Elektronische Bauelemente und Schaltungen der Energietechnik

von der µP-Technik bis zur Leistungselektronik

ELEKTROKOLLEG

Ingo J. Arnold
Michael Clewing
Dieter Hannemann
Walter Heinrich
Heinrich Moog
Karl-Heinz Schröder
(federführender Autor)
Rolf Stenzel

VDE-VERLAG GmbH

Inhaltsverzeichnis

Siehe auch www.DieterHannemann.de

1	**Grundlagen zur Stromleitung**	11
1.1	Allgemeines	11
1.2	Stromleitung im Vakuum	11
1.3	Stromleitung in Gasen und Dämpfen	11
1.4	Stromleitung in Metallen	12
1.5	Stromleitung in Halbleitern	12
2	**Bauelemente auf der Basis von Halbleitern**	15
2.1	PN-Übergang (Diode)	15
2.2	Si-Leistungsdiode	17
2.3	Transistor	17
2.4	Feld-Effekt-Transistor (FET)	19
2.5	Metall-Oxid-Silizium-Feld-Effekt-Transistor (MOS-FET)	20
2.6	Unijunktions-Transistor (Doppelbasis-Diode)	22
2.7	Thyristor	23
2.8	Thyristor-verwandte Bauelemente	26
2.9	Schrifttum	26
3	**Technologie integrierter Schaltkreise**	27
3.1	Planartechnik	27
3.2	Schaltkreisfamilien	29
3.3	Anwendungshinweise	30
4	**Allgemeine Grundlagen zur Programmierung von Mikroprozessoren**	35
4.1	Mensch und Computer	35
4.2	Zahlensysteme	36
4.3	Codierung	38
4.4	Programmierung	43
5	**Grundeinheiten eines Mikrocomputers**	47
5.1	Der Mikrocomputer	47
5.2	Die Speicher	48
5.3	Der Mikroprozessor	49
6	**Programmerstellung und Abarbeitung**	53
6.1	Die Befehle	53
6.2	Unterprogramme	63
6.3	Programm-Entwicklungssysteme	66

7	**Elektronischer Aufbau eines Mikrocomputers**	71
7.1	Der Mikroprozessor	71
7.2	Die Speicher	75
7.2.1	Schreib-/Lesespeicher (RAM)	78
7.2.2	Festwertspeicher (ROM)	79
7.3	Ein-/Ausgabe-Einheiten	80
8	**Mikrocomputer-Anwendung**	85
8.1	Mikrocomputer-Platine	86
8.2	Eingabetastatur	89
8.3	Anzeige-Einheit	90
8.4	Motorsteuerung	91
8.5	Schrifttum	94
9	**Der Transistor als Leistungsschalter**	95
9.1	Bipolar- und Feldeffekt-Transistor	96
9.2	Transistor-Parameter	97
9.3	Bipolar-Transistor	99
9.3.1	Sperrzustand	99
9.3.2	Schaltverhalten	102
9.3.3	Ansteuerung und Grundschaltungen von Schaltstufen	105
9.3.4	Darlington	108
9.4	MOSFET	110
9.4.1	Sperr- und Leitzustand	110
9.4.2	Ansteuerung, Schaltverhalten	113
9.5	Gehäuseformen, Transistormontage	116
9.6	Ausblick	118
9.7	Schrifttum	120
10	**Spezielle Thyristorarten der Leistungselektronik**	121
10.1	Einführung	121
10.2	Frequenz-Thyristoren	122
10.3	Asymmetrische Thyristoren	124
10.4	Rückwärtsleitende Thyristoren	124
10.5	„Gate assisted turn-off" — Thyristoren	126
10.6	Abschaltbare Thyristoren	126
10.7	Schrifttum	129
11	**Schaltungen der Leistungselektronik**	131
11.1	Einführung	131
11.2	Schaltungen für Gleich- und Wechselrichter	131
11.2.1	Einphasen-Schaltung	132
11.2.2	Zweiphasen-Mittelpunkt-Schaltung	134
11.2.3	Zweiphasen-Brücken-Schaltungen	135
11.2.4	Dreiphasen-Mittelpunkt-Schaltung	138

11.2.5	Dreiphasen-Brücken-Schaltung	141
11.2.6	Sechsphasige Mittelpunktschaltung	143
11.3	Antiparallelschaltungen	145
11.3.1	Kreisstromfreie Schaltung	145
11.3.2	Kreisstromführende Schaltung	146
11.4	Zusammenfassung	148
12	**Einsatzbeispiele von netzgeführten Stromrichtern in der Industrie**	151
12.1	Ausführung, Arten, Auswahl	151
12.2	Versorgung statischer Verbraucher	154
12.3	Anwendungen von Gleichstromantrieben	159
12.4	Schrifttum	174
13	**Servoantriebe**	175
13.1	Anforderungen an Servoantriebe	175
13.2	Komponenten des Antriebs	176
13.3	Drehzahlregelgeräte in analoger und getakteter Technik	179
13.4	Grundschaltungen getakteter Leistungsverstärker	183
13.5	Regler, Schutz- und Überwachungsfunktionen	188
13.6	Regelkreise für Drehmoment, Drehzahl und Position	190
13.7	Positionsgeregelte Vorschubachse	192
13.8	Schrifttum	193
14	**Antriebsregelungen mit dem Gleichstrom-Nebenschlußmotor und Stromrichtergeräten**	195
14.1	Gleichstrom-Nebenschlußmotor	195
14.1.1	Aufbau und Funktion	195
14.1.2	Vollgeblechte Ausführung	196
14.1.3	Anschluß-Kennbuchstaben	198
14.1.4	Kennlinien des Gleichstrom-Nebenschlußmotors	198
14.1.5	Beziehungen für Gleichstrom-Nebenschlußmotor und Stromrichtergerät	200
14.1.6	Nennspanung	201
14.2	Arbeitsmaschinen	202
14.2.1	Allgemeine Beziehungen der Antriebstechnik	202
14.2.2	Drehmoment- und Leistungskennlinien	204
14.3	Betriebsarten des thyristorgespeisten Gleichstrommotors	205
14.3.1	Quadrantenbetrieb I bis IV	205
14.4	Betriebsarten der Stromrichter	206
14.4.1	Betriebsarten eines Doppel-Stromrichters im Vierquadrantbetrieb	206
14.4.2	Arbeitsweise eines Zweifach-Stromrichters	207
14.4.3	Steuerkennlinie und resultierende Gleichspannung bei der Drehstrombrückenschaltung	208

14.5	Allgemeine Grundlagen zur Regelungstechnik	209
14.5.1	Regelung	209
14.5.1.1	Regelkreis	209
14.5.1.2	Vergleichsstelle	210
14.5.2	Operationsverstärker	211
14.5.2.1	Schaltzeichen	211
14.5.2.2	Idealer Operationsverstärker	211
14.5.2.3	Realer Operationsverstärker	212
14.5.2.4	Eingangs-Ausgangs-Kennlinie	213
14.5.3	Grundschaltungen	213
14.5.3.1	Invertierender Verstärker	213
14.5.3.2	Nicht invertierender Verstärker	215
14.5.3.3	Integratorschaltung	215
14.5.3.4	PI-Regler	215
14.5.3.5	PID-Regler	215
14.6	Regelungsarten	216
14.6.1	Drehzahl-Regelung im Ankerbereich für Einzelantriebe	216
14.6.1.1	Einquadrant- oder Geradeausbetrieb: I oder III	216
14.6.1.1.1	Drehzahlführungsgröße	217
14.6.1.1.2	Istwert-Erfassung	217
14.6.1.1.3	Überlagerter Drehzahlregelkreis	217
14.6.1.1.4	Dynamische Strombegrenzung	217
14.6.1.1.5	Unterlagerter Stromregelkreis	217
14.6.1.1.6	Steuerteil des Stromrichtergerätes	218
14.6.1.1.7	Tachomaschinen- oder Ankerspannungsregelung	218
14.6.1.1.8	Soll-/Istwert-Vergleich	219
14.6.1.1.9	Istwert-Anpassung	219
14.6.1.1.10	Sollwert-Integrator	219
14.6.1.1.11	Einquadrantbetrieb I oder III	220
14.6.1.2	Einquadrant- oder Geradeausbetrieb: I und III	220
14.6.1.3	Vierquadrant- oder Umkehrbetrieb: I bis IV	222
14.6.1.3.1	Elektronische Umkehrstufe	223
14.6.1.3.2	Drehrichtungswechsel	223
14.6.1.3.3	Kreisstromfreie Umschaltung	223
14.6.1.3.4	Wechselrichterbetrieb	223
14.6.1.3.5	Zugeordnete Strombegrenzung	224
14.6.2	Überlaufregelung im Anker- und Feldbereich, wahlweise Einquadrant- oder Vierquadrantbetrieb	224
14.6.2.1	Ablösebereich	224
14.6.2.2	Istwert-Erfassung	225
14.6.2.3	Drehzahl-Sollwert	225
14.6.2.4	Vierquadrantstromrichter	225
14.6.3	Mehrmotorenregelung mit relativem Drehzahlgleichlauf	225
14.6.4	Verstärker-Schaltung	227

