlen entsprechend Tafel 4.4.

Die Laufzeit dieses Programms berechnet sich zu:

$$T = [2 + ZK1 \cdot (6 + 4 \cdot ZK2)]\ \mu s.$$

Die maximal programmierbare Verzögerungszeit beträgt damit:

beim 8-bit-Rechner 263,682 ms,

beim 16-bit-Rechner 17180,262402 s (etwa 4 h 46 min).

Wird im obigen Programm $ZK1 = 202$ und $ZK2 = 246$ gesetzt, so ergibt sich die erforderliche Verzögerungszeit von 200 ms.

Tafel 4.4. Programm für eine Doppel-Zeitschleife

Adresse	Befehl	Erläuterung
	LADE | **Reg 1** | mit *ZK 1* (2 µs)	Register 1 und 2 werden mit Dualzahlen *ZK1* und *ZK2* geladen.
ADR 3	**LADE** | **Reg 2** | mit *ZK 2* (2 µs)	
ADR 4	**DEC** | **Reg 2** (1 µs)	Subtraktion von 1 bei Register 2 und Wiederholung, solange der Inhalt von Register 2 ungleich 0 ist.
	SPRUNG | wenn **NICHT NULL** | *ADR 4* (3 µs)	
	DEC | **Reg 1** (1 µs)	Danach Subtraktion 1 bei Register 1.
	SPRUNG | wenn **NICHT NULL** | *ADR 3* (3 µs)	Rücksprung zu ADR 3 und damit erneutes Laden von Register 2, solange Register 1 ungleich 0. Damit wird dieser Vorgang *ZK1*-mal wiederholt. Danach wird im Programmlauf fortgesetzt.
	Folgebefehl	

Beispiel 3

Bei Steuerungen mit Bedienerführung ist nahezu immer die Übermittlung von Nachrichten vom Rechner an den Bediener erforderlich, um diesen auf bestimmte Situationen aufmerksam zu machen, die Eingabe von Bedienkommandos abzufordern usw. Als Ausgabegeräte dienen dazu entweder Displayzeilen, Bildschirmgeräte oder Drucker.

Es soll nun ein Teilprogramm entwickelt werden, daß die Ausgabe eines Textes an ein Ausgabegerät bewirkt. Dabei soll die im Bild 4.4a angegebene Hardwarestruktur vorliegen (zunächst ohne BEREIT-Signal). Für den auszugebenden Text ist im Datenspeicher ab Adresse 3500 ein Feld von 250 Speicherplätzen reserviert. Jedes Textzeichen belegt einen Speicherplatz. Auf dem 1. Speicherplatz (Adresse 3500) soll dabei die Länge des auszugebenden Textes stehen. (Die Erzeugung und der Transport des Textes nach diesem Speicherbereich ist die Aufgabe eines anderen, hier nicht zu betrachtenden Teilprogramms.)

Die Aufgabe der Textausgabe könnte zunächst mit dem Programm nach Tafel 4.5 entsprechend dem Algorithmus im Bild 4.4b gelöst werden.

Mit den angegebenen Befehlsausführungszeiten wird durch dieses Programm dem Ausgabegerät alle 9 µs ein Zeichen übergeben. Das entspricht einer Schreibgeschwindigkeit von etwa 600 µs je Zeile (60···80 Zeichen) bzw. etwa 10 ms je Seite (1000 Zeichen). Oder anders betrachtet, die Ausgaberate des Steuerrechners beträgt etwa 110000 Zeichen/s (50···100 Seiten/s).

Ein direkt angeschlossenes Bildschirmgerät wäre in der Lage, die Zeichen mit dieser Geschwindigkeit zu übernehmen. Voraussetzung ist dann allerdings, wie in diesem Beispiel angenommen, daß der Text auf eine Seite begrenzt ist. Von mechanischen Druckern kann dagegen diese Zeichenrate nicht verarbeitet werden (wenn nicht im Drucker ein weiterer elektronischer Textspeicher realisiert wird).

Liegen solche ungleichen Paarungen (Elektronik–Mechanik) vor, dann ist eine Synchronisation zwischen Rechner und Ausgabegerät erforderlich, um dem Rechner die langsame Arbeitsweise aufzuzwingen. Dafür stehen, wie bereits im Abschnitt 2.2.5. gezeigt, als Realisierungsvarianten das Abfrageprinzip (polling) und das Anforderungsprinzip (Unterbrechungsforderung) zur Auswahl.

Betrachten wir zunächst, welche Programmänderungen bei Anwendung des **Abfrageprinzips** erforderlich sind: Hardwareseitig muß eine Rückführung vom Drucker zum Rechner vorhanden sein (s. Bild 4.4a: Anschluß des BEREIT-Signals an den Eingabeport). Vor Ausgabe eines Zeichens ist nun erst zu prüfen, ob Übernahmebereitschaft vorliegt und falls nicht, diese Abfrage zu wiederholen (Bild 4.4c). Dazu ist das oben betrachtete Programm entsprechend Tafel 4.6 um 4 Befehle zu erweitern.

Im Ergebnis können wir feststellen, daß softwareseitig das Abfrageprinzip einfach zu realisieren ist. Es ist nur eine Warteschleife zu programmieren. Der Nachteil dieses Verfahrens besteht aber darin, daß bei langsamer Druckgeschwindigkeit der Prozessor während des gesamten Druckvorgangs blockiert ist. Betrachten wir dazu ein Zahlenbeispiel. Mit den in Tafel 4.6 angegebenen Befehlsausführungszeiten beträgt das Abtastintervall der BEREIT-Leitung 8 µs (Dauer des Durchlaufs durch die Warteschleife). Bei ständiger Bereitschaft des Ausgabegerätes könnte aller 17 µs ein Zeichen ausgegeben werden (Dauer des Durchlaufs von ADR 5 bis zum letzten Befehl). Wird nun ein mechanischer Drucker angeschlossen, der maximal 25 Zeichen/s ausgeben kann, dann sind nur Zeichen im Abstand von 40 ms druckbar. Nach jeder Zeichenausgabe wird folglich 4999mal die Warteschleife durchlaufen, da die BEREIT-Leitung so lange keine Übernahmebereitschaft anzeigt. Erst danach wird der Rechner für 9 µs seine eigentliche Arbeit, die Ausgabe eines weiteren Zeichens, vornehmen können. Anders betrachtet, der Rechner verbringt bei dieser Ausgabeorganisation über 99,9% der Zeit mit Warten auf Ausgabebereitschaft.

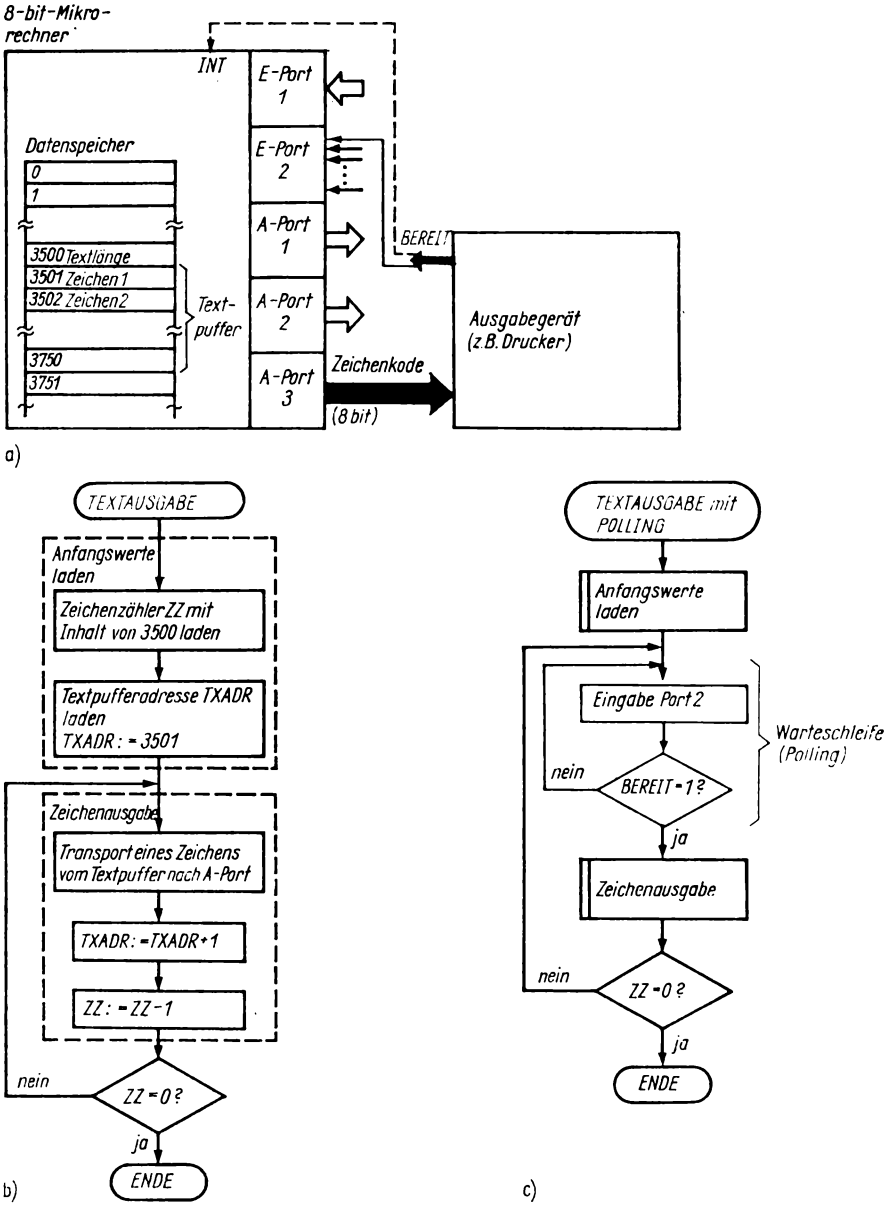


Bild 4.4. Textausgabe (Beispiel 3)

a) Hardwarestruktur; b) Programmablauf ohne Begrenzung der Ausgabegeschwindigkeit; c) Programmablauf mit Polling-Verfahren

Tafel 4.5. Programm für die Textausgabe

Adresse	Befehl	Erläuterung
	TRANSPORT | nach **Reg 3** | von **Sp 3500** (2 µs)	Der Inhalt des Datenspeicherplatzes 3500 wird nach Register 3 transportiert. Vereinbarungsgemäß steht auf diesem Platz die Textlänge.
	LADE | **Reg 1, 2** | mit **3501** (2 µs)	Mit diesem Befehl wird die Anfangsadresse des Text-Speicherbereichs in die Register 1 und 2 geladen. (Da Speicheradressen größer als 8 bit sind, wird aus 2 8-bit-Registern ein 16-bit-Register gebildet.)
ADR 5	**TRANSPORT** | nach **Akku** | von **(Reg 1, 2)** (2 µs)	Dieser Befehl transportiert jeweils ein Zeichen aus dem Datenspeicher in den Akkumulator. Die aktuelle Adresse für den Datenspeicher wird dabei dem Registerpaar 1,2 entnommen (indirekte Adressierung).
	AUSGABE | **Port 3** (2 µs)	Das Zeichen wird über den A-Port 3 dem Ausgabegerät übergeben.
	INC | **Reg 1, 2** (1 µs)	Danach wird die im Registerpaar 1,2 stehende Adresse um 1 erhöht und
	DEC | **Reg 3** (1 µs)	Register 3 zurückgezählt. Damit sind die Datenspeicheradresse und die verbleibende Textlänge korrigiert worden.
	SPRUNG | wenn **NICHT NULL** | *ADR 5* (3 µs)	Durch einen bedingten Sprung nach ADR 5 wird dieser Ablauf solange wiederholt, bis alle Zeichen (Textlänge gleich 0) ausgegeben sind.
	Folgebefehl	

Tafel 4.6. Programm für Textausgabe mit Warteschleife

Adresse	Befehl	Erläuterung
	TRANSPORT | nach Reg 3 | von Sp 3500	Die ersten beiden Befehle bleiben unverändert (s. Tafel 4.5.)
	LADE | Reg 1,2 | mit 3501	
ADR 5	EINGABE | Port 2 (2 µs)	Eingabe des aktuellen Zustands des E-Ports 2 nach Akkumulator
	LADE | Reg 4 | *MASKE* (2 µs)	In Register 4 wird ein Maskenwort geladen, das zusammen mit dem folgenden UND-Befehl den Zustand des BEREIT-Signals prüft und die übrigen Eingänge ignoriert.
	UND | Reg 4 (1 µs)	
	SPRUNG | wenn NULL | *ADR 5* (3 µs)	Ist das Ergebnis 0, dann liegt keine Übernahmebereitschaft vor, und die Abtastung wird durch Rücksprung auf ADR 5 solange wiederholt, bis das BEREIT-Signal den Zustand 1 annimmt.
	TRANSPORT | nach Akku | von (Reg 1,2)	Dann folgen in unveränderter Reihenfolge die übrigen in Tafel 4.5 dargestellten Befehle.

Das Abfrageprinzip (polling) kann daher nur dann angewendet werden, wenn der Rechner während dieser Zeit keine anderen zeitkritischen Aufgaben zu lösen hat. Anderenfalls ist das **Anforderungsprinzip** (Unterbrechungsprinzip) anzuwenden. Hardwareseitig wird dazu das BEREIT-Signal als Unterbrechungssignal genutzt (Bild 4.4a). Softwareseitig muß das Ausgabeprogramm als Unterbrechungsbehandlungsprogramm (UBP) programmiert werden. Was das für Konsequenzen hat, wollen wir im folgenden betrachten.

Mit jeder BEREIT-Meldung (Unterbrechungsforderung) des Druckers ist jeweils ein einzelnes Zeichen vom Rechner auszugeben. Das UBP mußte demzufolge nur die in Tafel 4.7 angegebenen Schritte ausführen. Dieses UBP mit einer Laufzeit von 12 µs wird (bei Zugrundelegung der obigen Zahlenwerte) alle 40 ms aufgerufen. Im Gegensatz zum Programm mit Warteschleife nach dem Abfrageprinzip, das den Rechner vollständig belegte, werden also bei diesem Prinzip nur 0,03% der Rechnerleistung für das Druckprogramm benötigt, da in der Pause zwischen den Unterbrechungen andere Programme arbeiten können.

Tafel 4.7. Unterbrechungsbehandlungsprogramm zur Textausgabe

Adresse	Befehl	Erläuterung
	TRANSPORT | nach **Akku** | von **(Reg 1,2)** (2 µs)	Holen eines Zeichens aus dem Datenspeicher in den Akkumulator und
	AUSGABE | **Port 3** (2 µs)	Ausgabe nach A-Port 3
	INC | **Reg 1,2** (1 µs)	Aktualisieren der Speicheradresse und Textlänge für den nächsten Programmlauf
	DEC | **Reg 3** (1 µs)	
	SPRUNG | wenn **NULL** | *ADR 1* (3 µs)	Ist der Inhalt von Register 3 0 geworden, also der gesamte Text ausgegeben, muß zu einem Programm gesprungen werden, das weitere Unterbrechungen vom Drucker verhindert (wird hier nicht näher betrachtet!).
	RÜCKKEHR (3 µs)	Durch den Rückkehrbefehl wird der bei der Unterbrechung gerettete Programmzähler wieder geholt und damit das unterbrochene Programm fortgesetzt.