14.6.5	Mehrmotorenregelung mit synchronem Drehzahlgleichlauf	227
14.6.6	Mehrmotorenregelung mit reib- oder kraftschlüssiger Gleichlastregelung und drehzahlgeführter Gleichlastkorrektur	229
15	**Statische Wechselrichter für die unterbrechungsfreie Stromversorgung**	231
15.1	Unterbrechungsfreie Stromversorgung (USV)	231
15.1.1	Einzel-USV mit Umgehung (elektronische Umschalteinrichtung)	232
15.1.2	Parallelredundante USV mit Umgehung	235
15.1.3	Besondere Ersatzstromversorgung nach VDE 0107	237
15.2	Wechselrichter mit Thyristoren	239
15.3	Wechselrichter mit Leistungstransistoren	242
15.4	Wesentliche Wechselrichter-Eigenschaften	243
15.4.1	Wirkungsgrad	244
15.4.2	Kurzschlußverhalten	245
15.4.3	Dynamisches Verhalten (Spannungsabweichungen)	247
16	**Kommutierung bei selbstgeführten Stromrichtern**	249
16.1	Definition	249
16.2	Kommutierung bei Stromrichtern mit Thyristoren	249
16.3	Stromrichterschaltung nach McMurray	250
16.3.1	Kommutierungsvorgang	252
16.3.2	Dimensionierung des Kommutierungskreises	256
16.4	Schrifttum zu Kapitel 15 und 16	257
17	**Anwendungen umrichtergespeister Antriebe**	259
17.1	Umrichterantriebe: wirtschaftlich, konkurrenzfähig?	259
17.2	Drehzahlverstellung bei Drehstrommotoren	261
17.3	Umrichter ohne Zwischenkreis	262
17.4	Umrichter mit Stromzwischenkreis (*I*-Umrichter)	264
17.4.1	Stromrichtergespeiste Synchronmaschinen	264
17.4.2	Untersynchrone Stromrichterkaskade	267
17.4.3	Umrichter mit eingeprägtem Strom für Käfigläufermotoren	268
17.5	Umrichter mit Spannungszwischenkreis (*U*-Umrichter)	270
17.5.1	*U*-Umrichter mit variabler Zwischenkreisspannung	271
17.5.2	*U*-Umrichter mit konstanter Zwischenkreisspannung (Pulsumrichter)	272
17.6	Zusammenfassende Anwendungskriterien	274
17.7	Schrifttum	276
18	**Stichwortverzeichnis**	277

2.8 Thyristor-verwandte Bauelemente

a) *Ausschaltbarer Thyristor (GTO = Gate turn-off)*
Der ausschaltbare Thyristor kann durch einen positiven Impuls gezündet und durch einen negativen wieder gelöscht werden. Bei der Löschung wird die Eigenschaft der Thyristoren ausgenutzt, das Haltestrom-Niveau durch einen negativen Steuerstrom kurzzeitig stark anzuheben, so daß es über dem Niveau des in dem Augenblick fließenden Durchlaßstromes liegt und der Thyristor somit wieder in den Blockierzustand zurückfällt.
Die Steuer- und Katodenelektrode sind dabei in besonderer Weise ineinander verschachtelt. Ausschaltbare Thyristoren für höhere Durchlaßströme befinden sich z. Z. in der Entwicklung. Die Grenze der Ausschaltbarkeit liegt heute bei einigen 100 A.

b) *Diac (Bidirektionale Thyristor-Diode)*
(Diac: Di = Diode, ac = Wechselstrom)
Dieser Thyristor ohne Steuerelektrode zeigt in beiden Stromrichtungen gleiches Verhalten. Zunächst sperrt er in beiden Richtungen, zündet dann aber nach Überschreiten der Null-Kipp-Spannung durch.
Verwendung: Steuerschaltungen für Thyristoren und Triacs (Impulserzeugung), Schwellwert-Schalter.

c) *Triac (Bidirektionale Thyristor-Triode)*
(Triac: Tri = Triode, ac = Wechselstrom)
Mit Hilfe von Stromimpulsen über die Steuerelektrode kann dieser Thyristor in beiden Richtungen gezündet werden. Er verhält sich somit wie zwei antiparallele Thyristoren und ermöglicht die Steuerung eines Wechselstromes. Es wird jedoch nur ein Kühlkörper und ein vergleichsweise einfaches Steuergerät benötigt.
Verwendung: Helligkeitssteuerungen und Drehzahlsteuerungen kleiner Wechselstrom-Motoren.

2.9 Schrifttum

[1] Beuth, K.: Bauelemente der Elektronik. Würzburg: Vogel-Verlag, 1978
[2] BBC: Silizium-Stromrichter-Handbuch. Baden: BBC-AG, 1971
[3] Heumann, K.: Grundlagen der Leistungselektronik. Stuttgart: Teubner-Verlag, 1974
[4] Tholl, H.: Bauelemente der Halbleiterelektronik. Stuttgart: Teubner-Verlag, 1972
[5] Jungclaus, H. J.; Neukamm, G.: Bauelemente und Grundschaltungen. Stuttgart: Verlag Berliner Union; Stuttgart, Berlin, Köln, Mainz: Verlag W. Kohlhammer, 1980

3 Technologie integrierter Schaltkreise

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Nachdem es 1963 gelungen war, mehrere Transistoren auf einem Silizium-Plättchen (Chip) zu einer Schaltung zu integrieren, begann eine stürmische Entwicklung auf diesem Gebiet.
Heute ist man in der Lage, bis zu 100 000 Transistorfunktionen (Transistoren, Dioden, Widerstände und Kondensatoren) auf einem Chip zu integrieren. Hierzu wurden verschiedene Herstellungsstechnologien entwickelt, die zu unterschiedlichen Schaltkreisfamilien führten.

3.1 Planartechnik

Während einzelne Halbleiterbauelemente — und dies insbesondere für die Leistungselektronik — auch heute noch meistens in der in den Kapiteln 1 und 2 beschriebenen Technik ausgeführt werden, kommt bei der Herstellung integrierter Schaltkreise die sogenannte Planartechnik zum Einsatz.
Durch eine Reihe von Herstellungsschritten lassen sich die bekannten Bauelemente vom Transistor bis zum Kondensator nebeneinander auf einem Chip erzeugen und durch Aluminiumleiterbahnen untereinander verbinden (**Bild 3.1**).

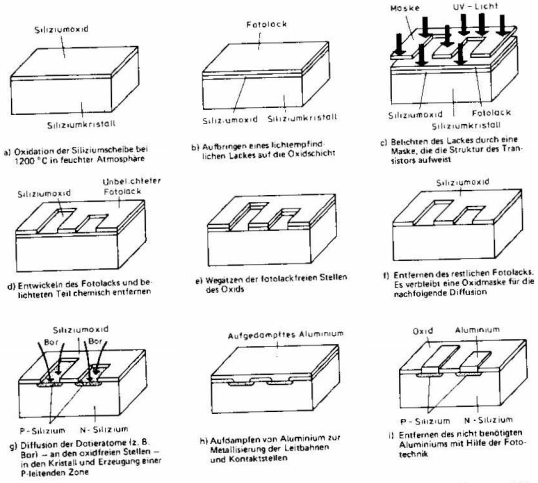

Bild 3.1. Herstellungsschritte zum Aufbau eines Einzelbauelementes in der Silizium-Planartechnik. Durch Oxidmaskierung werden bestimmte kleine Bereiche des Halbleiterkristalls zu N- oder P-leitenden Zonen dotiert.
N-leitend = Elektronenleitung, P-leitend = Löcherleitung.

Mittels der Planartechnik können sowohl Bauelemente in der Bipolar-Technologie als auch in der MOS-Technologie (Unipolar) erzeugt werden (**Bild 3.2**).
Die MOS-Technik (Metal-Oxide-Semiconductor) wird hauptsächlich bei der Herstellung großintegrierter digitaler Schaltungen angewandt. Es werden hierbei sogenannte MOSFET-Transistoren erzeugt. (FET = Feld-Effekt-Transistor). Die bipolare Technik hingegen findet überwiegend Anwendung bei der Herstellung analoger und besonders schneller digitaler Schaltkreise.
Je nach der Menge der auf einem Chip integrierten Bauelemente teilt man die Bausteine heute in folgende Kategorien ein:

SSI kleiner Integrationsgrad (small scale integration),
MSI mittlerer Integrationsgrad (medium scale integration),
LSI großer Integrationsgrad (large scale integration),
VLSI sehr großer Integrationsgrad (very large scale integration) mit etwa 100 000 Bauelementen auf einem Chip.

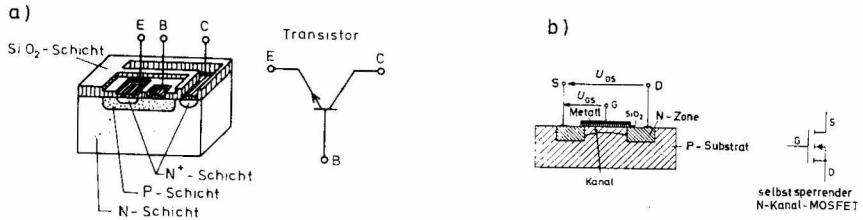

Bild 3.2. Beispiele für den Aufbau von Transistoren in der Planartechnik
a) Bipolarer Transistor
b) Unipolarer (MOS) Feldeffekttransistor (FET), selbstsperrender N-Kanal-MOS-FET (NMOS-Technik)
M = Metall Elektrode (Gate), O = Oxid (SiO_2), S = Semiconductor (Halbleiter Silizium).

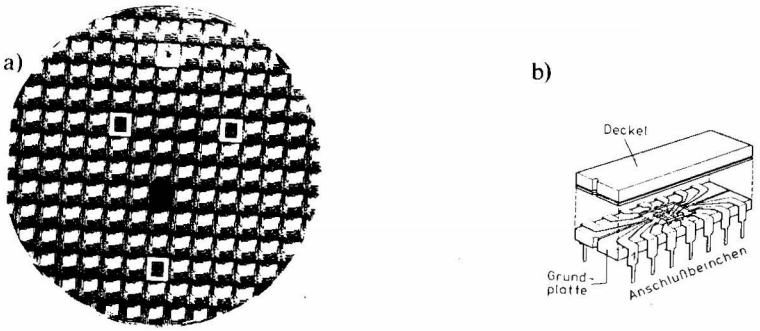

Bild 3.3.
a) Siliziumscheibe mit vielen integrierten Schaltkreisen (wafer)
b) Das aus dem Wafer herausgebrochene Si-Plättchen wird auf eine Grundplatte geklebt, mit Golddrähten an die Beinchen angeschlossen und dann mit einem Deckel versehen.

Bei der Herstellung integrierter Schaltkreise wird als Ausgangsmaterial höchstreines Silizium, in Form dünner Scheiben mit etwa 7,5 cm Durchmesser (wafer), benutzt. Auf einem solchen Wafer werden dann nebeneinander so viele der zu erstellenden Schaltkreise erzeugt, wie auf ihm Platz haben (z. B. 100—150 Stück). Nach den zur Herstellung in der Planartechnik erforderlichen — bis zu sechs — Produktionsschritten werden die einzelnen Chips — durch Anritzen des Wafers mit einem Diamanten — herausgebrochen und dann in ein Gehäuse gebracht (**Bild 3.3**).

3.2 Schaltkreisfamilien

Bipolar-Technik

Die am weitesten verbreitete digitale Schaltkreisfamilie ist die sogenannte TTL-Familie (Transistor-Transistor-Logik).

Baustein-Familie	TTL	-L	-S	-LS	ECL	NMOS	CMOS
Leistungsaufnahme je Gatter typ./mW	50	1	50	2	25	1	0,001–6
Maximale Frequenz f/MHz	20–50	2–5	80–125	50	125–....	10	5

Bild 3.4. Vergleich verschiedener Bausteinfamilien
L = Low-Power, S = Schottky-Technik, LS = Low-Power-Schottky, ECL = Emitter-Coupled-Logic, NMOS = N-Kanal-MOS-Technik, CMOS = Komplementäre MOS-Technik. Die Leistungsaufnahme der CMOS-Gatter hängt stark von der Arbeitsfrequenz ab. Beispiel für eine Bausteinbezeichnung: 74 LS 08 → TTL-Baustein Low-Power-Schottky.

Sie bietet heute das umfassendste und preiswerteste Standardprogramm digitaler Bausteine. Die Versorgungsspannung beträgt + 5 V. Bei der Herstellung wird die bipolare Technik angewandt (Bild 3.2 a). Aufbauend auf dieser Technik gibt es jedoch eine Reihe von Herstellungsvarianten, die den Bausteinen unterschiedliche Eigenschaften verleihen (**Bild 3.4**).

MOS-Technik

Die MOS-Technik hat im wesentlichen drei verschiedene Schaltkreisfamilien hervorgebracht, und zwar die N-Kanal- (Bild 3.2 b), die P-Kanal- und die komplementäre (C)-MOS-Familie.
Während die ersten Mikroprozessoren in PMOS-Technik aufgebaut worden sind (z. B. 1971 der MP 8008), wird heute die NMOS-Technik bevorzugt, da die Bausteine mit + 5 V betreibbar und daher TTL-kompatibel sind.
Bei der CMOS-Technik werden je ein P-Kanal- und ein N-Kanal-Transistor kombiniert. Hierdurch läßt sich eine Reduzierung der Leistungsaufnahme und ein großer Versorgungsspannungsbereich (z. B. + 3 V bis + 18 V erreichen (**Bild 3.5**). Außerdem erhöht sich die Störsicherheit. Auch Mikroprozessoren werden zunehmend in CMOS-Technik hergestellt.

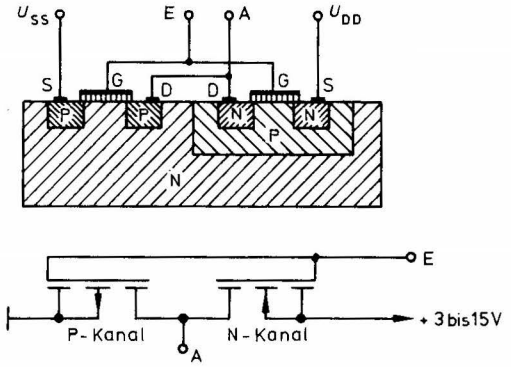

Bild 3.5. Aufbau und Schaltsymbol eines CMOS-Inverters
E = Eingangssignal, A = Ausgangssignal

Handhabung von MOS-Bausteinen
Infolge der extrem hohen Eingangsimpedanz aller Arten von MOS-Bausteinen sind besondere Vorkehrungen für den Umgang mit ihnen erforderlich.
Bei einer Eingangsimpedanz von 10 000 000 000 000 Ohm kann schnell die Durchbruchspannungsgrenze des Gate-Oxids (etwa 100 V) überschritten und damit der Baustein zerstört oder nachhaltig verändert werden. Solche Spannungen, und noch viel höhere, entstehen leicht durch statische Aufladungen (z. B. Nylonhemd: 2 kV). Deshalb sollten stets folgende Vorsichtsmaßnahmen beachtet werden:
— Bausteine nur in leitenden Behältern oder auf leitendem Material aufbewahren.
— Boden- und Tischbelag am Arbeitsplatz sollten keine Reibungselektrizität hervorrufen.
— Signale dürfen die Bausteine nur erreichen, wenn die Vorsorgungsspannung anliegt.
— Alle nicht benutzten Eingänge sollten entweder an die Versorgungsspannung oder an Masse angeschlossen werden.

3.3 Anwendungshinweise

Beim Einsatz integrierter Schaltungen sind einige Besonderheiten zu beachten.

Logische Funktionen
In der Digitaltechnik interessieren die Spannungen, welche an den Ein- und Ausgängen eines Bausteines anliegen, im allgemeinen solange nicht, wie sie sich innerhalb bestimmter Grenzwerte bewegen. Hier sind nur die den Spannungsbereichen zugeordneten, logischen Werte L oder H wichtig. Wenn die Zuordnung so ausfällt, daß dem L der niedrige und dem H der hohe Spannungsbereich zugeordnet wird, nennt man dies „positive Logik". Der umgekehrte Fall heißt dann „negative Logik" **(Bild 3.6)**.

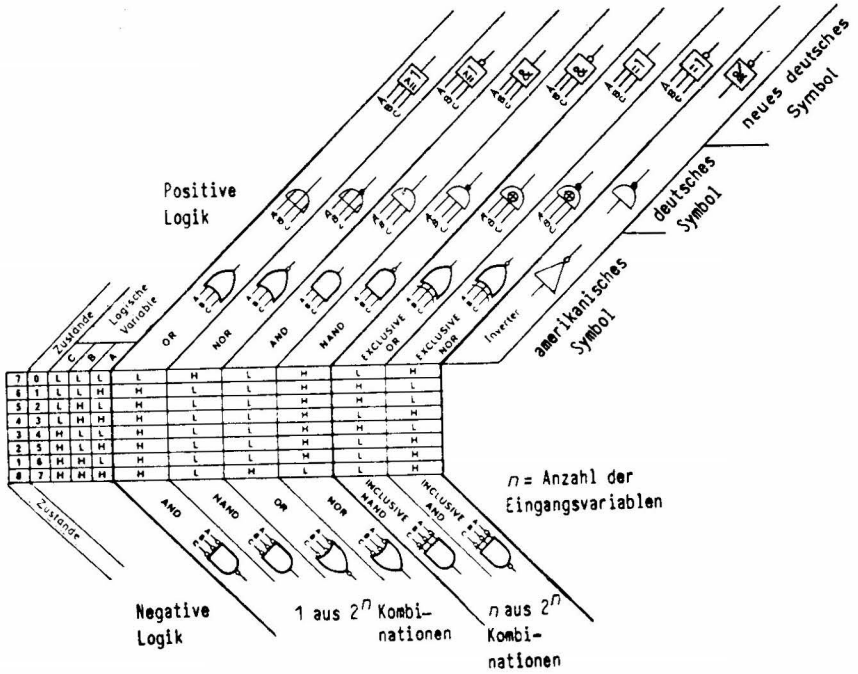

Bild 3.6. Fundamentale Logik-Funktionen

Die logischen Funktionen lassen sich durch entsprechende integrierte Schaltungen realisieren. Solche Schaltungen haben einen oder mehrere Eingänge und einen Ausgang. Sie werden meistens als „Gatter" bezeichnet. Auch in Mikrocomputern findet man häufig noch Bausteine, die einzelne Gatter enthalten.

Ausgänge
Standard-TTL-Gatter enthalten eine Gegentakt-Endstufe als Ausgang. Dies ist eine der gebräuchlichsten Ausgangsstufen. Daneben ist jedoch auch noch der sogenannte „offener Kollektor-Ausgang" (open collector) anzutreffen **(Bild 3.7)**. Wie schon der

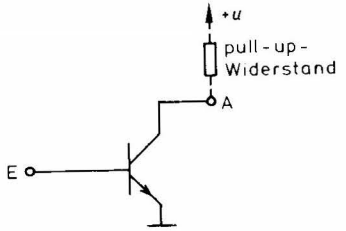

Bild 3.7. Ausgangsschaltung mit offenem Kollektor (open collector).

Name sagt, besteht diese Endstufe lediglich aus einem NPN-Transistor, dessen Emitter an Masse liegt.
Vorteile dieser Schaltung sind, daß sich diese Ausgänge parallel schalten lassen (wired AND bzw. wired-OR-Verknüpfung) und daß über einen sogenannten pull-up Widerstand, der auch an eine höhere Spannung gelegt werden kann, eine Umwandlung der den logischen Zuständen zugeordneten Spannungen möglich ist.

Eine solche Pegelwandlung kann beispielweise erforderlich sein, um einen TTL-Ausgang an einen mit höherer Spannung betriebenen CMOS-Baustein anzupassen.

Insbesondere die Realisierung von Bus-Systemen in Mikrocomputern, bei denen viele Ausgänge parallel geschaltet sind, machte es jedoch erforderlich, einen Ausgang zu schaffen, der sich „abschalten" läßt. Ein solcher Ausgang kann neben den beiden logischen Zuständen L und H auch einen sogenannten hochohmigen Zustand annehmen.
Während bei einer herkömmlichen Gegentakt-Endstufe immer nur ein Transistor gesperrt ist, besteht bei der Endstufe in **Bild 3.8** auch die Möglichkeit, beide Transistoren zu sperren. Eine solche Endstufe wird Tristate- (oder auch three state-) Ausgang genannt.
Die Sperrung der Ausgangstransistoren T1 und T2 erfogt über die beiden Transistoren T3 und T4. Der Eingang, welcher z. B. die Ausgänge eines Speichers in den inaktiven Zustand bringt, wird häufig mit CS (chip select = Bausteinauswahl) bezeichnet.

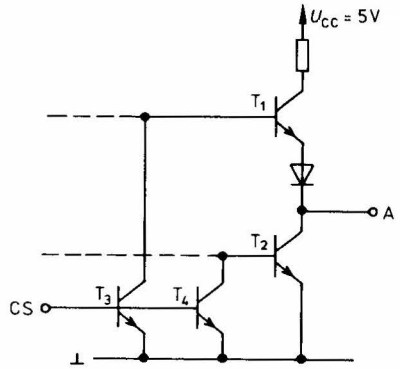

Bild 3.8. Tristate-Ausgang eines Speichers

Im hochohmigen Zustand nimmt dann der Ausgang den logischen Pegel an, der ihm von außen aufgeprägt wird (indifferenter Zustand, floating state).
Häufig sind in einer Schaltung an einem Ausgang mehrere Eingänge anderer Bausteine angeschlossen. Hierbei muß darauf geachtet werden, daß die Ausgangsbelastbarkeit (fan-out) nicht überschritten wird. Ein Fan-Out von 10 gibt an, daß an diesen Ausgang 10 Eingänge der gleichen Bausteinfamilie anschließbar sind.