Allerdings ist das UBP in der Regel nicht so einfach, wie oben angenommen, realisierbar. Es wurde dabei nämlich vorausgesetzt, daß die Register 1 und 2 ständig für dieses UBP reserviert bleiben, daß also diese Register während der gesamten Zwischenzeit nicht von anderen Programmen benutzt werden dürfen. Eine solche Einschränkung ist bei der begrenzten Anzahl von Registern in den meisten Fällen nicht zulässig. Das UBP muß deshalb dahingehend erweitert werden, daß vor der Ausführung der in Tafel 4.7 dargestellten Befehlsfolge die vom UBP benutzten Register erst „frei gemacht" und mit dem für das UBP erforderlichen Inhalt geladen werden. Das heißt, die Registerinhalte, die zum Unterbrechungszeitpunkt zufällig vorhanden sind, müssen auf Datenspeicherplätze gerettet werden und dafür die Inhalte von 3 ständig für das UBP reservierten Datenspeicherplätzen geholt werden. Nach Ausgabe eines Druckzeichens muß dieser Vorgang entsprechend umgekehrt werden. Das bedeutet, daß die vom UBP zuletzt erzeugten Registerinhalte wieder in den Datenspeicher ausgelagert und die des unterbrochenen Programms zurückgeladen werden müssen. Erst dann ist das UBP beendet, und das unterbrochene Programm kann fortgesetzt werden.

Das in Tafel 4.7 angegebene UBP ist damit durch 14 Transportbefehle zu erweitern. Vor dem 1. Befehl in Tafel 4.7 sind zunächst 4 Transportbefehle zum Retten von Akkumulator, Register 1, 2 und 3 in den Datenspeicher und 3 Trans-

portbefehle zum Holen der für das UBP aktuellen Werte für Register 1, 2 und 3 notwendig. Die entsprechenden 7 Gegenbefehle sind vor dem RÜCKKEHR-Befehl einzuschieben.

Die Laufzeit des UBP wird somit um 28 µs auf insgesamt 40 µs erhöht, d. h., die Belastung des Rechners durch das Druckprogramm beträgt in Wirklichkeit 0,1 %. Dies ist zwar immer noch ein in der Gesamtbilanz vernachlässigbarer Wert, aber es ist ersichtlich, daß bei diesem Organisationsprinzip ein beträchtlicher Zeitanteil für Hilfsarbeiten (Retten und Laden der Register) benötigt wird. Bei Programmstrukturen, die eine Vielzahl von Unterbrechungen und damit von Registerumschaltungen in der Zeiteinheit erfordern, kann der Rechner dadurch jedoch merklich belastet werden. Die Verbesserung der inneren Rechnerstrukturen und die Bereitstellung leistungsfähiger Maschinenbefehle, die solche Aufgaben vereinfachen, sind damit wichtige Ansatzpunkte bei der Entwicklung leistungsfähiger Rechner bzw. Prozessoren für Echtzeitanwendungen. So werden beispielsweise bei manchen Prozessoren Befehle bereitgestellt, die alle Register auf einmal in den Datenspeicher auslagern bzw. zurückholen, oder Registersätze werden mehrfach im Prozessor angeordnet, so daß nur Umschaltungen vorgenommen werden müssen.

4.1.2. Programmresidenz

In den bisherigen Betrachtungen wurde immer folgendes vorausgesetzt:

- Das Programm hat seinen Platz (es **residiert**) im Programmspeicher des Mikrorechners.
- Dieser Programmspeicher wird mit nichtflüchtigen Speicherzellen realisiert, so daß auch nach Ausfall bzw. nach Ausschalten der Stromversorgung das Programm erhalten bleibt.
- Das Programm wird vor der ersten Inbetriebnahme des Rechners in den Speicher geladen (das erfolgt durch Einstecken bzw. Einlöten der Speicherschaltkreise bzw. beim Einsatz von Einchiprechnern durch Programmieren des ROM bereits beim Schaltkreishersteller).

Diese Annahmen sind für die Mehrzahl der Mikrorechneranwendungen in Echtzeitsystemen auch zutreffend. Die folgenden Gründe können aber zu einer davon abweichenden Lösung zwingen:

- Das Programm überschreitet die Kapazität des Programmspeichers. Bei Einchiprechnern liegt eine eng begrenzte Programmspeicherkapazität vor (z. B. 2 Kbyte). Bei mikroprozessororientierten Systemen läßt sich zwar die Kapazität modular erweitern, ist letztlich aber auch durch den Adreßumfang des Mikroprozessors begrenzt.
- Das Programm ist nicht zeitinvariant. Es ist entweder zum Zeitpunkt der Inbetriebnahme noch nicht sicher, ob bereits alle Funktionen erfaßt sind und deshalb im Laufe der Zeit noch Ergänzungen notwendig werden können, oder es ist von vornherein beabsichtigt, eine Reihe von Funktionen erst bei der Nutzung festzulegen und später individuell zu programmieren.

Dieser letzte Fall trifft bei der Anwendung der Rechner in der Datenverarbeitung und in der wissenschaftlich-technischen Rechnung generell zu. Erst der Nutzer entscheidet, welche konkrete Aufgabe vom Rechner bearbeitet werden soll. Der Rechner muß folglich ständig mit anderen Programmen geladen werden. Während bei in großen Zeitabständen erforderlichen Programmänderungen der Austausch der ROM- bzw. PROM-Schaltkreise noch als praktikabel anzusehen ist, scheidet dieses Verfahren bei häufigem Wechsel aus.

Eine Lösung kann dann nur darin bestehen, daß Programmteile jederzeit und mit einfachen Mitteln von außen eingegeben werden können. Hierzu werden die externen Speichereinrichtungen verwendet. In der Mikrorechentechnik domi-

nieren dabei gegenwärtig die Folienspeicher (floppy disc). Für größere Speicherkapazitäten werden Magnetplattenspeicher verwendet. Weniger geeignet sind aufgrund der langen Suchdauer Magnetbandkassettengeräte, die jedoch immer noch schneller arbeiten als die bisher in der Rechentechnik weit verbreitete Lochbandtechnik. Allen diesen Geräten haftet allerdings der gemeinsame Nachteil an, daß sie mechanische Elemente besitzen und damit für viele Anwendungen nicht ausreichend zuverlässig arbeiten. Es sind deshalb für bestimmte Einsatzfälle rein elektronische Lösungen für externe Speicher erforderlich. Hierfür kommen entweder die gleichen, innerhalb des Rechners eingesetzten Schaltkreise (RAMs, ROMs) oder Magnetblasenspeicher (bubble-memory) in Betracht.

Diese externen Speichereinrichtungen sind aber nicht einfach als Erweiterung des Programmspeichers anzusehen. Für den Prozessor sind sie vielmehr nur E/A-Geräte; denn es gilt nach wie vor, daß der Zentralprozessor während der Abarbeitung seines Befehlszyklus den nächsten Befehl nur aus dem Programmspeicher des Rechners holen kann.

Damit erhebt sich die Frage, wie die extern lagernden Programme überhaupt wirksam werden können. Anhand von Bild 4.5 soll diese (auch programmtechnisch nicht einfache) Problematik erläutert werden.

Bild 4.5a zeigt zunächst den für Echtzeitsysteme typischen Fall. Das Gesamtprogramm findet im Programmspeicher Platz und bleibt während der gesamten Lebenszeit des Systems unverändert. Für den Programmspeicher werden bevorzugt ROM-Schaltkreise verwendet.

Bild 4.5b berücksichtigt den Fall, daß wegen zu großen Programmumfangs externe Speichereinrichtungen notwendig sind. Dabei kann nun nicht einfach das Programm geteilt und die überschüssigen Teile extern abgespeichert werden. Vielmehr muß bereits in der Phase der Programmentwicklung ein anderer Weg eingeschlagen werden. Auf jeden Fall ist zu sichern, daß die unmittelbar nach dem Einschalten des Systems abzuarbeitenden Befehle im Programmspeicher stehen. Darüber hinaus ist der Rechner mit spezieller „Intelligenz", also mit einem speziellen Programm auszurüsten, das analysiert, welche Programmteile als nächste abzuarbeiten sind und diese, falls nicht im Programmspeicher vorhanden, vom externen Speicher in den Programmspeicher holt. Dieses **Speicherverwaltungsprogramm** muß selbstverständlich auch ständig im Programmspeicher resident sein. Damit ergibt sich die im Bild 4.5b gezeigte Speicheraufteilung: Ein gewisser Teil des Programms ist ständig im Programmspeicher vorhanden (resident). Dieser Teil des Programmspeichers wird dementsprechend mit ROM- oder PROM-Schaltkreisen realisiert. Der restliche Teil wird aus RAM-Schaltkreisen aufgebaut. Die jeweils als nächste abzuarbeitenden Programmteile werden durch Eingabeoperationen vom externen Speicher in diesen RAM-Bereich geladen. Dieser Teil des Speichers wird somit mehrfach genutzt und von einem speziellen Programm verwaltet. Der Vorteil dieses Verfahrens ist, daß, wenn einmal ein solcher Speicherverwalter programmiert worden ist, ein nahezu unbegrenzter Speicherraum für Programme zur Verfügung steht. Programmveränderungen sind problemlos möglich, indem die Inhalte der externen Speicher ausgetauscht werden (Wechsel der Folienplatten, der Magnetbandkassetten usw.).

Diese Methode hat aber auch wesentliche Nachteile. So werden die Programmlaufzeiten durch die notwendigen Umschlagzeiten u. U. beträchtlich erhöht (Laufzeit des Speicherverwaltungsprogramms und Eingabedauer vom externen Speicher), und folglich wird die Leistung des Rechners reduziert. Darüber hinaus sinkt bei Verwendung elektromechanischer Speichereinrichtungen die Zuverlässigkeit des Systems beträchtlich (Geräteausfall, Störungen bei der Eingabe). Nicht zuletzt werden die Kosten des Systems durch solche Speichereinrichtungen bedeutend erhöht. Dies sind die Gründe dafür, daß bei Echtzeitsystemen, solange wie technisch möglich, die 1. Variante (Bild 4.5a) verwendet wird. Ist der Einsatz externer Speichereinrichtungen jedoch unvermeidbar, so muß zumindest durch geschickte

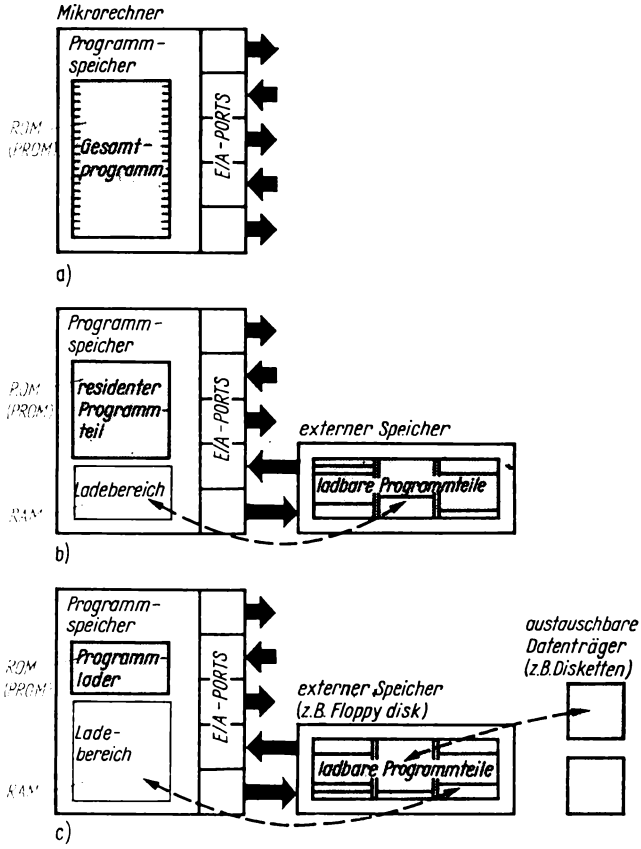


Bild 4.5. Programmresidenz

Programmaufteilung eine gewisse Verbesserung der obengenannten Parameter angestrebt werden (z. B. residente Speicherung der häufig benutzten bzw. der für die Systembereitschaft entscheidenden Programme).

Bei universeller Verwendung eines Mikrorechners ergibt sich dagegen eine Speicheraufteilung gemäß Bild 4.5c. Als residentes Programm wird jetzt nur noch das Anlaufprogramm benötigt. Dieses Programm sorgt nach Einschalten des Rechners dafür, daß ein Programm von außen (von einer externen Speichereinrichtung oder auch über eine Tastatur) eingegeben und gestartet werden kann. Der Programmspeicher wird dementsprechend zum überwiegenden Teil von RAM-Speichern gebildet, und nur ein kleiner Bereich ist mit ROM- oder PROM-Schaltkreisen realisiert. Allein in dem zuletzt genannten Bereich ist diese (oft auch als Anfangslader bezeichnete) Anlaufroutine untergebracht.

In den folgenden Abschnitten wird davon ausgegangen, daß das Gesamtprogramm im Programmspeicher resident ist, da dies der Standardfall beim Einsatz der Mikrorechner in Steuerungssystemen ist. Selbstverständlich existieren inzwischen zahlreiche Mikrorechneranwendungen, die Lösungen entsprechend den Bildern 4.5b und c erfordern. Beispiele dafür sind Bürocomputer und Mikrorechner-Entwicklungssysteme, also Mensch-Maschine-Dialogsysteme, bei denen sowohl an die Reaktionsgeschwindigkeit als auch an die Zuverlässigkeit keine extremen Forderungen gestellt werden.

4.2. Programmiersprachen

Die Programmbeispiele haben gezeigt, welche Detailkenntnisse notwendig sind, um Programme in der „Sprache" des Rechners (des Prozessors) zu schreiben. Von Anfang an waren deshalb in der Rechentechnik Bemühungen zu verzeichnen, einfach handhabbare, schnell erlernbare und auch vom Rechnertyp unabhängige Programmiersprachen zu entwickeln. Als Ergebnis ist inzwischen ein babylonisches Sprachgewirr entstanden. Dies deutet darauf hin, daß es bisher nicht gelungen ist (und wohl auch in Zukunft kaum möglich sein wird), die allen Forderungen gerecht werdende, universelle Sprache zu finden. Es muß vielmehr eingeschätzt werden, daß jede Methode und Sprache ihre spezifischen Vorteile und Nachteile aufweist und folglich abhängig vom Anwendungsfall verschiedene Programmiermethoden nebeneinander ihre Berechtigung haben.

Grundsätzlich können die Programmiersprachen zunächst unterteilt werden in:

- **Assemblersprachen** (maschinenorientierte Sprachen)
- **höhere Programmiersprachen** (problemorientierte Sprachen).

Die für jeden Rechnertyp vorhandene Assemblersprache lehnt sich sehr eng an dessen Maschinensprache an. Die höheren bzw. problemorientierten Sprachen dagegen sind vom Maschinentyp unabhängig.

4.2.1. Programmentwicklung und -übersetzung

Unabhängig, ob ein Programm in Assemblersprache oder in einer höheren Maschinensprache existiert, kann es erst dann in den Programmspeicher eines Rechners geladen werden, wenn es zuvor in dessen Maschinensprache übersetzt worden ist.

Da dieser Übersetzungsvorgang exakt beschreibbar (algorithmisierbar) ist, kann dafür auch ein Rechner benutzt werden. Die dann zu schaffenden Übersetzerprogramme werden als **Assembler** (für die Assemblersprache) bzw. als **Compiler** (für höhere Programmiersprachen) bezeichnet.

Das Grundanliegen bei der Anwendung von über dem Maschinenniveau liegenden Sprachen besteht darin, die Programmierarbeit in 2 Phasen zu trennen. Die 1. Phase umfaßt den schöpferischen Anteil, der vom Programmierer ausgeführt werden muß. In der 2. Phase sind Routinearbeiten zu bewältigen. Sie können automatisiert ablaufen, also von einem Rechner erledigt werden. (Nur am Rande sei hier vermerkt, daß die Grenze zwischen schöpferischer und Routinearbeit nicht klar zu definieren ist. Es wird ständig versucht, den Anteil der schöpferischen Arbeit zu verringern, d. h. die Rechnerunterstützung im Prozeß der Programmentwicklung laufend zu erweitern.)