Beispiel:

Bausteinfamilie	TTL	NMOS	CMOS
Typ. Fan-Out	10	20	50

4 Allgemeine Grundlagen zur Programmierung von Mikroprozessoren

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Die große Flexibilität beim Einsatz von Mikroprozessoren (MP) beruht vor allem auf ihrer Programmierbarkeit. Allein durch ein unterschiedliches Programm (software) kann ein und derselbe Mikrocomputer (hardware) die unterschiedlichsten Aufgaben wahrnehmen. Diese Möglichkeit der Programmierung — d. h. die Lösung von Problemen mittels spezieller Programme — stellt jedoch an den Entwickler neue Anforderungen.
Die primäre Frage ist, aus welchen Teilen besteht ein solcher Mikrocomputer (MC), und wie kann man mit ihm in Kontakt treten?
Es soll versucht werden, auf diese Frage in den folgenden vier Abschnitten eine kurze, einführende Antwort zu geben.

4.1 Mensch und Computer

Am Beispiel einer Waage läßt sich zeigen, wie ein Computer (Rechner) die Aufgaben eines Menschen übernehmen kann und wie die einzelnen Teile benannt werden.

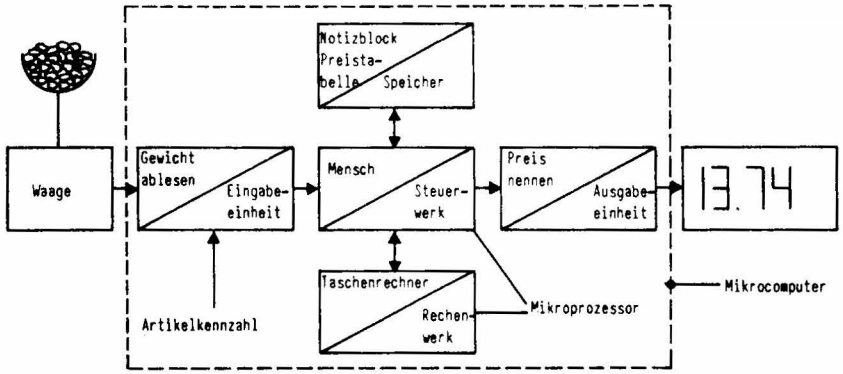

Bild 4.1. Mensch und Computer im Vergleich; am Beispiel des Verkaufs von Waren mit Hilfe einer Waage

In den Kästchen von **Bild 4.1** sind links oben jeweils die vom Menschen auszuführenden Arbeiten und rechts unten die entsprechenden Computereinheiten benannt.
Über Ein-/Ausgabe-Schnittstellen steht der Computer mit der Umwelt in Verbindung.
In welcher Form kann diese Verbindung hergestellt werden? Vom physikalisch-

technischen Standpunkt aus kommen hierzu nur elektrische Signale in Frage, doch darüber mehr im Abschnitt 4.3. Zunächst wollen wir uns der Frage vom grundsätzlichen, mathematischen Standpunkt aus nähern.
Da der Computer nur zwei verschiedene Buchstaben kennt und versteht (Schalter Ein oder Aus, Spannung Null oder positiv etc., 0 oder 1, L oder H), müssen wir uns fragen, wie sich mittels dieses Alphabetes verschiedene Wörter bilden lassen. Diese verschiedenen Wörter können dann, repräsentiert durch elektrische Spannungen, dem Computer übermittelt werden. Bei Wahl der richtigen Wörter faßt der Computer sie als Befehle auf und „gehorcht".
Die Begriffe „Buchstabe" und „Alphabet" sollen hier in einem allgemeineren Sinn verstanden werden; sie lassen sich auch durch die Begriffe „Zeichen" und „Zeichenvorrat" ersetzen.

4.2 Zahlensysteme

Eine Zahl läßt sich in verschiedenen Systemen darstellen. Charakteristisch für das Zahlensystem sind die Art und Anzahl der Zeichen (Buchstaben). Wobei die Menge der verschiedenen Zeichen den Zeichenvorrat (numerisches Alphabet) bestimmt. Gleichzeitig wird durch die Anzahl der Zeichen die Basis des Zahlensystems festgelegt. Am vertrautesten ist uns das Dezimalsystem.

Dezimalsystem
Der Zeichenvorrat umfaßt die Ziffern 0 bis 9. Die Basis B dieses Zahlensystems ist damit gleich 10.
Der Wert einer Zahl wird nach Potenzen von 10 geordnet. Auf diese Weise erhält man die sogenannte Potenzschreibweise einer Zahl. Vertrauter, weil einfacher, ist uns jedoch die sogenannte Stellenschreibweise, welche sich unmittelbar aus der Potenzschreibweise herleiten läßt.
Bei der Stellenschreibweise werden nur noch die Faktoren vor den Zehnerpotenzen aneinandergereiht. Beim Lesen einer solchen Zahl geht man dann davon aus, daß eine Ziffer, die weiter links steht, dem Faktor zur nächst höheren Zehnerpotenz entspricht.

Beispiele:
a) ganze Zahl

Stellenschreibweise	Potenzschreibweise
7349	$= 7 \cdot 1000 + 3 \cdot 100 + 4 \cdot 10 + 9 \cdot 1$
	$= 7 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 9 \cdot 10^0$

b) Dezimalbruch

$$2{,}37 \quad = 2 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2}$$

Auf gleiche Weise lassen sich auch andere Zahlensysteme mit anderem Zeichenvorrat (Basis) aufbauen. Hier soll uns vor allem das für die Computertechnik wichtige Dualsystem interessieren.

Dualystem
Das Dualsystem bildet einen Sonderfall im Bereich der Binärsysteme (binär = zweiwertig).
Unter einem Binärsystem versteht man irgendein Zahlensystem, das mit nur zwei Elementen auskommt (z. B. Tetradendarstellung in Abschnitt 4.3), bei dem aber die Zahl nicht unbedingt durchgängig nach Zweierpotenzen geordnet ist. Wenn eine Ordnung nach Potenzen von zwei vorgenommen wird, erhalten wir das duale Zahlensystem.

Beispiel:

$$\text{HLLH, HLH} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$
$$= 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625$$

Eine Dualzahl ist im Schnitt 3,3mal länger als die entsprechende Dezimalzahl.
Wegen der Länge der Dualzahlen gestaltet sich der Umgang mit ihnen besonders aufwendig. Aus diesem Grunde hat sich ein weiteres Zahlensystem bewährt, welches die abkürzende Darstellung einer vierstelligen Dualzahl (Tetrade) durch ein Zeichen gestattet.

Dezimal	Dual	Hexadezimal
$10^2\,10^1\,10^0$	$2^7\,2^6\,2^5\,2^4\;\;2^3\,2^2\,2^1\,2^0$	$16^1\,16^0$
0	L	0
1	H	1
2	H L	2
3	H H	3
4	H L L	4
5	H L H	5
6	H H L	6
7	H H H	7
8	H L L L	8
9	H L L H	9
1 0	H L H L	A
1 1	H L H H	B
1 2	H H L L	C
1 3	H H L H	D
1 4	H H H L	E
1 5	H H H H	F
1 6	H L L L L	1 0
1 7	H L L L H	1 1
⋮		
3 1	H H H H H	1 F
3 2	H L L L L L	2 0
⋮		
6 3	H H H H H H	3 F
6 4	H L L L L L L	4 0
⋮		
1 2 7	H H H H H H H	7 F
1 2 8	H L L L L L L L	8 0
⋮		
2 5 5	H H H H H H H H	F F

Bild 4.2. Vergleich der drei wichtigsten Zahlensysteme

Hexadezimalsystem
Eine vierstellige Dualzahl kann insgesamt $2^4 = 16$ verschiedeneWerte annehmen. Deshalb muß das zur Abkürzung einer Tetrade zu benutzende Zahlensystem die Basis 16 haben. Als Zeichenvorrat wird das numerische Alphabet (0 . . . 9) und die ersten sechs Zeichen des Buchstabenalphabetes (A . . . F) verwandt. Im folgenden werden diese Zeichen häufig HEX-Zeichen genannt.

Zeichenvorrat b_i: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Basis $B = 16$
Beispiel: $379H = 3 \cdot 16^2 + 7 \cdot 16^1 + 9 \cdot 16^0 = 889D$
$BAD_H = 11 \cdot 16^2 + 10 \cdot 16^1 + 13 \cdot 16^0 = 2989_D$

Zur Kennzeichnung, ob es sich um eine Dezimalzahl oder eine Hexadezimalzahl handelt, wird hinter die Zahl als Index, oder auch in gleicher Höhe, ein D oder H geschrieben.

Addition im Dualsystem
Das Rechnen mit Dualzahlen ist besonders einfach, da es ja nur zwei Zeichen gibt. Zum besseren Verständnis einiger späterer Besonderheiten soll hier kurz auf das Wesentliche eingegangen werden. Analog der uns bekannten Regeln im Dezimalsystem erhält man für die Addition von Dualzahlen:

dezimal		dual
0 + 0 = 0		L + L = L
0 + 1 = 1		L + H = H
1 + 0 = 1		H + L = H
1 + 1 = 2		H + H = HL

Beispiel:

70		HLLLHHL
+ 34		+ HLLLHL
1	Übertrag	HH
= 104		= HHLHLLL

Das Addieren von Dualzahlen ist eine der Basisfähigkeiten eines jeden Computers. Alle anderen Grundrechenarten können auf die, evtl. mehrfach durchzuführende, Addition zurückgeführt werden.

4.3 Codierung

Das Wort Codierung bedeutet Verschlüsselung, und unter einem Code versteht man eine Umwandlungsvorschrift für eine Information.
Zur Kontaktaufnahme mit einem Mikroprozessor bedient man sich des Binäralphabetes, d. h., mittels der beiden Binärzeichen L und H können Wörter gebildet werden, die ein Mikroprozessor „versteht". In gleicher Weise gibt auch der Mikroprozessor Informationen an seine Umwelt hinaus.

Codierung der Binärzeichen
Die technische Realisierung der beiden Binärzeichen L und H kann auf verschiedene Weise erfolgen.

Beispiele:

L entspricht:	Aus offen	kein Strom	keine Spannung
H entspricht:	Ein geschlossen	Strom	Spannung

Am häufigsten tritt heute die letzte Form auf, d. h. eine Spannungscodierung der Binärzeichen.

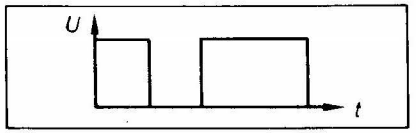

Bild 4.3. Ideales Spannungs-Zeit-Diagramm der Zahl HLHH

In **Bild 4.3** ist ein Wort (Zahl) dargestellt, welches aus vier Buchstaben (Binärzeichen) besteht. Dem L entspricht der Zustand „keine Spannung" und dem H der Zustand „positive Spannung". Diese unterschiedlichen Spannungszustände, die in einem bestimmten zeitlichen Rhythmus aufeinanderfolgen, können z. B. an dem Anschlußstift eines Mikroprozessors auftreten und für ihn eine Information bedeuten.
Wenn die einzelnen Zeichen zeitlich hintereinander erscheinen, so wie in Bild 4.3, nennt man das einen „seriellen Informationstransport" (oder serielle Datenübertragung). Dies ist vergleichbar mit einem Menschen, welcher ein Wort buchstabiert. Wenn jedoch der Mensch ein Wort liest, so nimmt er es im allgemeinen als Gesamtheit in sich auf. Dieser Vorgang wird bei einem Computer „paralleler Informationstransport" genannt. Angewandt auf einen Mikroprozessor bedeutet dies, daß z. B. an vier Anschlußstiften die vier Binärzeichen des Bildes 4.3 gleichzeitig für eine bestimmte Dauer anliegen.
Für jede einzelne Stelle in einer Dualzahl, d. h. für jede Ziffer, wurde der Begriff „bit" (binary digit) eingeführt. Ein Bit steht also für eine Alternativentscheidung L oder H.

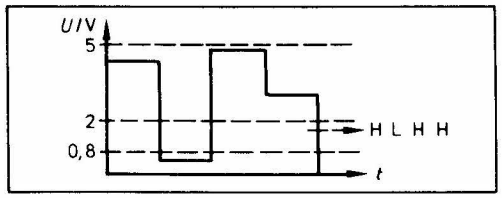

Bild 4.4. Reales Spannungs-Zeit-Diagramm am Beispiel der TTL-Technik (Kapitel 3)

Ein großer Vorteil der digitalen Informationsverarbeitung besteht darin, daß bei der Darstellung der beiden Binärzeichen durch elektrische Spannungen keine genauen Werte einzuhalten sind. In **Bild 4.4** ist eine Beispiel hierfür dargestellt. Alle Signale, die eine Spannung haben, welche zwischen 0V und 0,8V liegt, werden dem Binärzeichen L (Low) zugeordnet. Eine Spannung zwischen 2V und 5V dagegen bedeutet H (High).

Befehlsdarstellung

Wenn einem Mikroprozessor eine Dualzahl übermittelt wird, so kann sie für ihn die Bedeutung eines Befehlswortes oder eines Datenwortes haben, je nachdem zu welchem Zeitpunkt das Wort eintrifft.

Ein Befehlswort aktiviert bestimmte Fähigkeiten in einem Mikroprozessor. Das erste Wort (Operationsteil des Befehles) gibt Instruktionen an das Steuerwerk, und es kommt zur Ausführung einer Operation.

Die Menge der vom Mikroprozessor durchführbaren Operationen läßt sich in drei Gruppen einteilen.

- arithmetische Operationen +, —, . . .;
- logische Operationen >, =, . . .;
- organisatorische Operationen Verschieben, Springen, ...

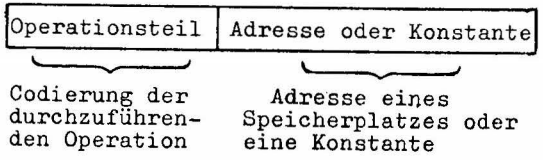

Bild 4.5. Aufbau eines Befehles

In **Bild 4.5** ist der typische Aufbau eines Mikroprozessor-Befehles dargestellt. Der Operationsteil dient zur Codierung der durchzuführenden Operation. Wenn der Operationsteil z. B. aus 8 bit besteht, so lassen sich hiermit $2^8 = 256$ verschiedene Operationen codieren (s. a. Abschnitt 6.2), d. h., jede Bitkombination aktiviert eine andere Tätigkeit des Mikroprozessors. Es gibt Befehle, die nur aus einem Wort bestehen und andere, an die noch eine Adresse oder eine Konstante angehängt werden muß.

Beispiele:

LLHHHHL → Erhöh den Inhalt des Rechenregisters um 1,
HHLLLLHH Adr. → Spring zur angegebenen Adresse,
LLHHHHL Konst. → Bring die Konstante ins Rechenregister.

Eine ausführliche Behandlung dieser Thematik erfolgt im Abschnitt 6.2 am Beispiel der Befehle des Mikroprozessors 8080/85.

Codierung alphanumerischer Zeichen

Der Informationsinhalt der Datenwörter, die mit dem Computer ausgetauscht werden, kann sehr unterschiedlich sein. Beim Beispiel der Waage in Bild 4.1 müssen von der Wägeeinrichtung die Gewichtsdaten übernommen und später der Preis in Dezi-

malziffern angezeigt werden. Da aber z. B. zur Anzeigeeinheit nur binäre Daten übertragen werden können, muß eine Code vereinbart werden, welcher eine Zuordnung zwischen den Dezimalziffern 0 . . . 9 und einem Binärwort herstellt.

Wie lang muß ein Binärwort sein, um die zehn Dezimalziffern darstellen zu können? Die Mathematik lehrt uns, daß mittels zweier verschiedener Zeichen (L und H), die zu einer Folge von s Zeichen aneinandergereiht werden, 2^s verschiedene Kombinationen gebildet werden können. Hieraus ergibt sich, daß 4 bit nötig sind, um die 10 Dezimalziffern zu codieren. Die Codierung erfolgt dann sinnvollerweise so, wie es bereits in Bild 4.2 dargestellt wurde.

Beispiele:

0 = LLLL, 1 = LLLH, . . ., 5 = LHLH, . . ., 9 = HLLH

Diese Form der Codierung wird auch BCD-Code (Binary Coded Decimals) oder auch 8-4-2-1-Code genannt. Die letzte Bezeichnúng gibt einfach die Wertigkeit der vier Bits wieder.

Da bei dieser Codierung immer vier Bits zusammengefaßt auftreten, ist auch noch die Bezeichnung Tetraden-Darstellung einer Dezimalzahl üblich (s. a. Abschnitt 4.2).

Beispiel:

Tetraden-Darstellung der Dezimalzahl 139

1	3	9
LLLH	LLHH	HLLH

Ein Teil der mit den vier Bit möglichen Kombinationen ($2^4 = 16$) wird nicht benötigt; diese Tetraden werden auch Pseudotetraden genannt. Es sind dies all diejenigen Tetraden, die beim Hexadezimalalphabet den Ziffern A . . . F entsprechen (s. a. Bild 4.2).

Wenn neben den numerischen Zeichen (0 . . . 9) auch noch die Buchstaben (Alphazeichen) und die Sonderzeichen (? + ; - / . . .) übertragen werden sollen, benötigt man mehr Bits.

Die Summe dieser Zeichen nennt man alphanumerische Zeichen oder alphanumerisches Alphabet.

Zur Codierung der alphanumerischen Zeichen werden zwei Tetraden benutzt. Diese beiden Tetraden ergeben zusammengefaßt ein 8-bit-Wort, das den namen Byte erhält:

1 Byte = 8 bit = 2 Tetraden = 2 HEX-Zeichen.

Beispiel:

HHLLLHHH = HHLL LHHH = C7

Die linken 4 bit werden höherwertige Tetrade und die rechten 4 bit niederwertige Tetrade genannt.

Mittels der höherwertigen Tetrade werden die gesamten alphanumerischen Zeichen in einzelne numerierte Zonen aufgeteilt, die jeweils 16 Zeichen umfassen. Die niederwertige Tetrade dagegen wird zur Durchnumerierung der 16 Zeichen einer Zone benutzt.

Beispiel: Darstellung des Buchstabens B

LHLL	LLHL
Zonenteil	Zifferteil

= 42H → Zone 4, drittes Zeichen

Durch den Zonenteil lassen sich 16 verschiedene Zonen adressieren, d. h., insgesamt wären 16 x 16 = 256 verschiedene Zeichen darstellbar. Dies ist jedoch im allgemeinen nicht erforderlich, deshalb wird beim sogenannten ASCII-Code das höchste Bit weggelassen und häufig als Paritätsbit zur Datensicherung benutzt.
Der ASCII-Code (American Standard Code of Information Interchange) wird in der Datenverarbeitung überwiegend angewandt. Neben allen aphanumerischen Zeichen beinhaltet der Code auch noch zwei Zonen mit insgesamt 32 sogenannten Steuerzeichen.