Solche Programmsysteme zur Unterstützung der Programmierarbeit (auch bereits die Übersetzerprogramme) erfordern aber einen großen Speicherumfang. Bisher war es in der Rechentechnik üblich, den Übersetzungsvorgang auf der gleichen bzw. sogar auf derselben Maschine auszuführen, für die das Maschinenprogramm bestimmt ist. Für andere Lösungen bestand auch kaum Notwendigkeit, da die Rechner bzw. EDV-Anlagen für einen häufigen Nutzerwechsel ausgelegt sind und folglich auch Übersetzungsläufe ausgeführt werden können. Eine andere Situation finden wir dagegen in der Mikrorechentechnik vor. Soll beispielsweise ein Programm für einen Einchiprechner, der über 2 Kbyte Programmspeicher verfügt, entwickelt werden, dann kann dieser kleine Rechner nicht als Sprachübersetzer benutzt werden. Es ist also durchaus üblich und notwendig, für die Programmentwicklung andere Rechner einzusetzen. Dabei bieten sich folgende Möglichkeiten an:

– Verwendung von beliebigen Universalrechnern bzw. EDV-Anlagen. Die erforderlichen Übersetzerprogramme werden dann als **Cross-Assembler** oder **Cross-Compiler** bezeichnet;
– Verwendung eines Mikrorechners vom gleichen Typ, der durch Ausbau seines Speichers, durch externe Speichereinrichtungen und durch geeignete E/A-Geräte speziell für den Zweck der Programmentwicklung ausgerüstet wird. Diese Systeme werden entsprechend als **Entwicklungssysteme** (development system) bezeichnet (Bild 4.6).

Unabhängig davon, welche der beiden Möglichkeiten angewendet wird, erfordert die Programmübersetzung folgende 3 Schritte:

1. Vom Programmierer muß der in einer bestimmten Sprache geschriebene Programmtext, das sog. **Quellprogramm,** manuell über eine Tastatur eingegeben werden. Die eingegebene Zeichenfolge wird vom Rechner unverändert in einem bestimmten Speicherbereich gespeichert und bildet damit eine zusammenhängende Datenmenge (Quelldatei).
2. Nach vollständiger Eingabe wird durch ein Kommando, das ebenfalls über die Tastatur eingegeben wird, der **Übersetzungslauf,** also das Übersetzungsprogramm (Assembler oder Compiler) gestartet. Dieses Programm analysiert Zeichen für Zeichen die Quelldatei und leitet daraus die erforderlichen Maschinenbefehle ab, die als Ergebnisse in einem anderen Speicherbereich aufbewahrt werden. Am Ende steht damit das Maschinenprogramm als eine weitere Datenmenge im Speicher.
3. Danach kann dieses Maschinenprogramm auf verschiedene Weise ausgegeben werden. Für Dokumentationszwecke ist z. B. eine gemeinsame Ausgabe des Quelltextes und des zugehörigen Maschinenprogramms über einen Drucker erforderlich **(programlisting).** Es ist aber auch durchaus möglich, durch Anschluß eines speziellen Programmiergerätes an den Rechner sofort die PROM-(EPROM-)Schaltkreise mit dem Maschinenprogramm zu laden.

Auf jeden Fall steht nach dem Übersetzungslauf das Maschinenprogramm zur

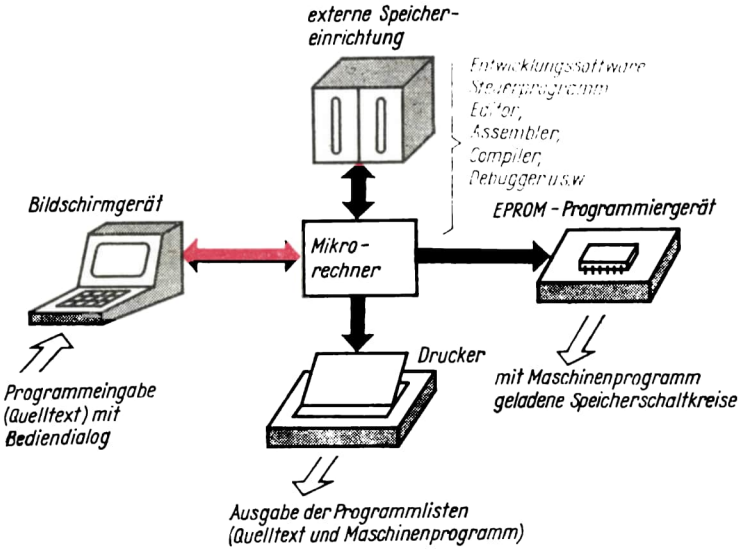


Bild 4.6. Mikrorechner-Entwicklungssystem

Verfügung. Die entsprechenden Schaltkreise können fertiggestellt werden, und ein erster Test des Programms im Originalrechner (-system) kann erfolgen. Beim Feststellen von Fehlern wird dieser Prozeß u. U. mehrfach wiederholt werden müssen.

An einer weiteren Rationalisierung dieses Programmentwicklungsprozesses wird daher ständig gearbeitet. Entscheidende Verbesserungen lassen sich erzielen, wenn dem Entwicklungssystem über die Übersetzerfunktion hinaus weitere Hilfsarbeiten übertragen werden können, indem neben dem Assembler und verschiedenen Compilern weitere Programmsysteme bereitgestellt werden.

An 1. Stelle ist hier die Editorfunktion zu nennen. Als **Editor** wird ein Programmsystem bezeichnet, das die direkte Eingabe und Korrektur eines Programmquelltextes im Dialog ermöglicht. Der Programmierer kann im Prinzip auf Papier, Bleistift und Radiergummi verzichten und dafür ein Bildschirmgerät benutzen. Er gibt die Folge seiner Anweisungen mit seiner individuellen Arbeitsgeschwindigkeit ein. Der Bildschirm liefert den Kontrollausdruck, wobei durch Kommandos jederzeit die Möglichkeit besteht, beliebig zurückliegende Zeilen sichtbar zu machen und zu korrigieren (Streichen und Einfügen von Zeilen, Ändern von Zeichen usw.). Auf diese Weise entsteht papierlos der Quelltext im Speicher des Rechners, der dann dem „Übersetzer" übergeben werden kann.

Eine weitere Beschleunigung der Programmentwicklung läßt sich erreichen, wenn auch der Test des übersetzten Maschinenprogramms noch auf dem Entwicklungsrechner vorgenommen werden kann. Wird als solcher ein anderer Rechnertyp verwendet, so ist zunächst durch ein weiteres Programmsystem der Mikrorechner nachzubilden (Simulatorprogramm). Wird der gleiche Rechner-(Prozessor-)Typ verwendet, ergeben sich relativ einfache Bedingungen, da dieser unmittelbar zum Abarbeiten des erhaltenen Maschinenprogramms benutzt werden kann. Zusätzlich sind nun jedoch Möglichkeiten zur „Beobachtung" der Programmabarbeitung zu schaffen. Es muß also beispielsweise möglich sein, das Programm befehlsweise oder bis zu einem beliebig vorgebbaren Befehl abarbeiten zu lassen und danach den bis dahin eingetretenen Zustand (Inhalt der Register oder Speicherplätze) über Bildschirm oder Drucker anzuzeigen. Danach muß die Abarbeitung des Programms fortgesetzt werden können. Auf diese Weise lassen sich Programmfehler aufspüren. Das für die Steuerung solcher Funktionen erforderliche Programmsystem wird oft als **Debugger** (Entlauser) bezeichnet. Es muß aber betont werden, daß auf diese Weise meist kein vollständiger Programmtest möglich ist, da das zu testende Programm in einer „fremden Umgebung" ablaufen muß. So lassen sich die im Anwendungssystem vorhandenen E/A-Geräte (Meß- und Stelleinrichtungen usw.) und ihr zeitliches Verhalten oft nicht real am Entwicklungsrechner nachbilden.

4.2.2. Assemblersprache (maschinenorientierte Programmierung)

Die Programmierung in Assemblersprache entspricht einer Programmierung in Maschinensprache mit einigen wenigen Bequemlichkeiten. Diese Bequemlichkeiten haben wir bereits bei der Betrachtung der Beispiele für Maschinenprogramme im Abschnitt 4.1.2. eingeführt. Sie sind im wesentlichen auf folgende 3 Punkte beschränkt:

1. Der Programmierer braucht die Kodierung der Maschinenbefehle (also die verschiedenen Kombinationen aus Nullen und Einsen) nicht zu kennen. Er kann sich stattdessen einprägsamerer Bezeichnungen (mnemonics) bedienen. (Wir haben das in den bisherigen Abschnitten bereits praktiziert. So wurde der im Abschnitt 2.2. eingeführte Musterbefehlssatz ausschließlich durch mnemonische Ausdrücke beschrieben.) Das Übersetzerprogramm, der Assembler,

ersetzt die mnemonischen Ausdrücke durch die entsprechenden Kodewörter der Maschinenbefehle.

2. Der Programmierer kann **symbolische Programmspeicheradressen** verwenden. Das Übersetzungsprogramm zählt die ermittelten Maschinenbefehle selbst mit und ersetzt damit die symbolischen durch reale Adressen.
3. Der Programmierer kann im Quelltext **symbolische Angaben für Speicheradressen und Daten** verwenden, muß aber selbstverständlich am Schluß des Programmtextes die konkreten Werte dem Assembler mitteilen. Hierfür existieren spezielle Pseudoanweisungen in der Assemblersprache.

Daraus lassen sich unmittelbar die Grundeigenschaften der Assemblersprache ableiten:

- Eine Assemblersprache ist an einen bestimmten Rechner-(Prozessor-)Typ gebunden. Sie spiegelt dessen kompletten Befehlssatz und Architektur (Registersatz, Organisation des Statusspeichers und des Unterbrechungssystems, Adressierungsarten usw.) wider. (Weisen 2 Prozessoren die gleiche Assemblersprache auf, dann sind sie vom gleichen Typ und können sich bestenfalls in ihren Arbeitsgeschwindigkeiten unterscheiden.)
- Beim Übersetzungsvorgang bleibt die Programmstruktur unverändert, da jede in Assemblersprache geschriebene Anweisung genau einem Maschinenbefehl entspricht.

Daraus resultiert der entscheidende Vorteil der Assemblerprogrammierung: Das nach der Übersetzung erhaltene Maschinenprogramm ist genau so gut (oder schlecht), wie es der Programmierer in der Assemblersprache geschrieben hat.

Das Programmieren auf diesem Niveau verlangt zwar das Erlernen des für jeden Mikrorechner- oder Mikroprozessortyp spezifischen Befehlssatzes, verschafft aber zugleich die Kenntnisse über alle hardwareseitig bedingten Leistungsmerkmale, über die interne Struktur und über die Arbeitsweise. Dadurch wird die Schaffung zeit- und/oder speicherplatzoptimaler Programme möglich.

Das ist auch der Grund für die Anwendung dieses Programmniveaus bei solchen Aufgaben, bei denen es entweder auf kurze Laufzeit oder auf die Begrenzung des Programmspeicher- und Datenspeicherumfangs ankommt. Bei Echtzeitsystemen spielen diese Parameter oft eine dominierende Rolle. Beim Einsatz eines Einchiprechners kann beispielsweise durchaus die Fähigkeit des Programmierers darüber entscheiden, ob die vorliegende Aufgabenstellung mit dem verfügbaren Speicherumfang und damit mit diesem Schaltkreis realisierbar ist oder nicht.

Die Assemblersprache hat auch entscheidende Nachteile. An 1. Stelle ist zu nennen, daß in dieser Sprache geschriebene Programme nur auf Rechnern mit gleichem Prozessortyp laufen können. Sollen gleiche oder ähnliche Aufgaben mit einem anderen Rechner gelöst werden, dann ist das Programm neu zu erstellen. Außerdem sind in dieser Sprache geschriebene Programme schlecht selbstdokumentierend. Das heißt, am Quelltext des Programms ist die konkrete Funktion eines Befehls innerhalb des Programmablaufs und damit der dem Programm zugrunde liegende Algorithmus nur schwer erkennbar. (Selbst der ursprüngliche Programmierer ist meist nicht in der Lage, zu einem späteren Zeitpunkt sein eigenes Programm sofort zu verstehen.) Es ist deshalb bei der Assemblerprogrammierung besonders wichtig, den Programmtext mit zusätzlichen Kommentaren zu versehen. Eine solche **Programmdokumentation** ist unbedingt erforderlich, um Zweck und Besonderheiten des Programms für spätere Aufgaben (Programmmodifikation; Verwendung in anderen Programmsystemen) festzuhalten.

4.2.3. Höhere Programmiersprachen

Die Nachteile der Assemblerprogrammierung werden durch die Verwendung höherer Programmiersprachen weitgehend vermieden. Der entscheidende Vorteil der Programmierung auf höherem Niveau ist, daß das Programm (Quelltext) universell anwendbar in dem Sinne ist, daß es durch Compiler in die verschiedensten Maschinensprachen übersetzt werden, also damit auf beliebigen Rechnern laufen kann. Voraussetzung ist, daß die entsprechenden Compiler zur Verfügung stehen.

Damit ist eine neue Qualitätsstufe bei der Softwareentwicklung erreicht. Es lohnt sich beispielsweise, Programmbibliotheken einzurichten, in denen Programme gesammelt und für mehrfache Nutzung aufbewahrt werden.

Weitere Vorteile der höheren Programmiersprachen sind die höhere Programmdichte und die bessere Selbstdokumentierung des Quelltextes. So lassen sich bei diesen Sprachen relativ lange Maschinenbefehlsfolgen durch wenige Worte Programmtext beschreiben, die auch zugleich den Zweck dieser Befehlsfolge meist klar zum Ausdruck bringen.

Die ersten Ansätze solcher über dem Maschinenniveau liegenden Programmiersprachen gab es bereits in der Zeitetappe, in der Rechner noch ausschließlich für wissenschaftlich-technische Berechnungen eingesetzt wurden. Aus diesem Grund entstanden zunächst Sprachen, die insbesondere die Programmierung von Algorithmen zur numerischen Rechnung (Umsetzung von Formeln) erleichtern sollten, z. B. FORTRAN (**formula translation**) und ALGOL. Für die Behandlung der in der folgenden Etappe dominierenden kommerziellen Anwendungen (Datenverarbeitung) wurde später die Sprache COBOL eingeführt.

Bei diesen Sprachen liegt der Schwerpunkt in der Vereinfachung und Beschleunigung des Programmierprozesses für den Rechneranwender. Dieser braucht nur die seinem spezifischen Problemkreis angepaßte Sprache zu erlernen und sich nicht um die Besonderheiten der jeweils verfügbaren Rechenanlage zu kümmern. Die einmal geschriebenen Programme können auch beim Austausch der Rechner weiter verwendet werden. Die Sprachen sind relativ einfach erlernbar und damit auch für den gelegentlichen Nutzer (Programmierer) geeignet. Aus dieser ursprünglichen Zielstellung resultiert auch die häufig verwendete Bezeichnung **problemorientierte Sprachen.**

Inzwischen spricht man jedoch zunehmend von **höheren Programmiersprachen.** Dies ist darin begründet, daß die neueren Sprachen nicht mehr auf spezielle Problemkreise ausgerichtet sind, sondern vielmehr universell die Softwarearbeit auf ein höheres Niveau heben sollen. Sie sind damit auch nicht für den gelegentlichen Nutzer, sondern für die professionellen Programmierer der System- und Anwendersoftware gedacht, also zur Rationalisierung der umfangreichen Softwarearbeiten, die von zahlreichen Spezialisten der rechnerherstellenden Industrie bzw. der Anwendungsindustrie geleistet werden müssen. Im Vordergrund stehen deshalb bei diesen Sprachen die Programmiereffizienz, der Grad der Selbstdokumentation und natürlich auch die Möglichkeit, die einmal geschaffenen Programme bei Änderung der Hardware weiter verwenden zu können.

Als Beispiele für solche höheren Programmiersprachen sollen genannt werden: PL/M und PLZ (von bedeutenden Mikrorechnerherstellern speziell für diese kleinen Rechner entwickelt), CHILL und ADA (speziell für die Programmierung von Echtzeitsystemen), PASCAL (ursprünglich für die Softwareausbildung entwickelt).

Es kann nicht Anliegen dieses Buches sein, eine bestimmte Programmiersprache zu behandeln. Vielmehr soll versucht werden, das Wesen der höheren Programmiersprachen zu erläutern, um den Unterschied zwischen diesem Sprachniveau und dem Maschinenniveau zu verdeutlichen. Generell ist zunächst festzustellen, daß sich die Vielzahl der höheren Programmiersprachen weniger voneinander

unterscheiden, als es auf den ersten Blick scheint. Die Vielfalt ergibt sich hauptsächlich durch unterschiedliche Darstellungs- und Beschreibungsformen.

Das Grundanliegen einer höheren Programmiersprache besteht darin, Algorithmen in ausschließlich textlicher Form zu beschreiben. Durch Zeichen, Zeichenfolgen (Worte) und Wortfolgen (Satzkonstruktionen) muß all das darstellbar sein, wozu bei der zwischenmenschlichen Kommunikation Kombinationen von Texten und grafischen Elementen bevorzugt werden. Ein Beispiel für solche Kombinationen sind die in diesem Buch bisher immer verwendeten Ablaufpläne als Beschreibungsmittel für Algorithmen. Bei der Entwicklung höherer Programmiersprachen geht man davon aus, daß Algorithmen aus folgenden Grundelementen zusammensetzbar sind (Bild 4.7):

- **Sequenz:** Folge zeitlich nacheinander auszuführender Aktionen;
- **Alternative:** Verzweigung in 2 alternative Programmwege in Abhängigkeit von der Erfüllung einer Bedingung;
- **Zyklus** (Schleife): Mehrfache Wiederholung eines Programmzweiges, wobei das Verlassen der Schleife wiederum von der Erfüllung einer Bedingung abhängt.

Werden nun für diese Grundelemente bestimmte textliche Darstellungen vereinbart, dann läßt sich die gemischt grafische und textliche Darstellung eines Ablaufplans allein durch eine Wortfolge ersetzen. Es entsteht ein Programm in höherer Programmiersprache. Bei Sequenzen ist die Umsetzung einfach. Sie erfolgt entsprechend Bild 4.7a.

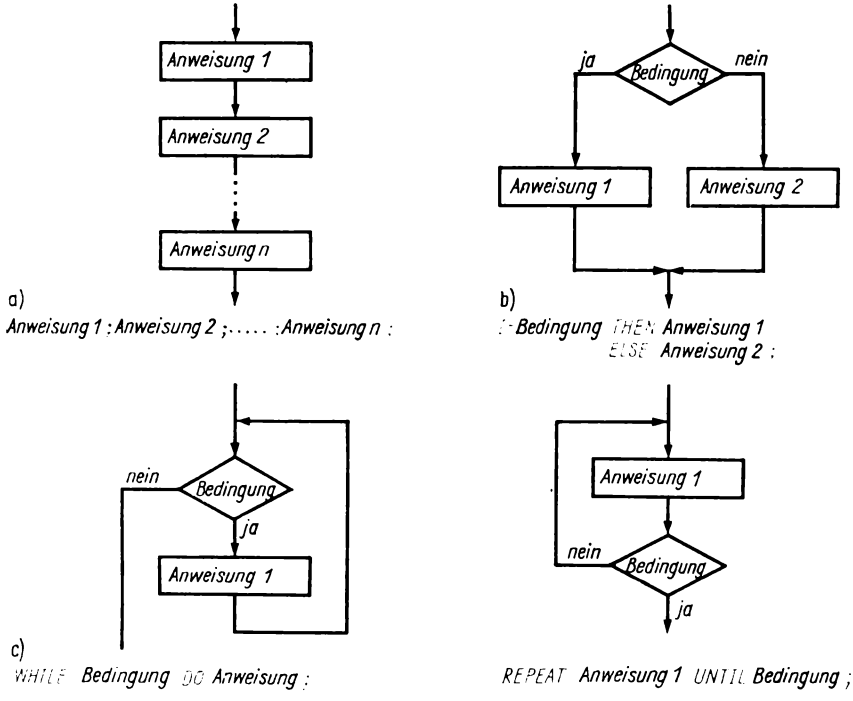


Bild 4.7. Algorithmendarstellung in Form von Ablaufplänen und mit Hilfe höherer Programmiersprachen

a) Sequenz; b) Alternative; c) Schleife (Zyklus)

Tafel 4.8. Durch Compiler für eine einzelne Anweisung erzeugtes Maschinenprogramm

TRANSPORT	nach **Akku**	von **Sp** *A*
TRANSPORT	nach **Reg 1**	von **Sp** *B*
ADD	**Reg 1**	
TRANSPORT	nach **Sp** *Y*	von **Akku**

Der Inhalt des Speicherplatzes *A* wird nach dem Akkumulator transportiert, anschließend der Inhalt von Speicherplatz *B* nach Register 1. Diese Transporte sind erforderlich, da die Addition nur zwischen dem Akkumulator und einem Register ausführbar ist. Anschließend muß das Ergebnis auf einem für *Y* reservierten Speicherplatz transportiert werden. (Die realen Adressen *A*, *B* und *Y* werden vom Compiler selbst eingesetzt.)

Jede Anweisung der höheren Programmiersprache wird durch das Übersetzungsprogramm, durch den Compiler, in eine Folge von Maschinenbefehlen umgesetzt. So wäre z. B. die einfache Anweisung $Y = A + B$ vom Compiler in das in Tafel 4.8 dargestellte Maschinenprogramm zu überführen. Dabei wird vorausgesetzt, daß die Abarbeitung auf einem Rechner mit dem bisher verwendeten Musterbefehlssatz (Tafel 2.1) erfolgen soll und die Operanden *A* und *B* sowie das Ergebnis *Y* jeweils durch ein Rechnerwort kodierbar sind.

Die Bilder 4.7b und c zeigen, wie Alternativen (Programmverzweigungen) und Programmschleifen durch Ablaufpläne bzw. durch für höhere Programmiersprachen typische Satzkonstruktionen beschrieben werden können.

Im Bild 4.8 sind alle Elemente eines Algorithmus beispielhaft zusammengefaßt. Damit wird ersichtlich, daß höhere Programmiersprachen eine bedeutend dichtere Darstellung eines Programms ermöglichen als Assemblersprachen.

Von den höheren Programmiersprachen wird aber noch eine 2. Seite verlangt: Ein für ein Echtzeitsystem erforderliches Programm wird aus einer Menge von Teilprogrammen (Programmodulen) gebildet, die untereinander im Zusammenhang stehen (s. auch folgenden Abschnitt). Um auch diese Grobstruktur eines Programmsystems beschreiben zu können, werden ebenfalls bestimmte Sprachelemente und Satzkonstruktionen in einer höheren Programmiersprache bereitgestellt.

Aus dem oben Gesagten könnte gefolgert werden, daß höhere Programmiersprachen bevorzugt angewendet werden. Dem steht jedoch gegenüber, daß bisher die meisten Programme in Assemblersprache geschrieben werden. Für diese Diskrepanz gibt es eine Reihe von Gründen. Zu den subjektiven Gründen gehören Beibehaltung bisheriger Programmiergewohnheiten, fehlende Kenntnisse über höhere Sprachen usw. Ein wesentlicher objektiver Grund ist jedoch folgender Nachteil, der allen höheren Programmiersprachen (bzw. ihren Compilern) anhaftet:

Jeder Compiler ist letztlich auch nur ein mit einem endlichen Zeitaufwand erstelltes Produkt, ein in ein Programm umgewandelter Übersetzungsalgorithmus. Es ist deshalb nicht zu erwarten, daß einem solchen maschinellen Übersetzer der Überblick und die Erfahrungen „beigebracht" werden können, über die ein geübter Programmierer verfügt. Die Arbeitsweise eines Compilers besteht nur darin, schrittweise die Anweisungen der höheren Sprache zu analysieren und jedem Anweisungstyp eine vorgefertigte Maschinenbefehlsfolge zuzuordnen, die jeweils nur geringfügig modifiziert wird. Im Ergebnis des Übersetzungslaufs entstehen Maschinenprogramme, die in ihrer Mikrostruktur ein standardisiertes Aussehen

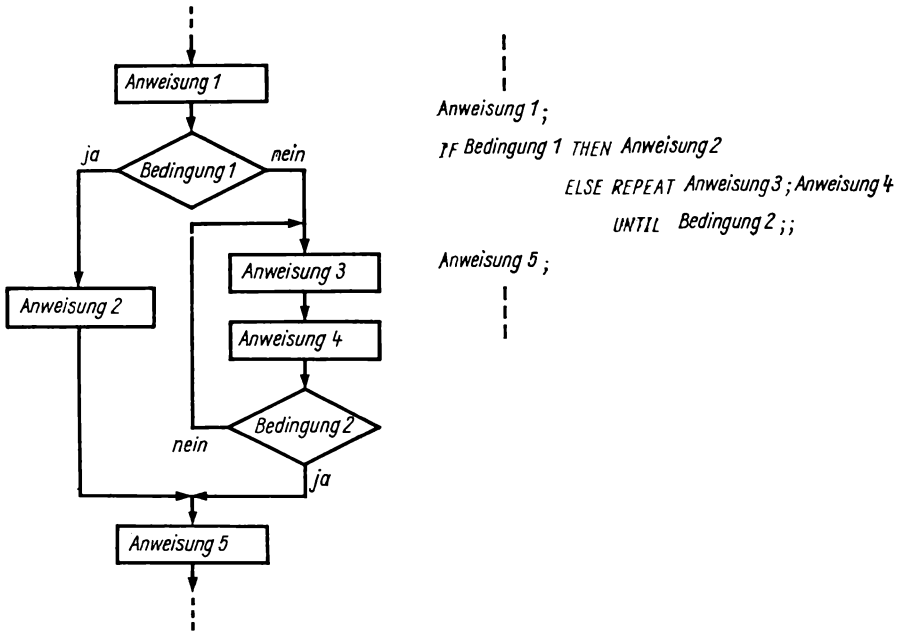


Bild 4.8. Ablaufplan und Beschreibung durch höhere Programmiersprache (Beispiel)

aufweisen und die länger sind, also sowohl mehr Speicherplatz als auch größere Ausführungszeiten benötigen, als die in Assemblersprache formulierten.

Die Entscheidung, welches Programmierniveau anzuwenden ist, sollte allein nach objektiven Kriterien getroffen werden.

Solche Kriterien sind:

- **Programmlaufzeit**
- **Speicherplatzbedarf** des Maschinenprogramms;
- erforderliche **Gesamtzeit für den Programmierprozeß** (von der Aufstellung des Algorithmus bis zum ausgetesteten und dokumentierten Programm);
- **Lesbarkeit (Selbstdokumentation)** des Quelltextes als Faktor für den Einarbeitungsaufwand bei notwendigen Programmänderungen bzw. -erweiterungen;
- Möglichkeit der **Programmverlagerung** auf andere Hardware.

Sind die ersten beiden Kriterien bestimmend, dann ist die Programmierung in Assemblersprache erforderlich. Die anderen Kriterien dagegen führen zu höheren Sprachen.

Die Entscheidung, welches Programmierniveau anzuwenden ist, muß dabei auch nicht für das Gesamtprogramm getroffen werden, sondern ist für die einzelnen Programmodule partiell zu finden, so kann ein umfangreiches Gesamtprogramm gleichzeitig aus in Assemblersprache programmierten Teilen und aus in höheren Sprachen formulierten Modulen zusammengesetzt werden. Eine Vereinigung dieser Module kann sowieso nur auf der untersten Ebene, nämlich auf der Ebene des Maschinenprogramms (also nach den jeweiligen Übersetzungsläufen), erfolgen.

4.3. Strukturierung von Programmsystemen

Nachdem bisher nur die programmseitige Lösung begrenzter Teilaufgaben und verschiedene Möglichkeiten der Programmdarstellung betrachtet wurden, ist es das Ziel dieses Abschnittes, den strukturellen Aufbau eines Gesamtprogramms kennenzulernen.

Dabei soll vorausgesetzt werden, daß nur solche Echtzeit-Rechneranwendungen vorliegen, bei denen das Gesamtprogramm permanent im Programmspeicher resident ist (s. Abschnitt 4.1.2.).

4.3.1. Grundstruktur

Die Grundstruktur eines Echtzeitprogrammsystems zeigt Bild 4.9. Mit Einschalten des Systems (Zuschalten der Betriebsspannung oder nachträgliches manuelles Rücksetzen) wird zunächst ein Programmteil gestartet, dessen Aufgabe die Systeminitialisierung (Systemanlauf) ist. Bei der Behandlung des Mikroprozessors und des Einchiprechners haben wir gesehen, daß für diesen Systemanlauf hardwareseitig ein Steuereingang (RESET) vorhanden ist, dessen Aktivierung den Programmstart ab einer bestimmten Programmspeicheradresse (z. B. Platz 0) erzwingt. Auf diesem Speicherplatz muß entsprechend der 1. Befehl des **Initialisie-**

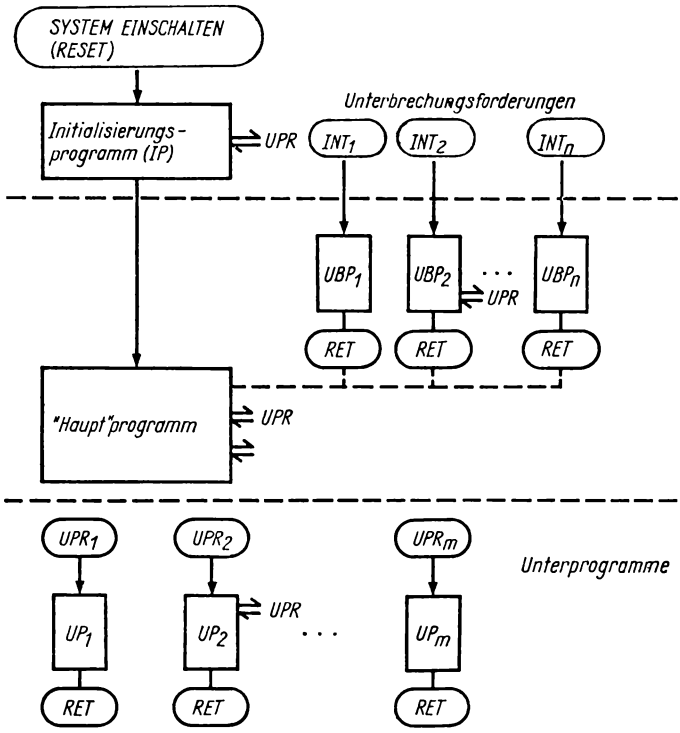


Bild 4.9. Grundkomponenten eines Echtzeitprogramms

INT Unterbrechungssignal (Interrupt); *UBP* Unterbrechungsbehandlungsprogramm; *UBR* Unterprogrammruf; *RET* Rückkehrbefehl (return)

rungsprogramms (IP) stehen, das die folgenden beiden Aufgaben zu übernehmen hat:

– Programmierung der flexiblen Hardwarekomponenten
– Herstellung des System-Anfangszustandes.

Sind hardwareseitig Module (Schaltkreise) eingesetzt worden, deren Funktion im Detail erst durch Laden interner Steuerregister festgelegt wird (z. B. programmierbare Schaltkreise für E/A- und Zeitgeberfunktionen), dann sind Ausgabeoperationen zum Laden dieser Schaltkreisregister erforderlich, die zweckmäßigerweise am Anfang des Programmsystems angeordnet sind. Das gleiche trifft auch für einige interne Register des Prozessors zu (z. B. Füllung des Stapelzeigers, um die Lage des Statusspeichers innerhalb des Datenspeichers festzulegen). Diese ersten Befehle dienen also dazu, die innere Rechnerstruktur zu fixieren. Danach schließt sich als 2. Aufgabe die Herstellung eines Grundzustandes des Gesamtsystems an (Anfangszustand der Stell- und Anzeigefunktionen). Betrachten wir als Beispiel ein System, das als Anzeigeeinrichtung ein Bildschirmgerät aufweist. Beim Einschalten solcher Geräte ist es die Regel, daß sich auch ein zufälliger Inhalt in jenem Speicher einstellt, der die Bildinformation enthält. Dementsprechend würde sich beim Einschalten ein Zufallsbild ergeben. Eine Teilaufgabe des IP muß deshalb die Löschung dieses Speicherbereiches sein (Einschreiben von Leerzeichen). Da dieser Vorgang nur wenige Millisekunden Programmlaufzeit benötigt (die gesamte Initialisierung benötigt nur Bruchteile einer Sekunde), ist er bereits abgeschlossen, bevor die Bildröhre hell wird.