Paritätsbit oder

7	6	5	4	3	2	1	0	HEX	0	CTRL	1	CTRL	2	3	4	5	6	7
									L L L L		L L L H		L H L L	L H L H	H L L L	H L H L	H H L L	H H L H
								0	NUL	@	DLE	P	⌣	0	@	P	`	p
								1	SOH	A	DC 1 X-ON	Q	!	1	A	Q	a	q
								2	STX	B	DC 2 TAPE	R	"	2	B	R	b	r
								3	ETX	C	DC 3 X-OFF	S	#	3	C	S	c	s
								4	EOT	D	DC 4	T	$	4	D	T	d	t
								5	ENQ	E	NAK	U	%	5	E	U	e	u
								6	ACK	F	SYN	V	&	6	F	V	f	v
								7	BEL	G	ETB	W	'	7	G	W	g	w
								8	BS	H	CAN	X	(	8	H	X	h	x
								9	HT	I	EM	Y	)	9	I	Y	i	y
								A	LF	J	SUB	Z	*	:	J	Z	j	z
								B	VT	K	ESC	[	+	;	K	[	k	{
								C	FF	L	FS	/	,	<	L	\	l	|
								D	CR	M	GS	]	-	=	M	]	m	} ALT MODE
								E	SO	N	RS	^	.	>	N	^ ↑	n	~
								F	SI	O	US	-	/	?	O	_ ←	o	DEL RUB OUT

Bild 4.6. ASCII-Code-Tabelle. Die ersten beiden Spalten enthalten Control-Zeichen in der üblichen amerikanischen Bezeichnungsweise (⌣ = Leerschritt, blank)

Diese Steuerzeichen sind im allgemeinen unsichtbar, d. h., es ist ihnen kein sichtbares Zeichen zugeordnet. Mittels einer Standard-Tastatur lassen sie sich dadurch erzeugen, daß die sogenannte CTRL-Taste und eine Zeichen-Taste gleichzeitig gedrückt werden. In **Bild 4.6** sind alle gültigen ASCII-Zeichen zusammengestellt. Jede Spalte entspricht einer Zone. Die einem Zeichen zugeordnete Dualzahl (Byte) und die entsprechende HEX-Zahl, setzen sich jeweils aus den Teilen am oberen und am linken Tabellenrand zusammen.

Beispiele:
A = LHLLLLLH = 41H,
5 = LLHHLHLH = 35H,
t = LHHHLHLL = 74H.

4.4 Programmierung

Indem man einem Mikroprozessor eine Folge von Befehlen erteilt, kann er zu einer sinnvollen Tätigkeit aktiviert werden. Eine solche Befehlsfolge wird Programm genannt, wenn sie eine lückenlose und in einzelnen Operationsschritten angegebene Lösungsvorschrift zur Bearbeitung eines Problemes darstellt.

Flußdiagramm

Um sich über den Lösungsweg eines Problemes klar zu werden, hat es sich bewährt, zunächst ein Flußdiagramm anzufertigen.

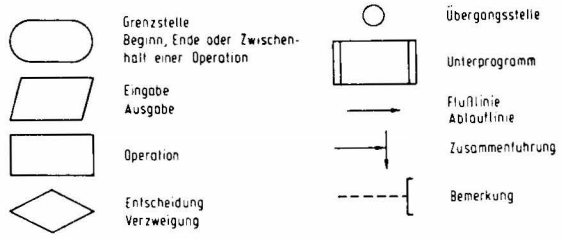

Bild 4.7. Flußdiagrammsymbole nach DIN 66 001

Zur Herstellung dieses Flußdiagrammes können die in **Bild 4.7** dargestellten Symbole benutzt werden.
Ein Flußdiagramm kann zur groben Strukturierung eines größeren Problemes oder aber zur detaillierten Darstellung eines Lösungsweges benutzt werden. Im zweiten Fall kann dies so weit gehen, daß in den Kästchen des Flußdiagrammes die einzelnen Befehle eines Programmes stehen. Ein einfaches Beispiel zur Lösung eines „alltäglichen" Problemes ist in **Bild 4.8** dargestellt.

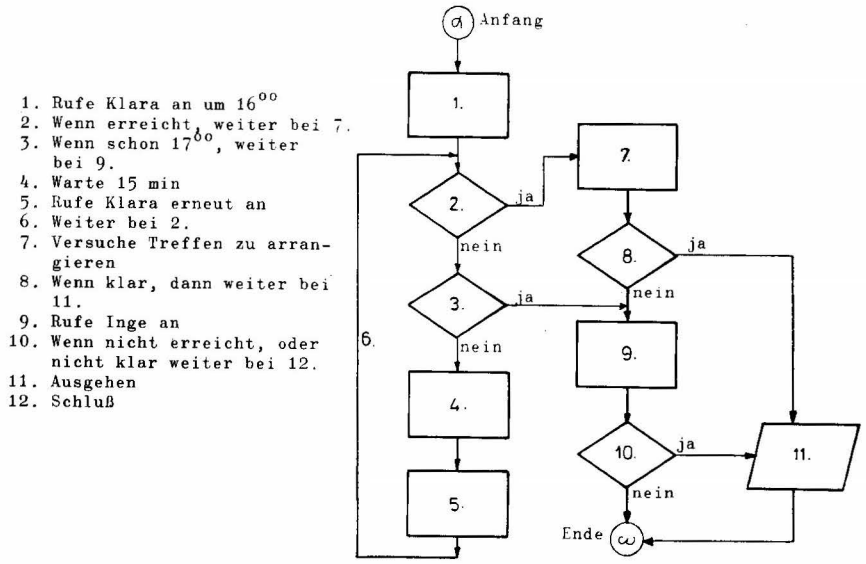

Bild 4.8. Flußdiagramm zur Lösung des Problemes einer Rendevous-Vereinbarung

Programmiersprachen

Zur Programmierung von Problemen auf Groß- und Minirechnern sowie auf Tischrechnern (z. B. Personal Computer) werden im allgemeinen höhere Programmiersprachen benutzt.

Beispiele:

- zur Lösung naturwissenschaftlich-technischer Probleme:
  ALGOL (= algorithmic language)
  FORTRAN (= formula translation)
- zur Lösung wirtschaftswissenschaftlicher und kommerzieller Probleme:
  COBOL (= common business oriented language)
- universelle Programmiersprachen:
  PL/1 (= programming language 1)
  PASCAL (benannt nach Blaise Pascal, 1623-1663, Erfinder der ersten Rechenmaschine).

Programme, die in einer solchen Programmiersprache geschrieben worden sind, werden dann von einem speziellen Übersetzerprogramm (Compiler) in den Maschinencode (Maschinensprache) des jeweiligen Computers umgewandelt.
Der Maschinencode ist für jeden Computer anders. Da es aber im allgemeinen für jeden Computer und jede höhere Programmiersprache einen Compiler gibt, sind Programme, geschrieben in einer höheren Programmiersprache, auf jedem Computer ablauffähig.

Obwohl fast jeder Mikroprozessor eine andere Maschinensprache (Maschinen- oder Objektcode) versteht und deshalb Programme, geschrieben für einen speziellen Mikroprozessor, von einem anderen nicht verstanden werden, wird bei technischen Anwendungen (Steuerungs- oder Regelaufgaben) meistens in der Maschinensprache programmiert. Der Grund dafür liegt in der größeren Effektivität solcher Programme; dies macht sich besonders in kürzeren Programmen und schnelleren Programmabläufen bemerkbar.
Im Kapitel 6 wird deshalb auf die Programmierung von Mikroprozessoren in der Maschinensprache eingegangen.
Zur Vermeidung unnötiger Verwirrungen wird hier nur ein Mikroprozessor — als Beispiel für all die anderen — behandelt.

5 Grundeinheiten eines Mikrocomputers

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Zum besseren Verständnis des nächsten Kapitels — dort wird die Programmierung in der Maschinensprache erklärt — soll hier kurz auf die wichtigsten Komponenten eines Mikrocomputers eingegangen werden.
Weitere Erörterungen zu diesem Thema enthält Kapitel 7.

5.1 Der Mikrocomputer

Das Kernstück eines Mikrocomputers (MC) ist der Mikroprozessor (MP). Die meisten Mikroprozessoren sind in einem Gehäuse mit den Abmessungen $5 \times 1{,}5 \times 0{,}5\ cm^3$ untergebracht. Der Chip im Inneren des Gehäuses ist über 40 Kontaktbeinchen mit der Außenwelt verbunden.

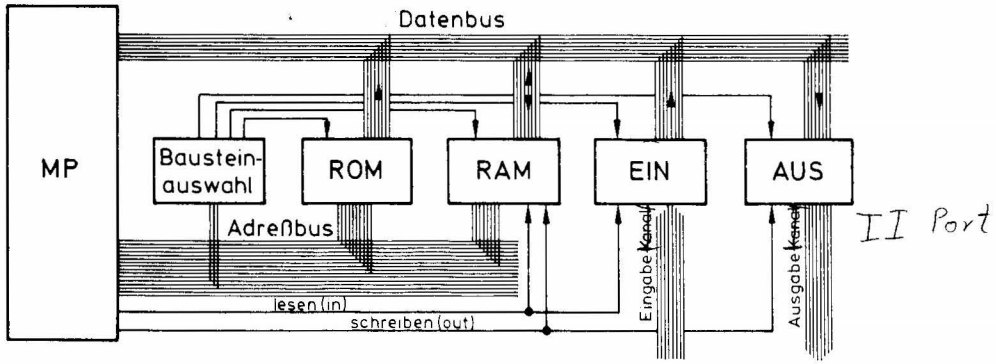

Bild 5.1. Aufbau eines Mikrocomputers
ROM (Read Only Memory) = Programmspeicher;
RAM (Random Access Memory) = Arbeitsspeicher;
Ein/Aus = Schnittstellen zum Datenaustausch mit der Umwelt