Nach dem Initialisierungsprogramm folgt der eigentliche zentrale Programmteil, der für die Abwicklung der konkreten Steuerungsaufgabe verantwortlich ist. Wir wollen diesen Teil zunächst als **Hauptprogramm** (HP) bezeichnen. In diesem Teil bewegt sich folglich der Rechner, solange das System in Betrieb ist. Eine Ausnahme davon bilden lediglich die **Unterbrechungsbehandlungsprogramme** (UBPs) und die **Unterprogramme** (UPs).

Ist beim Hardwareentwurf eine Anzahl von Unterbrechungssignalen vorgesehen worden, dann muß eine gleiche Anzahl von UBP im Programmsystem existieren, denn jede Unterbrechungsforderung benötigt spezifische Reaktionen. Während des IP sind dabei Unterbrechungen noch gesperrt, um erst die notwendigen Vorarbeiten ungestört durchführen zu können. Mit Start des Hauptprogramms wird jedoch die Unterbrechungssperre aufgehoben. Erst jetzt wird bei Unterbrechungsforderungen ein UBP gestartet. Nachdem dann die notwendigen Sofortmaßnahmen ergriffen wurden, erfolgt am Ende des UBPs der Rücksprung in das an einer beliebigen Stelle unterbrochene Hauptprogramm. Wie Bild 4.9 zeigt, laufen die UBPs alternativ zum Hauptprogramm. Das Hauptprogramm wird auch oft als **Hintergrundprogramm** (background) bezeichnet, da es immer erst dann zum Laufen kommt, wenn keine UBPs mehr aktiv sind.

Die **Unterprogramme (UPs)** bilden eine untergeordnete, separate Programmebene. Unterprogramme sind nichts anderes als Befehlsfolgen, die in allen anderen Programmteilen mehrmals, und zwar an verschiedenen Stellen, benötigt werden, die aber zur Einsparung von Programmspeicherplätzen dort herausgebrochen und nur einmal in den Programmspeicher eingetragen werden. Bild 4.10 verdeutlicht dieses Verfahren an einem Beispiel. Das auf der linken Seite stehende Programm weist an 3 Stellen gleiche Befehlsfolgen auf (die Befehle sind durch Buchstaben symbolisiert). Diese Befehle können folglich als UP herausgelöst werden. In die entstehenden Lücken muß allerdings ein Rufbefehl (Call-Befehl) eingetragen werden, der die Programmverzweigung nach dem Unterprogramm bewirkt. Am Ende des UP muß dementsprechend der Rückkehrbefehl (Return-Befehl) stehen, der die Fortsetzung des Hauptprogramms mit dem auf den Rufbefehl folgenden Befehl bewirkt. Auf diese Weise wird letztlich derselbe Programmablauf erreicht, aber der Programmspeicher wird „verdichtet". Im gezeigten Bei-

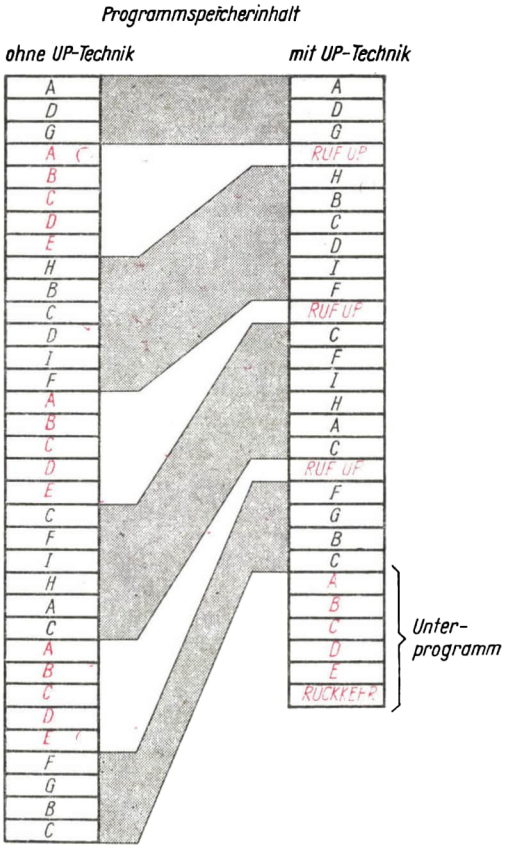


Bild 4.10. Programmverdichtung durch Unterprogrammtechnik

spiel werden nur 6 Speicherplätze gespart, da durch die Ruf- und Rückkehrbefehle zusätzliche Plätze benötigt werden. Es ist offensichtlich, daß sich dieses Verfahren dementsprechend erst bei größeren UPs bzw. bei hohem Wiederholgrad lohnt. Bild 4.9 zeigt, daß dieses Einfügen der als unterste Ebene dargestellten UPs in alle darüberliegenden Programmebenen, aber auch in die UPs selbst erfolgen kann.

4.3.2. Einige Beispiele

Anhand von Beispielen wollen wir nun untersuchen, ob auch die als Hauptprogramm bezeichnete Programmebene noch weiter strukturell untergliedert werden kann. Diese Ebene ist eigentlich die wesentliche Programmebene, da der Prozessor sich, abgesehen von den kurzen Laufzeiten durch IP, UBPs und UPs, während der gesamten Einschaltdauer eines Systems in diesem Hauptprogramm bewegt. (Die Auswahl der folgenden Beispiele erfolgte dabei allein nach dem Gesichtspunkt, die zu behandelnde Problematik verständlich darzustellen. Die Frage, ob es sich dabei um geeignete Rechnereinsatzfälle handelt, stand nicht im Vordergrund.)

Beispiel 1

Nehmen wir den einfachen Fall an, daß eine Verkehrssignalanlage im 24-h-Betrieb mit einer festen Schaltfolge betrieben werden soll. Das wesentliche Merkmal dieser Aufgabe ist, daß keine äußeren Ereignisse (außer System einschalten) vom Rechner „beobachtet" werden müssen. Das Hauptprogramm hat in diesem Fall in abwechselnder Folge 2 Aktivitäten zu vollziehen:

- Ausgabe eines Schaltzustandes der Signallampen und
- Realisierung einer bestimmten Verweilzeit dieses Zustandes, bevor der nächste Schaltzustand folgen darf.

Nach einer endlichen Menge solcher Folgen wiederholt sich der Vorgang periodisch (Bild 4.11).

Das Hauptprogramm wird folglich von einer Programmschleife gebildet, die endlos durchlaufen wird. Der gesamte Programmablauf vollzieht sich vollkommen determiniert. Zur Einsparung von Programmspeicherkapazität würde sich in diesem Fall nur anbieten, die Realisierung der Verweilzeiten als Unterprogramm herauszulösen. Die Realisierung der Zeitdifferenz kann dabei nach dem in Abschnitt 4.1. (Beispiel 2) erläuterten Verfahren erfolgen.

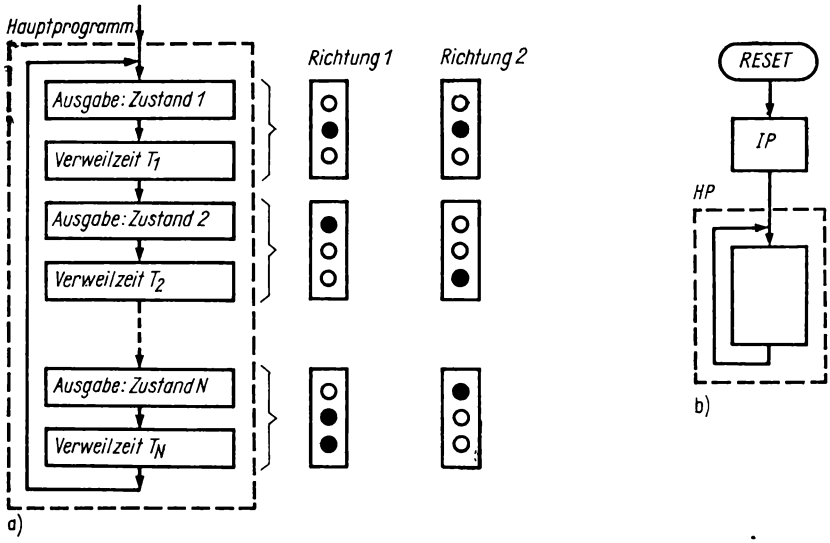


Bild 4.11. Steuerung einer Verkehrssignalanlage (Beispiel 1)
a) Ablaufplan des Hauptprogramms; b) Struktur des Gesamtprogramms

Beispiel 2

Zur Gewährleistung einer lastunabhängigen konstanten Drehzahl eines elektrischen Antriebs soll die im Bild 4.12a gezeigte Anordnung vorliegen:

- Der Zeitabstand T_i der von der Meßeinrichtung gelieferten Impulsfolge ist ein Maß für die momentane Drehzahl. Der Mikrorechner hat die Aufgabe, diese Zeitdifferenz zu bestimmen, mit einem gegebenen Sollwert zu vergleichen und je nach Abweichung vom Sollwert die Steuerspannung, die als Stellgröße für den Elektromotor dient, entsprechend zu korrigieren.

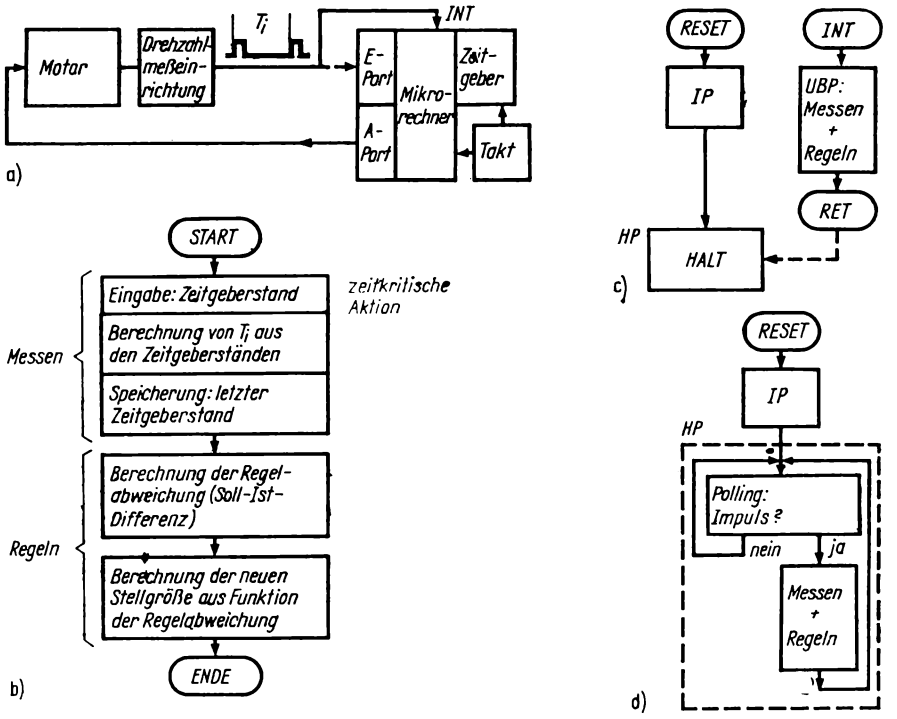


Bild 4.12. Drehzahlregelung (Beispiel 2)

a) Hardwarestruktur; b) Programmablaufplan; c) Programmstruktur bei Unterbrechungsverfahren; d) Programmstruktur bei Polling

IP **Initialisierungsprogramm;** *HP* **Hauptprogramm;** *UBP* **Unterbrechungsbehandlungsprogramm;** *PS* **Polling-Schleife**

- Der Mikrorechner soll zu diesem Zweck mit einem Zeitgeber ausgerüstet sein (s. Abschnitt 3.2.), dessen Arbeitsweise durch das Initialisierungsprogramm wie folgt festgelegt (programmiert) worden ist. Das Zählregister wird von einem Takt, der vom stabilisierten Grundtakt des Mikrorechners durch Teilung intern gewonnen wird, zyklisch weitergezählt. Dieser Zähler durchläuft alle Zustände und beginnt bei Überlauf wieder bei 0. Das Betreiben des Zeitgebers erfordert nach dessen einmaliger Initialisierung keine weiteren Programmaktivitäten.
- Durch eine Leseoperation kann der Stand des Zählregisters vom Prozessor jederzeit ermittelt werden. Bei wiederholten Lesebefehlen ist die Differenz der Zählregisterinhalte folglich dem Zeitabstand dieser Lesevorgänge proportional.

Die Programmstruktur kann nun entsprechend den beiden Möglichkeiten, die zur Erkennung eines Impulses existieren (Unterbrechungs- oder Polling-Verfahren), gewählt werden.

Unterbrechungsverfahren: Jeder Impuls der Meßeinrichtung gilt als Unterbrechungsforderung und löst den Start des UBP aus, das die im Bild 4.12b aufgeführten Schritte ausführen muß, um die Regelungsaufgabe zu lösen.

Betrachten wir anhand einiger konkreter Zahlenwerte die Zeitbilanz dieses Programms: Die Solldrehzahl soll 3000 min^{-1} betragen.

Die Meßeinrichtung gibt während einer Umdrehung 10 Impulse ab. Folglich werden Unterbrechungsforderungen im Abstand von 2 ms erzeugt. Die Programm-

ausführungszeit des UBP wird bei dem erforderlichen Arbeitsumfang von 20 bis 40 Maschinenbefehlen mit etwa 50 µs geschätzt. Daraus folgt, daß die Rechnerbelastung

$$A = \frac{50\ \mu s}{2\ ms} = 25 \cdot 10^{-3} = 2{,}5\%$$

beträgt.

Es ergibt sich die Frage, was macht der Rechner während der Pausen, und welche Aufgabe hat eigentlich das Hauptprogramm, wenn die gesamte Arbeit bereits vom UBP erledigt wird? Bei diesem Beispiel liegt im Vergleich zum Beispiel 1 ein anderer Grenzfall vor. Das Hauptprogramm entartet zu einem „Lükkenfüller". Die einzige Aufgabe dieses Programms besteht darin, auf Unterbrechungssignale zu warten.

Mit dem in Tafel 2.1 eingeführten Befehlssatz würde das Hauptprogramm folglich aus den in Tafel 4.9 angegebenen 3 Befehlen bestehen. Das Gesamtprogramm hätte eine Struktur gemäß Bild 4.12c.

Tafel 4.9. Für Beispiel 2 (Drehzahlregelung) erforderliches Hauptprogramm

Adresse	Befehl	Erläuterung
ADR 6	EININT	Ausschalten der Unterbrechungssperre
	HALT	Prozessor geht in Halt-Zustand. In diesem Zustand vollführt der Prozessor als einzige Aktivität die Abtastung der Unterbrechungssignale. Eine Unterbrechungsforderung bewirkt den Start eines UBP.
	SPRUNG wenn — | ADR 6	Nach Ausführung des UBP erfolgt durch den Rückkehrbefehl die Fortsetzung des Hauptprogramms mit diesem Sprungbefehl, der den unbedingten Sprung an den Anfang bewirkt.

Polling-Verfahren: Anstelle der Anwendung des Unterbrechungsprinzips läßt sich diese Aufgabe auch noch durch Abfrage lösen. Dazu wird die Meßeinrichtung an einen E-Port angeschlossen. Die Programmstruktur ändert sich entsprechend Bild 4.12d. Vor dem eigentlichen Arbeitsprogramm, das die Meß- und Regelaufgaben realisiert, befindet sich ein Programmodul, dessen Aufgabe die ständige Abtastung der Leitung eines E-Ports ist, an der die Drehzahlmeßeinrichtung angeschlossen ist. Es ist also eine Warteschleife zu programmieren, wie wir sie im Abschnitt 4.1. (Beispiel 3) im Prinzip auch angewendet haben.