Damit der Mikroprozessor arbeiten kann, braucht er einen Programmspeicher, aus dem er sich seine Befehle holen kann. Hierzu wird im allgemeinen ein ROM (Read Only Memory) benutzt. Diese Lesespeicher sind integrierte Halbleiterbausteine, die bei Spannungsausfall ihren Inhalt behalten. Dies ist wichtig, damit das Programm beim Ausschalten nicht verloren geht.
Damit der Mikroprozessor sich Zwischenergebnisse oder andere Daten merken kann, benötigt er noch einen zusätzlichen Speicher — einen sogenannten Arbeitsspeicher —in den er wahlweise etwas hineinschreiben oder herauslesen kann. Hierzu werden

wiederum Halbleiterspeicher, sogenannte RAMs (Random Access Memory), benutzt. Zur Vervollständigung des Mikrocomputers müssen noch ein oder mehrere Ein-/Ausgabe Bausteine hinzugefügt werden, denn nur dann kann der Mikroprozessor eine sinnvolle Funktion erfüllen, indem er über diese Schnittstellen (interface) mit seiner Umwelt in Kontakt tritt. Wie aus **Bild 5.1** ersichtlich, sind die Bausteine, welche zusammen den Mikrocomputer bilden, über sogenannte Busse miteinander verbunden. Ein Bus ist eine mehradrige Sammelschiene für den Informationstransport. Im Mikrocomputer gibt es den Datenbus — der heute meistens aus acht Leitungen besteht — zur Ein- und Ausgabe von Datenwörtern, wie z. B. Meßergebnisse, Rechenergebnisse, externe Steuersignale und zum Einholen von Befehlswörtern.
Der Adreßbus hat häufig 16 Leitungen. Er dient dem Mikroprozessor dazu, den anderen Bausteinen mitzuteilen, wer und welches Teil in dem angesprochenen Baustein über den Datenbus etwas mitteilen oder eine Information entgegennehmen soll. Über den Adreßbus werden also vom Mikroprozessor nur Adressen hinausgegeben, während auf dem Datenbus Informationen in beiden Richtungen fließen können.
Da alle Bausteine am Datenbus angeschlossen sind, wird eine Zusatzschaltung benötigt, welche die vom Mikroprozessor ausgesandten Adressen in sogenannte Bausteinauswahlsignale (Chip Select, CS; Chip Enable, CE) umwandelt. auf diese Weise wird erreicht, daß immer nur ein Baustein über den Datenbus mit dem Mikroprozessor verbunden ist; alle anderen befinden sich dann im sogenannten hochohmigen Zustand (s. a. Bild 3.8). Diese Bausteinauswahl kann z. B. durch einen einfachen 1-aus-8-Decoder vorgenommen werden. Die Bausteinauswahlleitungen und einige weitere Leitungen — z. B. diejenigen, die anzeigen, ob der Mikroprozessor lesen oder schreiben (in/out) will — werden auch Steuerbus genannt.
Die Weiterentwicklung der Halbleitertechnologie führt zu immer höheren Integrationsdichten bei den Bausteinen. Hierdurch ist es gelungen, die soeben beschriebenen Einheiten eines Mikrocomputers in einem Baustein zu vereinen (single chip computer).

Beispiel:
Der Single-Chip-Mikrocomputer 8051 beinhaltet neben dem Mikroprozessor:
4096 Byte Programmspeicher,
128 Byte Arbeitsspeicher,
32 Ein-/Ausgabe-Kanäle, und zusätzlich noch einige spezielle Komponenten.

5.2 Die Speicher

Der Speicher eines Computers dient zur Aufnahme des Programmes und als Arbeitsspeicher.
Der Programmspeicher setzt sich meistens aus einem sogenannten residenten Teil (Festwertspeicher) zur Aufnahme des Grundprogrammes (Betriebssystem oder Monitor) und einem frei programmierbaren Teil zur Aufnahme von Anwenderprogrammen zusammen.

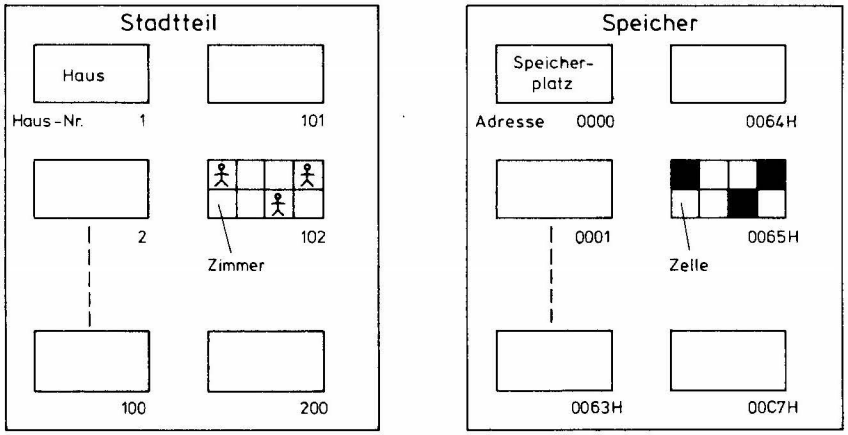

Bild 5.2. Vergleich zwischen einem Stadtteil und einem Speicher

Die Speicher im Mikrocomputer bestehen heute ausschließlich aus sogenannten Halbleiterspeichern. Darüber hinaus gibt es noch externe Massenspeicher zur Aufnahme sehr großer Programm- oder Datenmengen.
Die Struktur eines solchen Speichers geht aus **Bild 5.2** hervor.
Ein Speicher, meist in der Form eines Halbleiterbausteines (IC), besteht aus einer großen Menge von Speicherplätzen, entsprechend den Häusern in einem Stadtteil. So, wie die Häuser durchnumeriert sind, erhalten die Speicherplätze eine fortlaufende Adresse, die allerdings mit 0 beginnt. Diese Adressierung wird im allgemeinen hexadezimal vorgenommen, da der Computer ja nur die den HEX-Zeichen zugeordneten binären Tetraden „verstehen" kann.
Bei 8-bit-Mikroprozessoren enthält jeder Speicherplatz acht Zellen, den acht Zimmern eines Hauses vergleichbar. Jede Zelle kann ein Bit (L oderH) speichern, äquivalent dazu, ob ein Zimmer bewohnt (H) oder unbewohnt (L) ist.
Der Speicherplatz mit der Adresse 0065H in Bild 5.2 enthält dann die Information HLLHLLHL oder hexadezimal 92.

5.3 Der Mikroprozessor

Zum Verständnis der Möglichkeiten, die bei der Erstellung eines Mikroprozessorprogrammes (Kapitel 6) zur Verfügung stehen, ist es erforderlich, einige innere Strukturelemente von Mikroprozessoren kennenzulernen. Dies soll am Beispiel des Mikroprozessors 8080 erfolgen.
Die Komponenten eines Mikroprozessors sind auf einem einzelnen Chip untergebracht und durch ein Bus-System miteinander verbunden, ähnlich wie bei einem Mikrocomputer. Insbesondere der externe Datenbus setzt sich ins Innere des Mikroprozessors fort. Die meisten der zur Zeit verfügbaren Mikroprozessoren haben einen

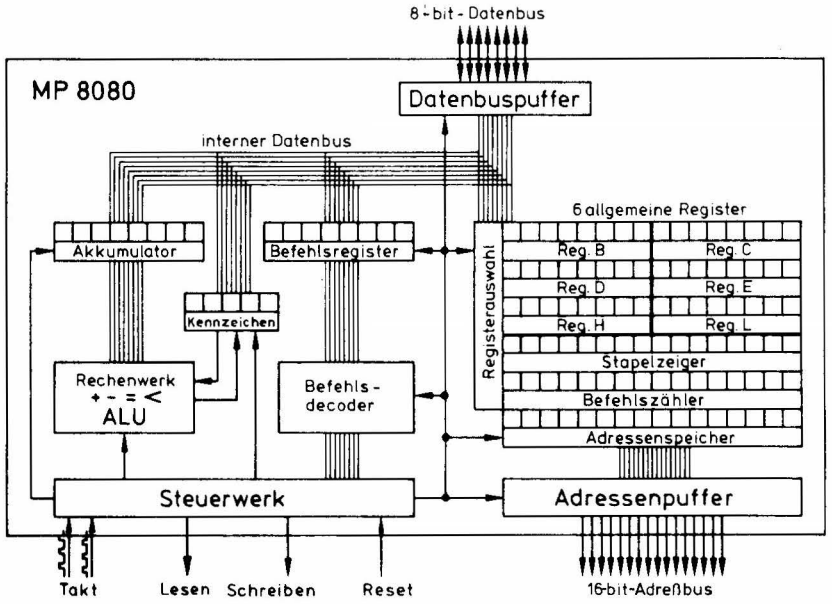

Bild 5.3. Innerer Aufbau des Mikroprozessors 8080 in vereinfachter Darstellung

8-bit-Datenbus, so wie der in **Bild 5.3** dargestellte Mikroprozessor 8080. Sie werden deshalb auch 8-bit-Mikroprozessoren genannt. Der Adreßbus ist häufig 16 bit breit, d. h., eine auf ihm parallel ausgesandte Adresse ist durch vier HEX-Zeichen darstellbar.

Beispiel:
Logischer Zustand der 16 Adreßbusleitungen
L L H L H H L L H L L L L H H H = LLHL HHLL HLLL LHHH = 2C87H.

Wenn z. B. der 100. Speicherplatz in Bild 5.2 den Inhalt seiner acht Zellen auf den Datenbus geben soll, muß der Mikroprozessor auf dem Adreßbus die Adresse 0063H aussenden.
Beinhaltet der angesprochene Speicherplatz z. B. ein Befehlswort (Byte), so wird es über den Datenbus in das Befehlsregister eingeschrieben. Der Befehlsdecoder interpretiert dann den Befehl, und das Steuerwerk steuert die Befehlsabarbeitung.
Arithmetisch-logische Operationen werden im Rechenwerk des Mikroprozessors durchgeführt (ALU = Arithmetic Logic Unit).
Weitere wichtige Bestandteile sind die in Bild 5.3 als eine Reihe von Kästchen dargestellten Register. Diese Kästchen symbolisieren einzelne Speicherzellen, d. h., die Register stellen 8- oder 16-bit-Speicherplätze dar.
Neben den sechs allgemeinen 8-bit-Registern gibt es noch das Rechenregister, den

sogenannten Akkumulator (Akku). Wie man später — bei der Beschreibung der Befehle — sehen wird, nimmt der Akku eine Sonderstellung unter den Registern ein.

Jeweils zwei der allgemeinen Register lassen sich zu sogenannten Registerpaaren zusammenfassen, auf diese Weise erhält man die drei 16-bit-Register BC, DE und HL. Zwei Sonderregister von doppelter Bitbreite sind das Befehlszählregister (program counter, PC) und der Stapelzeiger (stack pointer, SP). Die Bedeutung dieser Register wird später beschrieben.

Befehlsablauf
Das Steuerwerk im Mikroprozessor sorgt für den richtigen zeitlichen Ablauf aller Vorgänge im Mikrocomputer. Hierzu benötigt es einen von außen vorgegebenen Takt, der meistens über einen Schwingquarz erzeugt wird und deshalb sehr frequenzgenau ist.
Im Rhythmus dieses Taktes werden aus dem Programmspeicher Befehlsbytes in den Mikroprozessor geholt und abgearbeitet.
Die verschiedenen Befehle können aus einer unterschiedlichen Anzahl von Bytes bestehen und auch in ihrer Komplexität variieren. Deshalb erfordern die einzelnen Befehle zwischen 4 bis 18 Takte (8080) zum Einholen und Abarbeiten.
Wenn die Taktfrequenz z. B. $f = 2$ MHz beträgt, ergibt sich eine Befehlszykluszeit von 2 bis 9 μs.