Der wesentliche Unterschied zwischen beiden Lösungsvarianten ist, daß im letzten Fall der Rechner mit 100% ausgelastet wird, wobei aber während 97,5% der Zeit die Warteschleife durchlaufen wird.
Die eigentliche Arbeitszeit beträgt also wie beim Unterbrechungsprinzip auch nur 2,5%.

Beispiel 3

Da bei der Regelung eines Motors der Rechner 97,5% der Zeit mit Warten beschäftigt wird, liegt es nahe, dem Rechner noch zusätzliche Aufgaben zu übertragen. Wir wollen deshalb als weiteres Beispiel annehmen, daß die Notwendigkeit besteht, an einem Ort mehrere Motoren gleichzeitig zu regeln.

Die Hardwarestruktur ändert sich jetzt dahingehend, daß für jeden Motor jeweils ein Unterbrechungseingang und ein Steuersignalausgang existieren müssen. Programmseitig muß dementsprechend für jeden Motor ein eigenes UBP vorhanden sein.

Es könnte nun versucht werden, den Programmablauf wie im Beispiel 2 vollkommen dem Zufall zu überlassen. Jeder Impuls würde dann das gerade laufende Programm unterbrechen, sein UBP starten und den Stellvorgang auslösen. Die Pausen würden wieder durch das Hauptprogramm mit Warten (Halt-Befehl) überbrückt werden. Bei dieser Ablauforganisation würde jedoch mit bestimmter Wahrscheinlichkeit die Situation eintreten, daß 2 und mehr Unterbrechungsforderungen auflaufen, wodurch eine „Überlagerung" mehrerer UBP notwendig wäre. (Mehrere UBP streiten sich um den gleichzeitigen Besitz des Prozessors.) Hardwareseitig bestünden in diesem Fall nur folgende 2 Möglichkeiten, um diesen „Streit" zu schlichten:

1. Sofort nach Start des UBP wird die Unterbrechungssperre eingeschaltet, indem als erster Befehl des UBP Interruptsperre programmiert wird. Erst nach Beendigung des UBP (Rückkehr-Befehl) wird das Hauptprogramm fortgesetzt und durch dessen ersten Befehl die Unterbrechungen wieder erlaubt. Damit kann ein einmal gestartetes UBP ungestört abgearbeitet werden. Unterbrechungssperre bedeutet dabei nicht, daß Unterbrechungsforderungen verlorengehen. Sie werden nur vom Prozessor nicht beachtet, bleiben aber als Forderungen bestehen und führen sofort nach Unterbrechungsfreigabe zu einer erneuten Unterbrechung (Bild 4.13a).
2. Wird von der Unterbrechungssperre kein Gebrauch gemacht, dann erfolgt durch jedes weitere Unterbrechungssignal eine Unterbrechung des laufenden UBP und der Start eines weiteren.
   Es können sich also im ungünstigsten (wenn auch sehr unwahrscheinlichen) Fall alle UBP ineinander verschachteln (Bild 4.13b). Gelangt ein UBP bis zum letzten Befehl (Rückkehrbefehl), dann wird vom Prozessor automatisch das zuletzt unterbrochene Programm fortgesetzt. Auf diese Weise wird das zuerst gestartete UBP zwangsläufig als letztes beendet.

Beiden Varianten ist gemeinsam, daß keine exakte Angabe über tatsächliche Bearbeitungszeit einer Unterbrechungsforderung möglich ist, sondern diese vielmehr vom Zufall abhängt. Während bei der 1. Variante sich eine Warteschlange von Unterbrechungsforderungen vor dem Rechner bildet, entsteht beim 2. Verfahren eine Warteschlange von noch nicht zu Ende geführten UBP innerhalb des Rechners.

Wir wollen nun analysieren, welches Verfahren für das Beispiel geeignet wäre. Mit den im Beispiel 2 angegebenen Zahlenwerten wäre der Rechner erst beim Anschluß von etwa 50 Motoren in dem Sinne voll ausgelastet, daß keine Pausenzeiten mehr entstehen. Es ist aber andererseits anhand der Bilder 4.13a und b auch offensichtlich, daß sich bereits bei paralleler Regelung weniger Motoren solche Wartezeiten ergeben können, daß eine genaue Regelung nicht mehr gewährleistet ist. Die kritischste Kennziffer ist dabei die Verzögerungszeit zwischen dem Auftreten des Impulses (Unterbrechungsforderung) und der Übernahme des Zeitgeberstandes, da diese Verzögerung sich direkt als Meßfehler bei der Drehzahlmessung auswirkt. Beim 1. Verfahren können daher durch Wartezeiten zwischen Unterbrechungsforderung und Start des UBP und beim 2. Verfahren durch Unterbrechung eines UBP, noch bevor es den Zeitgeberstand übernommen hat, erhebliche Verfälschungen bei der Drehzahlmessung entstehen. Die alleinige Verwendung eines der beiden Verfahren muß deshalb ausscheiden. Günstigere Verhältnisse ergeben sich, wenn beide Verfahren wie folgt kombiniert werden: Die UBPs werden während des zeitkritischen Anfangsteils, der für das Ablesen des Zeitgeberstandes verantwortlich ist, für Unterbrechungen gesperrt (Verfah-

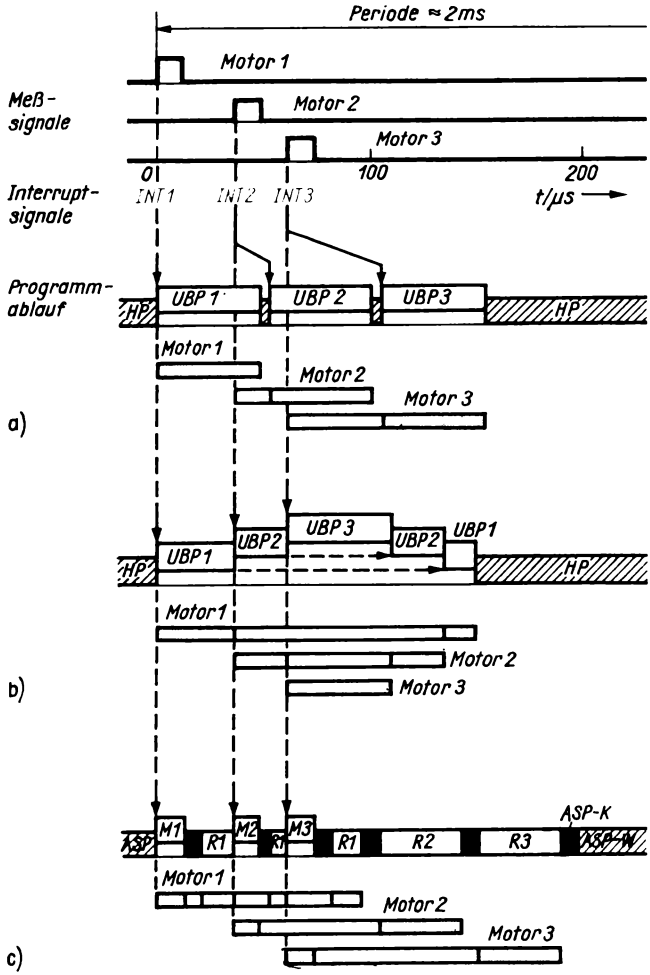


Bild 4.13. Zeitliche Ablauffolge des Programms (Beispiel 3)

a) Anwendung der Unterbrechungssperre während der Unterbrechungsbehandlung
b) verschachtelte Unterbrechungsabarbeitung
c) Steuerung durch Ablaufsteuerprogramm
HP Hintergrundprogramm; *ASP* Ablaufsteuerprogramm; *ASP-K* Kernroutine des *ASP*; *ASP-W* Warteschleife des *ASP*

ren 1), danach aber wieder freigegeben (Verfahren 2). Auf diese Weise wird erreicht, daß die zufälligen Wartezeiten zwischen Unterbrechungsforderung und Start des UBP (und damit der Zeitmessung) wesentlich verringert werden. Die Berechnung des Stellsignals kann jedoch trotzdem noch beträchtlich verzögert werden, so daß unzulässige Verzögerungen zwischen Messung und Stellvorgang auftreten können.

Schlußfolgernd ist festzustellen, daß mit den durch die Hardware gebotenen Möglichkeiten offensichtlich überhaupt keine befriedigende Lösung für diesen Einsatzfall möglich ist. Es muß deshalb versucht werden, durch Software eine andere Ablauforganisation zu erreichen.

4.3.3. Programmgesteuerte Ablauforganisation

Die anhand des oben betrachteten Beispiels gewonnenen Erkenntnisse lassen sich wie folgt verallgemeinern:

Werden die in einem Echtzeitsystem anfallenden Teilaufgaben durch Unterbrechungen gestartet, dann kann es zur Überlagerung mehrerer Unterbrechungen und damit zu zufälligen Bearbeitungszeiten kommen, die u. U. die vom Anwendungsfall geforderten Zeitbedingungen verletzen. Die Wahrscheinlichkeit dafür wird um so größer, je höher die Rechnerauslastung ist. Dieses Prinzip ist damit nur dann anwendbar, wenn entweder unkritische Echtzeitbedingungen vorliegen oder die Rechnerauslastung auch bei der parallelen Abarbeitung mehrerer Aufgaben noch sehr gering bleibt.

Anderenfalls kann der Ausweg nur darin bestehen, daß eine übergeordnete Instanz geschaffen wird, deren Aufgabe die Überwachung und Steuerung der Reihenfolge der Bearbeitung der Teilaufgaben ist. Dies kann bei Verwendung eines Einprozessorsystems nur durch ein Programm erfolgen, das entsprechend als **Ablaufsteuerprogramm** (auch: **Monitorprogramm, Supervisor**) bezeichnet wird.

Dieses Steuerprogramm nimmt innerhalb des Gesamtprogramms eine zentrale Stelle ein und schließt sich unmittelbar an das Initialisierungsprogramm an.

Um uns die prinzipielle Aufgabe und Arbeitsweise eines solchen Programms zu verdeutlichen, soll das im Abschnitt 4.3.2. betrachtete Beispiel 3 weitergeführt werden. Damit jeder Motor eine genaue Drehzahlmessung und möglichst schnelle Stellung erfährt, wäre folgende Ablauforganisation anzustreben:

a) Die von den Impulsen ausgelösten Unterbrechungen müssen sofort angenommen und der aktuelle Zeitgeberstand abgespeichert werden, um den Meßfehler gering zu halten (Ausführung der zeitkritischen Aktionen);
b) Die Stellsignale sollen danach in der Reihenfolge berechnet werden, wie die Unterbrechungsforderungen aufgetreten sind (Abfertigung nach dem FIFO-Prinzip).

Um eine solche Ablauforganisation zu ermöglichen, kann nicht mehr die gesamte Arbeit durch das UBP geleistet werden. Es muß vielmehr eine Aufspaltung in 2 Programme erfolgen. Das Programm „Messen" führt die zeitkritischen Aktionen, die sich auf das Ablesen des Zeitgeberstandes beschränken, aus. Dieses Programm muß dementsprechend als UBP laufen. Alle anderen Aktionen (Vergleich zwischen Soll- und Istwert, Berechnung und Ausgabe der Stellgröße) werden durch das Programm „Regeln" übernommen. Dieses Programm kann durchaus etwas verzögert gestartet werden und muß sich nicht unmittelbar an das UBP „Messen" anschließen.

Einen solchen, nicht durch Hardware unterstützten Ablauf durchzusetzen, ist die Aufgabe des Ablaufsteuerprogramms, indem es die Zuteilung des Prozessors zu den anstehenden Aufgaben (Programmen) übernimmt.

Es ergibt sich damit die im Bild 4.14 dargestellte Strukturierung des Gesamtprogramms. Dem Ablaufsteuerprogramm sind nur die UBPs „Messen" übergeordnet. Die Programme „Regeln" sind dem Steuerprogramm untergeordnet und sollen insgesamt als **Arbeitsprogramme** (APs) bezeichnet werden. Darunter schließt sich die Ebene der Unterprogramme an (im Bild nicht dargestellt).

Der zeitliche Ablauf ist, um den Vergleich mit den beiden hardwaregesteuerten Varianten zu ermöglichen, im Bild 4.13c bereits angegeben worden. Dieses Bild soll im folgenden erläutert werden: Solange keine andere Arbeit vorliegt, wird der Prozessor durch das Ablaufsteuerprogramm (ASP) belegt, das seinerseits in den Halt-Zustand übergeht, wenn es keine Funktionen mehr auszuführen hat und auf Unterbrechungsforderungen wartet. Sobald ein Unterbrechungssignal von einem Motor i eintrifft (ausgelöst durch einen Impuls von dessen Drehzahlmeßeinrichtung), wird sofort das UBP M_i gestartet. Dieses Programm sperrt weitere Unter-

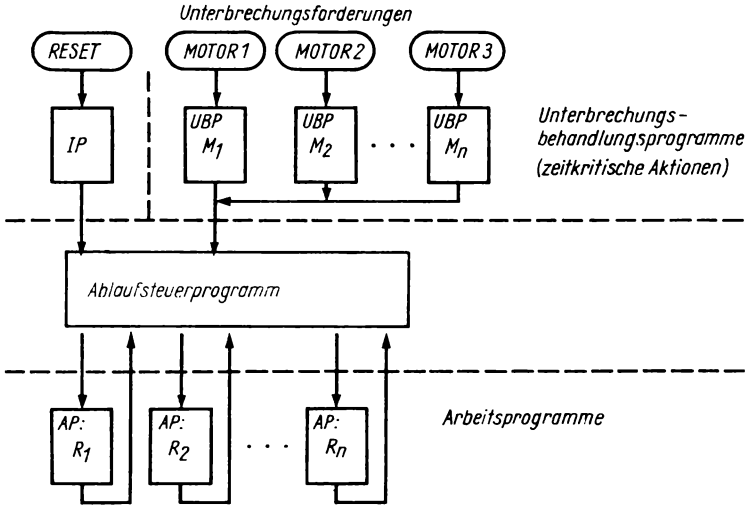


Bild 4.14. Programmstruktur für Beispiel 3
AP Arbeitsprogramme; *M* Programm „Messen" ; *R* Programm „Regeln"

brechungsannahmen, liest den aktuellen Zeitgeberstand und — das kommt bei dieser Organisation als neue Aktivität hinzu — erzeugt (hinterlegt im Speicher) eine Meldung an das Ablaufsteuerprogramm, mit der die Notwendigkeit zur Abarbeitung des Programms R_i (Regeln des Motors i) angezeigt wird (Anmeldung eines Arbeitsprogramms durch ein UBP). Damit sind die dringendsten Aktionen erledigt, und das UBP gibt den Prozessor an das ASP ab.

Das ASP untersucht, wann immer es die Regie erhält, zunächst, ob Anmeldungen für APs vorliegen. Liegt nur eine Anmeldung vor, wird dieses AP gestartet. Sind mehrere Anmeldungen aufgelaufen, dann hat das ASP nach vorgegebenen Entscheidungsregeln das dringlichste AP auszuwählen und zu starten. Liegt dagegen gar keine Anmeldung für ein AP vor, so geht das ASP wieder in den Halt-Zustand. In allen 3 Fällen muß so schnell wie möglich die Unterbrechungssperre ausgeschaltet werden (z. B. mit Start des AP und vor Eintritt in den Halt-Zustand des ASP). Jede Unterbrechungsforderung bewirkt damit den Abbruch eines AP und Start eines anderen UBP M_i. Nach Beendigung dieses UBP wird jedoch nicht das unterbrochene AP fortgesetzt, sondern immer erst das ASP gestartet, das erneut darüber entscheidet, welches AP gestartet bzw. fortgesetzt werden soll. In unserem Beispiel wird immer das unterbrochene AP fortgesetzt, da alle APs als gleich dringlich angesehen werden (s. Bild 4.13c). Der Vergleich der 3 Varianten im Bild 4.13 läßt erkennen, daß mit der Ablauforganisation c) deutliche Verbesserungen erreichbar sind. Die Bearbeitungszeiten für einen Regelvorgang sind in den Fällen a) und c) nahezu gleich. Der dem Fall a) anhaftende Nachteil, daß Meßfehler durch Wartezeiten infolge langer Unterbrechungssperren auftreten, ist jedoch im Fall c) beseitigt.

Fassen wir diese Erkenntnisse verallgemeinert zusammen:

Führt die Programmierung von Echtzeitsystemen zu dem Problem, daß mehrere Programme zur Gewährleistung der Echtzeitforderungen um den Prozessor konkurrieren und die hardwareseitige Unterbrechungsorganisation

dafür keine ausreichende Lösung bietet, dann (und nur dann) muß versucht werden, durch eine softwaremäßig realisierte Ablaufsteuerung zu besseren Resultaten zu kommen. Das einzuführende Ablaufsteuerprogramm erhält dazu einen den eigentlichen Arbeitsprogrammen übergeordneten Rang.

Es ist allerdings zu bemerken, daß diese Lösung nicht unbedingt zum Erfolg führen muß. Das Ablaufsteuerprogramm benötigt seinerseits Prozessorzeit und Platz im Programmspeicher. Es muß also jeweils abgeschätzt werden, ob der zusätzliche Organisationsaufwand (in der Rechentechnik ist hierfür der Begriff Overhead üblich) in einem entsprechenden Verhältnis zu der erreichten Leistungssteigerung steht. (Es besteht die Gefahr, den Rechner durch die Verwaltungsaufgaben in solch einem Maße zu beanspruchen, daß seine eigentliche „Nutz"-Leistung gegen 0 geht.)

Führt also auch diese Maßnahme nicht zum Erfolg, dann ist der in der Aufgabe enthaltene Arbeitsumfang nicht mit der vorliegenden Rechnerstruktur (Hardware) zu lösen, und es sind leistungsfähigere Hardwarekomponenten zu wählen.

4.4. Universelles Konzept für Echtzeitprogramme (multitasking)

Ziel dieses Abschnittes ist, die im vorhergehenden Abschnitt anhand von Beispielen dargelegten Probleme bei der Strukturierung des Gesamtprogramms zu verallgemeinern und eine Betrachtungsweise einzuführen, wie sie für die Programmentwicklung umfangreicher Echtzeitsysteme üblich ist.

4.4.1. Prozeßbegriff

Die Analyse des zuletzt betrachteten Beispiels 3 läßt noch einen weiteren Aspekt erkennen. Für jeden zu regelnden Motor sind zwar jeweils ein eigener Unterbrechungseingang, eine eigene Ausgangsleitung und eigene Speicherplätze für Sollwert, Istwert und Momentanzustand der Stellgröße erforderlich, aber die auszuführenden Aktionen in den Programmteilen „Messen" und „Regeln" sind identisch, so daß eigentlich kein Grund besteht, wie bisher angenommen, für jeden Regelkreis getrennte Programme einzusetzen.

Dies ist eine für die weiteren Betrachtungen wichtige Erkenntnis: Es kann also durchaus der Fall eintreten, daß – wird der Zustand des Programmablaufs zu einem bestimmten Zeitpunkt betrachtet – ein und dasselbe Arbeitsprogramm zur Lösung verschiedener Aufgaben benutzt werden kann. Dabei kann selbstverständlich bei einem Einprozessorsystem nur jeweils eine Aufgabe zur Zeit (Programm) bearbeitet werden. Weitere Aufgaben, die das gleiche Programm verwenden, können sich jedoch in Warteposition befinden. Das heißt, diese Aufgaben sind entweder angemeldet oder können sogar schon teilweise ausgeführt sein, sind aber unterbrochen und danach vom Ablaufsteuerprogramm zurückgestellt worden.

Die bisher praktizierte Betrachtungsweise, wonach die Gesamtfunktion des Rechners als zeitliche Folge nacheinander abzuarbeitender Programme aufgefaßt wird, scheint für einen solchen Ablauf nicht zweckmäßig zu sein. Aus diesem Grund hat sich für den Programmentwurf bei Echtzeitsystemen eine andere Darstellung durchgesetzt, die auf dem Begriff Prozeß (als Abkürzung von Rechenprozeß) basiert.

Unter einem **Prozeß** (engl. **task**, Aufgabe) wird eine abgeschlossene Teilaufgabe verstanden, die durch ein Programm realisiert wird, das Operationen an einer

Eingangsdatenmenge ausführt und damit eine Ausgangsdatenmenge erzeugt. Dabei können verschiedene Prozesse jeweils unterschiedliche Programme und Datenmengen, aber auch die gleichen Programme und/oder Datenmengen benutzen.

Die für einen konkreten Einsatzfall zu realisierende Gesamtfunktion ist dementsprechend in eine Menge solcher Prozesse (tasks) zu zerlegen. Die Prozesse bilden die kleinste Einheit bei der Ablauforganisation. Die Abarbeitung des Programms für einen solchen Prozeß ist dabei ein determinierter Vorgang. Die Ablauffolge der Prozesse untereinander wird dagegen meist ein zufälliger Vorgang sein. Die Prozesse können untereinander um den Prozessor und eventuell um weitere Betriebsmittel konkurrieren.

Unter **Betriebsmittel** sollen dabei alle Hardwareeinrichtungen (Speicherbereiche, E/A-Geräte usw.) verstanden werden, die zur Durchführung des Prozesses notwendig sind.

Um trotz des Zufallcharakters der Prozeßaufrufe eine geordnete Abarbeitungsreihenfolge der Prozesse (im Sinne des Einhaltens kritischer Zeitbedingungen) zu gewährleisten, ist ein Ablaufsteuerprogramm erforderlich, das entsprechend auch als **Multitask-Steuerprogramm** oder **Echtzeitsteuerprogramm** bezeichnet wird. Ein solches Programm kann je nach Aufgabenstellung im Detail sehr unterschiedlich aufgebaut sein. Die folgenden Betrachtungen sollen dazu dienen, dessen grundsätzliche Arbeitsweise genauer kennenzulernen.

4.4.2. Wechselwirkung zwischen Prozessen

Der einfachste Fall liegt vor, wenn die Gesamtfunktion durch einen einzigen Prozeß realisiert werden kann. Dann erübrigt sich der Einsatz eines Ablaufsteuerprogramms. (Die Beispiele 1 und 2 im Abschnitt 4.3. entsprachen diesem Fall.)

Ist eine Gliederung in mehrere Prozesse notwendig, dann können zwischen den Prozessen unterschiedliche Wechselwirkungen auftreten. Ein Grenzfall liegt vor, wenn die Prozesse voneinander vollkommen unabhängig sind, wie das im Beispiel 3 des Abschnittes 4.3. der Fall war. Der einzige Berührungspunkt der Prozesse „Regeln Motor" besteht bei diesem Beispiel darin, daß sie um den Prozessor konkurrieren. Man spricht in diesem Fall von **konkurrierenden Prozessen.** Die Aufgabe des Ablaufsteuerprogramms besteht damit nur in deren Koordinierung. Dieser Fall tritt immer dann auf, wenn allein wegen einer besseren Rechnerauslastung Prozesse überlagert werden. Im Grunde sind diese Aufgaben mit separaten Prozessoren der Rechnern lösbar, ohne daß zwischen ihnen Informationen ausgetauscoht werden müssen.

Bei den meisten Einsatzfällen existieren jedoch kausale Zusammenhänge zwischen den Prozessen. So kann beispielsweise ein Prozeß einen oder (je nach der Erfüllung vorgegebener Bedingungen) verschiedene Nachfolgeprozesse haben, oder ein Prozeß kann Daten oder Betriebsmittel benötigen, die auch noch von anderen Prozessen genutzt werden usw. Es liegen damit **kooperierende Prozesse** vor. Das Ablaufsteuerprogramm hat dann neben der Koordinierung (Zuordnung des Prozesses zu einem Prozeß) noch die Aufgabe der Synchronisation (Beachtung der notwendigen zeitlichen Reihenfolge) zu lösen.

Betrachten wir dazu ein Beispiel: Das Beispiel 3 soll dahingehend modifiziert werden, daß immer dann, wenn ein Motor nicht mehr auf Sollwert regelbar ist (Ausfall eines Motors, Überlast usw.), die Mitteilung „Störung Motor i" über Drucker oder Bildschirmgerät ausgegeben wird. Gegenüber der im Bild 4.14 gezeigten Struktur kommt ein Prozeß hinzu, der die Aufgabe hat, einen Text an das Ausgabegerät zu senden. (Im Abschnitt 4.1. hatten wir das erforderliche Maschinenprogramm aufgestellt.)

Außerdem ist das AP „Regeln" noch zu modifizieren. Es müssen Aktionen eingebaut werden, die feststellen, ob die Regelabweichung einen vorgegebenen Grenzwert überschreitet und auch nach mehreren Stellvorgängen nicht verringert werden kann. Tritt dieser Fall bei Motor i ein, dann muß der Prozeß „Regeln i" den Prozeß „Störung" beim Multitask-Steuerprogramm anmelden.

Sobald nun keine dringlicheren Prozesse mehr vorliegen, wird dieser Prozeß gestartet und die Störungsmeldung ausgegeben. Dabei gilt es aber zu beachten, daß dieser Prozeß eine im Verhältnis zu den anderen Prozessen sehr lange Laufzeit aufweist; mit einem mechanischen Druckwerk werden dazu je nach zulässiger Druckgeschwindigkeit bis zu 2 s benötigt.

Dieser Prozeß der Störungsmeldung wird damit laufend gestört werden, indem ihn eintreffende Unterbrechungsforderungen von den Drehzahlmeßsystemen aus dem Prozessor verdrängen. Nach Abarbeitung des UBP muß vom Multitask-Steuerprogramm erneut entschieden werden, welcher Prozeß fortgesetzt wird. Dabei wird dieser Ausgabeprozeß die geringste Dringlichkeit erhalten und erst dann wieder gestartet werden, wenn kein Regelprozeß mehr auf den Prozessor wartet.

Die Kooperation zwischen den Prozessen besteht bei diesem Beispiel darin, daß die voneinander unabhängigen Regelprozesse einen gemeinsamen Nachfolgeprozeß, den Prozeß „Störung" bewirken können. Dabei kann durchaus der Fall eintreten, daß mehrere Regelprozesse diesen Ausgabeprozeß gleichzeitig anfordern.

Dann müssen diese Anforderungen sequentiell bedient werden, da sonst eine Mischung der Texte auf dem Ausgabegerät erfolgt. Anhand dieses einfachen Beispiels ist bereits erkennbar, daß eine Reihe unterschiedlicher Bedingungen, die aus dem speziellen Anwendungsfall hervorgehen, bei der Verwaltung der Prozesse durch das Steuerprogramm zu beachten sind.

4.4.3. Prozeßzustände

Aus den bisherigen Überlegungen folgt, daß die Prozesse verschiedene Zustände einnehmen können. Wir wollen das folgende Grundmodell einführen, wonach sich die Prozesse zu jedem Zeitpunkt in einem von 3 Zuständen befinden (Bild 4.15):

1. AKTIV: Das dem Prozeß zugeordnete Programm befindet sich im Prozessor in Abarbeitung (Prozeß „rechnet").
2. EIN: Der Prozeß befindet sich in Warteposition und verlangt nach Zuweisung des Prozessors (Prozeß ist „rechenwillig").
3. AUS: Der Prozeß benötigt zu diesem Zeitpunkt keine Prozessorzuweisung.

Bei einem Einprozessorsystem kann dabei höchstens ein Prozeß aktiv sein. Alle übrigen Prozesse befinden sich dementsprechend im EIN- oder AUS-Zustand.

Die Übergänge zwischen diesen Zuständen werden durch eine Reihe spezifischer Ereignisse ausgelöst, die in Tafel 4.10 aufgelistet sind.

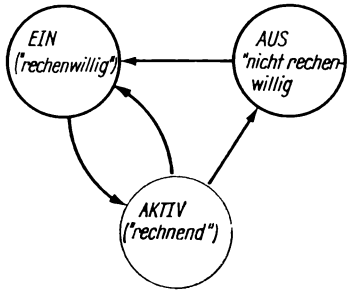


Bild 4.15. Prozeßzustände

Tafel 4.10. Zustandsübergänge der Prozesse und ihre Ursachen

Übergang	Ursachen
AKTIV → AUS	1. Der Prozeß ist beendet und gibt den Prozessor ab. 2. Der Prozeß ist noch nicht beendet, benötigt aber zum weiteren Ablauf ein Betriebsmittel oder das Eintreffen eines äußeren Ereignisses. Da mit großer Wahrscheinlichkeit damit Wartezeiten entstehen, gibt der Prozeß freiwillig den Prozessor frei.
AUS → EIN	Ein Prozeß kann eingeschaltet (angemeldet) werden durch 1. ein UBP, das aufgrund eines externen Ereignisses gestartet wurde; 2. ein UBP, das aufgrund eines internen Ereignisses gestartet wurde (z. B. Zeitgeberunterbrechung); 3. durch ein anderes Arbeitsprogramm.
EIN → AKTIV	Das Ablaufsteuerprogramm analysiert alle im EIN-Zustand befindlichen Prozesse. Der Prozeß mit der momentan höchsten Dringlichkeit wird ermittelt und gestartet.
AKTIV → EIN	Der aktive Prozeß wird durch das Steuerprogramm zugunsten eines Prozesses mit höherer Dringlichkeit zwangsweise in den EIN-Zustand zurückgestellt. (Diese Situation kann nur dann eintreten, wenn während der Abarbeitung eines Prozesses eine Unterbrechung erfolgt, die bei ihrer Bearbeitung (UBP) eine Prozeßneuanmeldung bewirkt, so daß danach eine andere Dringlichkeitsstruktur entsteht.)

Nach diesem Zustandsmodell können sich also sowohl in der EIN- als auch in der AUS-Menge Prozesse befinden, die bereits teilweise abgearbeitet sind.

4.4.4. Multitask-Steuerprogramm

Nach den Vorbetrachtungen über die Zustände, die ein Prozeß annehmen kann, und die Ereignisse, die zu einem Zustandswechsel führen, läßt sich nun genauer angeben, welche Programmkomponenten ein Ablaufsteuerprogramm enthalten muß, d. h., welche innere Struktur der im Bild 4.14 mit Ablaufsteuerprogramm bezeichnete Programmblock besitzt. Das Ergebnis zeigt Bild 4.16.

Es soll im folgenden näher erläutert werden:
Die zentrale Aufgabe eines solchen Programms ist die Auswahl des dringlichsten Prozesses aus der Menge der eingeschalteten (rechenwilligen) Prozesse und der Start des zugehörigen Arbeitsprogramms. Dieser Programmteil des Multitask-Steuerprogramms soll dementsprechend als **Kernroutine** bezeichnet werden. Aus Tafel 4.10 folgt, daß diese Kernroutine bei 4 folgenden Ereignissen aktiv werden muß:

1. Ein AP ist beendet und gibt den Prozessor an das Steuerprogramm zurück („AP-Ende").
2. Ein AP gibt den Prozessor an das Steuerprogramm zurück, da es auf ein Ereignis warten muß („AP-Warten").
3. Ein UBP ist beendet und gibt den Prozessor an das Steuerprogramm zurück („UBP-Ende").
4. Das IP ist beendet und ein erstes AP soll gestartet werden („IP-Ende").

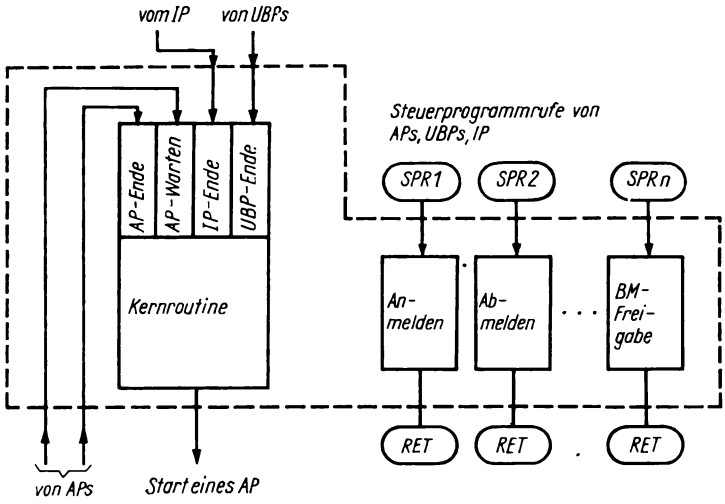


Bild 4.16. Aufbau eines Multitask-Steuerprogramms

Die Kernroutine benötigt für ihre Entscheidungsfindung Informationen über alle eingeschalteten Prozesse. Dazu sind im Datenspeicher auf festgelegten Speicherplätzen für jeden Prozeß Statusworte reserviert, in denen die aktuellen Eigenschaften jedes Prozesses vermerkt sind, z. B. Zustand des Prozesses, Dringlichkeit, Start- bzw. Fortsetzungsadresse des zugehörigen Arbeitsprogramms usw. Für jeden Prozeß sind damit mehrere Speicherplätze erforderlich, die insgesamt als **Prozeßstatusblock** bezeichnet werden sollen. Die Menge aller Prozeßstatusblöcke bildet damit die Eingangsinformation für die Kernroutine.

Die Strategie, nach der die Kernroutine den Folgeprozeß auswählt, kann verschieden realisiert werden und hängt vom Anwendungsfall ab. Mögliche Varianten sind:

1. FIFO-Prinzip: Die Prozesse werden in ihrer Anmeldungsreihenfolge gestartet.
2. Zyklische Zuweisung: Die Prozesse werden entsprechend einer einmalig festgelegten Reihenfolge gestartet.
3. Statische Priorität: Die Prozesse erhalten einmalig eine Dringlichkeitsstufe, die für ihre Auswahl entscheidend ist.
4. Dynamische Prioritäten: Die Prozesse werden entsprechend ihrer Dringlichkeitsstufe gestartet, die aber nicht einmalig vereinbart ist, sondern während der Arbeit des Systems verändert werden kann.

Nachdem nun die Kernroutine den nächsten Prozeß ausgewählt hat, muß sie die notwendigen Voraussetzungen für den Start des zugehörigen AP schaffen, d. h., es müssen eventuelle Anfangswerte in die Prozessorregister geladen und zuletzt durch einen unbedingten Sprungbefehl die Start- bzw. Fortsetzungsadresse des AP in den Programmzähler des Prozessors transportiert werden. Mit diesem letzten Befehl der Kernroutine gibt das Steuerprogramm den Prozessor an das AP ab.

Das Steuerprogramm erhält erst dann den Prozessor wieder, wenn entweder das AP beendet ist oder freiwillig in den Wartezustand (AUS-Zustand) geht bzw. wenn infolge einer Unterbrechungsforderung ein UBP gestartet und beendet worden ist.

Neben dieser Kernroutine umfaßt das Steuerprogramm noch eine Reihe von Programmen, deren Aufgabe die Erzeugung, Änderung und Verwaltung der Prozeßstatusblöcke ist, also jener Datenspeicherplätze, die die Prozeßeigenschaften aufbewahren. Diese Programme sind ihrem Charakter nach Unterprogramme, da sie von den APs und UBPs gerufen werden, um jeweils spezifische Eintragungen und Änderungen in den Prozeßstatusblöcken vorzunehmen. Betrachten wir einige Beispiele:

a) Ein UBP muß die Möglichkeit haben, einen Folgeprozeß P_i anzumelden, dessen Abarbeitung aufgrund der Unterbrechungsforderung notwendig wird. Zu diesem Zweck ruft es das Programm „Anmelden" und übergibt die Prozeßnummer i. Das Programm „Anmelden" ändert daraufhin im Prozeßstatusblock dieses Prozesses P_i den Status von AUS in EIN. Damit wird dieser Prozeß in die Menge der rechenwilligen Prozesse eingereiht. Durch einen Rückkehrbefehl am Ende des Programms „Anmelden" erfolgt der Rücksprung ins UBP.
b) Weiterhin muß die Möglichkeit bestehen, ein Betriebsmittel durch einen Prozeß zu belegen („BM belegen"), dessen Belegungszustand zuvor abzurufen („BM Zustand") und es wieder freizugeben („BM Freigabe").
c) Ein Prozeß muß andere Prozesse ausschalten können („Abmelden").

Alle diese und eine Reihe weiterer Funktionen werden durch spezielle Programme wahrgenommen, die durch einen Rufbefehl von einem UBP oder von einem AP gestartet werden können. Nach Ausführung ihrer Aufgabe wird das Programm, das sie aufgerufen hat, fortgesetzt. Im Unterschied zu den bisher immer in die unterste Ebene eingeordneten UPs sind diese Programme jedoch Bestandteil des Steuerprogramms. Die Zugriffe zu diesen UPs werden daher als **Steuerprogrammrufe** (SPR) bezeichnet. Diese SPRs sind damit das Kommunikationsmittel zwischen den Prozessen und dem Steuerprogramm.

4.4.5. Systemprogramme (Betriebssysteme)

Die bisher gewonnenen Erkenntnisse zur Strukturierung eines Gesamtprogramms für ein Echtzeitsystem lassen sich gemäß Bild 4.17 zusammenfassen.

Es ergeben sich 4 Programmebenen. Die 1. Ebene wird von den **Unterbrechungsbehandlungsprogrammen** gebildet, die bei Auftreten von Unterbrechungsforderungen die unbedingt notwendigen bzw. zeitkritischen Aktionen ausführen und darüber hinaus Folgeprozesse anmelden können. (Das Initialisierungsprogramm kann als spezielles UBP betrachtet werden.) Die 2. Ebene bildet das **Ablaufsteuerprogramm**, bestehend aus der Kernroutine zur Ermittlung der Ablauffolge und einer Menge von Unterprogrammen, die durch Steuerprogrammrufe aktiviert werden können und in erster Linie die Prozeßstatusblöcke verwalten. Darunter schließt sich die Ebene der **Arbeitsprogramme** an, die nur vom Steuerprogramm aktiviert werden können und zusammen mit den UBP die eigentliche Arbeit leisten. Die letzte Ebene wird von den **Unterprogrammen** gebildet, die von allen darüberliegenden Ebenen Standardaufgaben übernehmen und durch Unterprogrammrufe aktiviert werden können.

Diese Programmstruktur gilt für den allgemeinen Fall. Wie wir in den im Abschnitt 4.2.2. betrachteten Beispielen gesehen haben, kann sich diese Struktur in vielen Fällen wesentlich vereinfachen. So entfällt die Ebene der UBP (außer IP), wenn das Polling-Prinzip zur Erfassung äußerer Ereignisse verwendet werden kann. Die umfangreichsten Strukturänderungen ergeben sich jedoch in den Fällen, wenn die Ablauforganisation entweder fest vereinbart oder ganz dem Zufall überlassen werden kann. Dann entfällt scheinbar die Ebene 2. In Wirklichkeit verschmelzen die Ebenen 2 und 3 zu einem für die vorliegende Echtzeitaufgabe

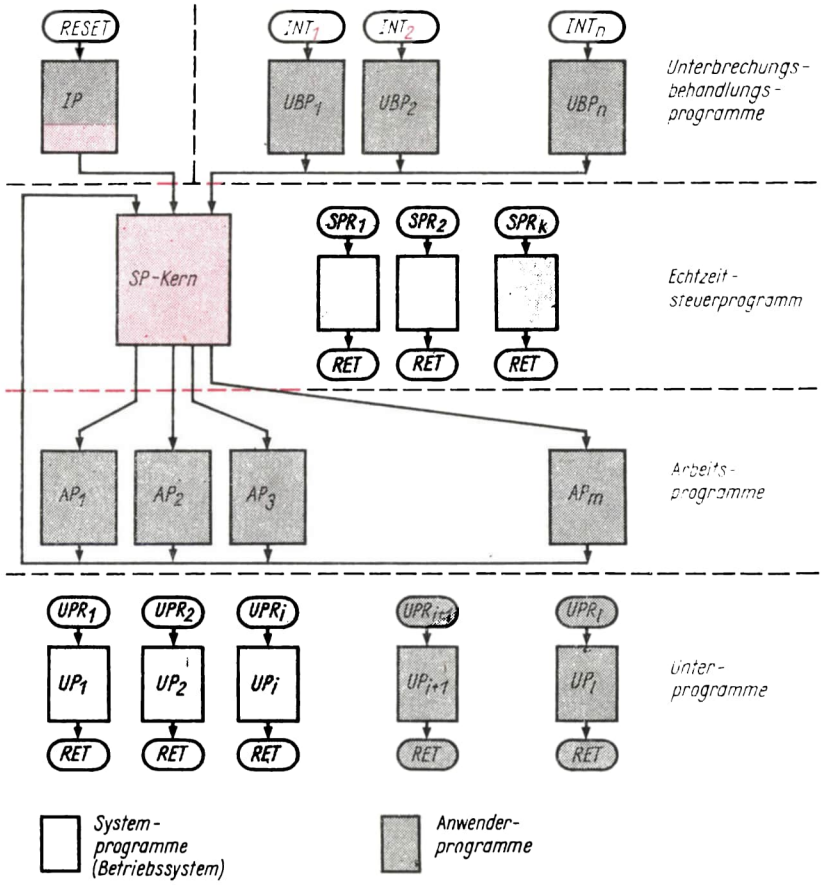


Bild 4.17. Gliederung eines Echtzeitprogramms/System- und Anwenderprogramme

individuell erstellten Hauptprogramm. Erst wenn das zu realisierende Echtzeitsystem einen hohen Komplexitätsgrad annimmt, wird eine Struktur nach Bild 4.17 erforderlich. Dann stellt sich auch die Frage, ob alle diese Programmebenen für jeden Anwendungsfall individuell programmiert werden müssen oder ob es Möglichkeiten gibt, dafür universell einsetzbare, also bereits vorgefertigte Softwaremodule bereitzustellen. Die Analyse der einzelnen Programmebenen unter diesem Gesichtspunkt zeigt, daß die APs, die UBPs und das IP weitgehend individuell programmiert werden müssen. Dagegen haben das Steuerprogramm und die UPs einen universelleren Charakter. Offensichtlich wird das bei den UPs, die immer wiederkehrende Hilfsarbeiten zu verrichten haben. Hierunter sind im wesentlichen 2 Aufgabengebiete zu verstehen:

1. die Ausführung mathematischer Funktionen (höhere, nicht in der Prozessorhardware implementierte Rechenoperationen und Funktionaltransformationen);
2. die Standardroutinen zum Betreiben bestimmter, häufig eingesetzter peripherer Geräte. Diese UPs werden auch als Gerätetreiber (driver) bezeichnet.

Ebenso ist es möglich, ein Echtzeitsteuerprogramm so zu konzipieren, daß es für eine breite Klasse von Anwendungsfällen verwendbar ist. Dabei wird entweder ein solches Programm für eine bestimmte Maximalanzahl von Prozessen ausgelegt oder aber die Möglichkeit eingeräumt, das Programm noch für den speziellen Anwendungsfall zu generieren, d. h. durch Festlegung einiger offener Parameter individuell anzupassen.

Von dieser Möglichkeit der Bereitstellung bereits vorgefertigter Softwarekomponenten entweder durch den Mikrorechnerhersteller oder durch spezielle Softwareproduzenten, wird zunehmend auch in der Mikrorechentechnik Gebrauch gemacht. Allgemein werden diese Programmkomponenten als **Systemprogramme** (im Unterschied zu den vom Nutzer geschriebenen Anwenderprogrammen) oder **Systemsoftware** bezeichnet. Auch die mit den EDV-Anlagen der 3. Generation entstandene Bezeichnung Betriebssystem wird angewendet.

Unter einem **Betriebssystem** wird ein Programmsystem (Softwarepaket) verstanden, das zusammen mit der Hardware die Basis für eine bestimmte Anwendungsklasse eines Rechners schafft, so daß vom Anwender nur noch die für den spezifischen Anwendungsfall notwendigen Programmteile ergänzt werden müssen. Insofern bildet ein Ablaufsteuerprogramm, ergänzt durch eine Menge von Standard-UPs, ein Betriebssystem, das – wird es mit einem leistungsfähigen Mikrorechner geliefert – diesen für eine breite Klasse von Echtzeitaufgaben anwendbar macht. Bezogen auf das einleitend in diesem Kapitel eingeführte Schalenmodell (Bild 4.1) kann die Wirkung eines Betriebssystems so erläutert werden, daß der Rechnerkern damit bereits zu wesentlichen Teilen „umhüllt" und folglich spezialisiert worden ist. Dem Anwendungsprogrammierer bieten sich geringere Freiheitsgrade, aber auch beträchtliche Einsparungen an Programmierzeit. Im Grunde genommen ist dies ein Modularkonzept für die Softwareseite: Für häufig auftretende Aufgaben werden vorgefertigte Softwarebausteine bereitgestellt, die oft noch in begrenztem Umfang flexibel, d. h. an die konkrete Anwendung anpaßbar sind. Selbstverständlich können solche für eine breitere Anwendungsklasse konzipierten Programmsysteme für den Spezialfall nicht optimal sein. Dem Vorteil der Vereinfachung der Programmentwicklung stehen im allgemeinen ein größerer Speicherplatzbedarf und längere Programmlaufzeiten gegenüber.

Abschließend kann damit für die Hardware- und für die Softwareseite gleichermaßen ausgesagt werden:

Modularkonzepte erleichtern durch Verkürzung der Entwicklungszeiten die Realisierung von Rechnersystemen für unterschiedlichste Applikationen. Für jene Rechneranwendungen dagegen, bei denen die Minimierung der Hardware im Vordergrund steht (z. B. zur Reduzierung der Kosten von Massenprodukten, bei begrenztem zulässigem Volumen des Rechners usw.), wird die Schaffung individueller Software meist unumgänglich, um die vorhandene Hardware maximal auszunutzen.

Ergänzende und weiterführende Literatur

[1] *Matschke, J.:* Von der einfachen Logikschaltung zum Mikrorechner. 2. Aufl. Berlin: VEB Verlag Technik 1984

[2] *Schwarz, W.; Meyer, G.; Eckardt, D.:* Mikrorechner. Wirkungsweise, Programmierung, Applikation. 3. Aufl. Berlin: VEB Verlag Technik 1984

[3] *Kieser, H.; Meder, M.:* Mikroprozessortechnik. Aufbau und Anwendung des Mikroprozessorsystems U 880. 2. Aufl. Berlin: VEB Verlag Technik 1984

[4] *Lampe, B.; Jorke, G.; Wengel, N.:* Algorithmen der Mikrorechentechnik. Maschinenprogrammierung und Interpretertechniken des U 880. Berlin: VEB Verlag Technik 1983

[5] *Claßen, L.:* Programmierung des Mikrorechnersystems U 880 — K 1520. Reihe Automatisierungstechnik. Bd. 192. 3. Aufl. Berlin: VEB Verlag Technik 1983

[6] *Oefler, U.; Claßen, L.:* Mikroprozessor-Betriebssysteme, Problemorientierte Programmierung U 880 (K 1520). Reihe Automatisierungstechnik. Bd. 201. 2. Aufl. Berlin: VEB Verlag Technik 1984

[7] *Töpfer, H.; Kriesel, W.:* Automatisierungstechnik — Gegenwart und Zukunft. Reihe Automatisierungstechnik. Bd. 200. Berlin: VEB Verlag Technik 1982

[8] *Neumann, P.:* Mikrorechner in Automatisierungsanlagen. Reihe Automatisierungstechnik. Bd. 202. Berlin: VEB Verlag Technik 1983

[9] *Werner, D.:* Programmierung von Mikrorechnern. Berlin: VEB Verlag Technik 1983

[10] *Löber, Ch.; Will, G.:* Mikrorechner in der Meßtechnik. Berlin: VEB Verlag Technik 1983

Sachwörterverzeichnis