Beispiel:
Eine 8-bit-Addition benötigt vier Takte.
Taktzeit $T = 1/f = 5 \times 10^{-7}$ s, 4 $T = 2$ μs.
Das ergibt 500 000 Additionen/s.

Wenn ein Mikrocomputer eingeschaltet wird, sorgt eine spezielle Schaltung (Einschaltreset, Bild 8.1) dafür, daß der Mikroprozessor immer in gleicher Weise seine Arbeit beginnt.
Zunächst muß der Mikroprozessor seinen ersten Befehl holen, damit er mit einer sinnvollen Tätigkeit anfangen kann. Aus diesem Grunde sendet der Mikroprozessor zu Beginn die Adresse des ersten Befehlsbytes und ein Lesesignal aus. Diese Startadresse wird nach jedem Reset vom Mikroprozessor selbständig zunächst ins Befehlszählregister (BZR) geschrieben und dann der Inhalt des BZR auf den Adreßbus gegeben. Beim MP 8080 hat diese zuerst anstehende Adresse immer den Wert 0000, d. h., das erste Befehlsbyte muß stets im ersten Programmspeicherplatz stehen.

6 Programmerstellung und Abarbeitung

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Aufbauend auf den Kenntnissen über die Darstellung und den Ablauf von Befehlen in einem Mikroprozessor, sowie dessen innerer Struktur, soll nun dargelegt werden, wie mittels dieser Befehle Programme erstellt werden können und wie der Mikroprozessor sie abarbeitet. Über die allgemeine und grundsätzliche Erörterung hinaus wird hier immer der Mikroprozessor 8080 bzw. der weiterentwickelte Typ 8085 als Beispiel herangezogen (s. a. [1]).

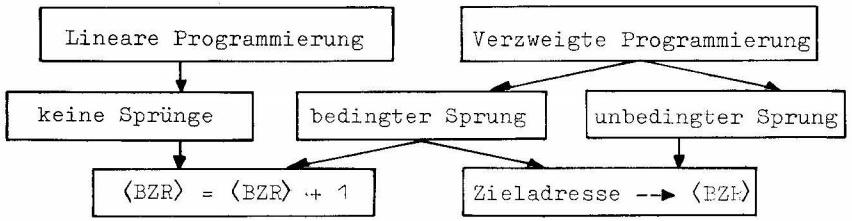

Bild 6.1. Programmierungsarten (BZR = Befehlszählregister)

Doch zuvor soll anhand von **Bild 6.1** kurz auf die lineare und die verzweigte Programmierung eingegangen werden.
Eine Verzweigung in einem Programm wird immer durch Sprünge erreicht, d. h., anstelle der nächsten Befehlsadresse wird die des Sprungzieles in das Befehlszählregister geschrieben. Wobei zu diesen Sprüngen auch Unterprogrammaufrufe und Unterprogrammrücksprünge gehören (Abschnitt 6.2).
Neben den unbedingten Sprüngen, welche immer ausgeführt werden, gibt es jedoch auch noch die bedingten Sprünge. Bei einem bedingten Sprungbefehl wird dieser nur ausgeführt, wenn eine bestimmte Bedingung erfüllt ist, andernfalls wird das Programm linear weiter abgearbeitet.
Die bedingten Sprungbefehle stellen ein besonders wichtiges Hilfsmittel bei der Programmierung dar.

6.1 Die Befehle

Kennzeichenbits
Bei der Bearbeitung von Befehlen können verschiedene Ereignisse auftreten, z. B. wenn das Ergebnis einer arithmetischen Operation Null ist oder wenn eine zu große Zahl entsteht. Ein Mikroprozessor hat deshalb eine Reihe von Speicherzellen, durch die er sich das Eintreten solcher Ereignisse merken kann. Für jedes unterschiedliche

Ereignis ist eine bestimmte Speicherzelle zuständig. Wenn das Ereignis eintritt, wird ein H in die Speicherzelle geschrieben, andernfalls ein L. Das Einschreiben von H in die Speicherzelle wird „Setzen" des Kennzeichenbits (flag) genannt. Wird dagegen ein L eingeschrieben, weil das Ereignis nicht eingetreten ist, so spricht man vom „Rücksetzen" oder „Nicht-Setzen" des betreffenden Kennzeichenbits. Diese Kennzeichenbits sind es nun, von deren Zustand die Ausführung eines bedingten Sprunges abhängig gemacht wird.
Zwei der wichtigsten Kennzeichenbits sind das Zero- und das Carry-Flag.
Wenn im folgenden ein Buchstabe oder eine Buchstabenkombination in spitzen Klammern steht, so bedeutet dies immer „Inhalt von . . ." (z. B. ⟨ Z ⟩ = Inhalt des Zero-Flags).

Null (Zero) Kennzeichen: Z
- das Ergebnis eines Befehles ist 0 : H → ⟨ Z ⟩;
- das Ergebnis ist ungleich 0 : L → ⟨ Z ⟩;

Übertrag (Carry) Kennzeichen: C

Wenn ein Übertrag entsteht, d. h. eine Zahl ist zu groß geworden, um mit den 8 bit des Akkumukators noch darstellbar zu sein, wird ein H in den Übertrags-Kennzeichenspeicher geschrieben.

Beispiel:
Addition zweier 8-bit-Registerinhalte

```
      HLHHLLHH
    + LHHHLHLL
    ----------
    H HHH        Übertrag
⟨C⟩→H LLHLLHHH   Ergebnis länger als 8 bit.
```

Die Kennzeichen werden bei der Abarbeitung eines Befehles gesetzt oder nicht gesetzt. Dies bedeutet jedoch nicht, daß jeder Befehl die Kennzeichenbits beeinflußt. Es gibt vielmehr eine Reihe von Befehlsarten, die nur einige der Kennzeichenbits oder auch gar keine beeinflussen. In diesen Fällen bleiben die alten Zustände der Kennzeichenspeicher erhalten.

Befehlsaufbau
Der Mikroprozessor 8080 bzw. 8085 verfügt über Befehle unterschiedlicher Länge.

Bitmuster	Hex-Code	Mnemonic-Code
niederwertig / höherwertig H H L L L L H H L L H H L H L L L L L H L L H L Operationsteil — Adreßteil eines Befehls	C3 34 12	JMP 1234_H

Bild 6.2. Der Befehl „Spring zum Befehl mit der Adresse 1234H" in seinen drei Darstellungsformen

Ein einfacher Verschiebebefehl — z. B. „Kopier den Inhalt des Registers B in das Register E" — besteht nur aus einem Byte. Wenn dagegen in ein Register eine Zahl geladen werden soll, muß dem ersten Byte noch das zu ladende Byte als zweites folgen. Der längste Befehl besteht aus drei Bytes. Dies ist z. B. bei einem Sprungbefehl der Fall, da dem ersten Byte eine 16-bit-Adresse (2 Byte) folgen muß (**Bild 6.2**).
Der hier dargestellte Sprungbefehl zeigt die Aufteilung der Befehlsinformation auf die drei Bytes. Nach dem Opertionsbyte (Operationsteil) folgt zunächst das niederwertige Adreßbyte und dann das höherwertige Adreßbyte. Die beiden Adreßbytes stehen also vertauscht in Bezug auf die normale Stellenschreibweise einer Zahl. Dies wird besonders deutlich, wenn man das Befehls-Bitmuster in die entsprechenden HEX-Zeichen umformt und diese dann mit der Adresse im sogenannten Mnemonic-Code vergleicht.

Hex-Code
Als Hex-Code eines Befehles wird die Umwandlung der den Befehl repräsentierenden Dualzahl in eine Hexadezimalzahl verstanden.
Dieser Code ist besser les- und handhabbar als der Binär-Code. Zur Eingabe eines Programmes im Hex-Code reicht eine Tastatur mit 16 Tasten aus. Jeder Taste ist dann ein HEX-Zeichen zugeordnet. Einfache Mikrocomputer verfügen meistens über solch eine Tastatur und eine mehrstellige Anzeigeeinheit, auf der sich HEX-Zeichen darstellen lassen (**Bild 6.3**).

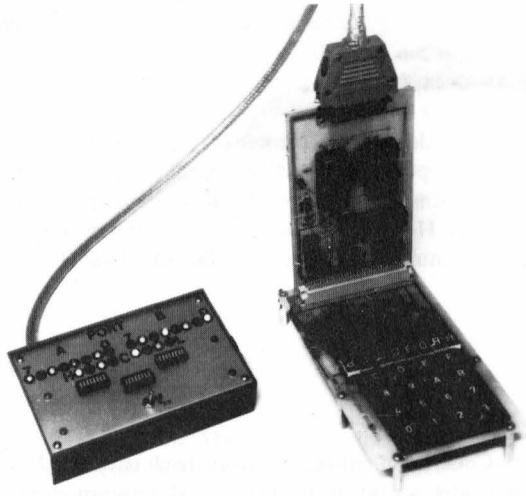

Bild 6.3. Im Hex-Code programmierbarer Mikrocomputer (MICO 85) mit 8stelliger Hexadezimalanzeige.
Die Mikrocomputerplatine ist unterhalb der Tastatur-/Anzeigeplatine angebracht. Die senkrecht stehende Platine dient zur Programmierung von Speicherbausteinen (EPROMs) und/oder zur Steuerung von Maschinen etc. (Werkbild: PTFE, s. a. [4] in Kapitel 8).

Das Programmieren im Hex-Code wird jedoch nur zu Lehrzwecken, bei einfachen Programmen oder bei kleinen Programmkorrekturen angewandt. Zur Erstellung längerer Programme sollte man sich des sogenannten Mnemonic-Codes bedienen.

Mnemonic-Code

Beim Mnemonic-Code (Mneme = Gedächtnis) wird der Operationsteil eines Befehls durch eine Buchstabenkombination ersetzt, die sich aus der Befehlsfunktion herleitet. Bei 2- oder 3-Byte-Befehlen werden die zusätzlichen Bytes im Hex-Code angehängt. Eine Vertauschung des niederwertigen und höherwertigen Bytes wird nicht vorgenommen.

Beispiele:

JMP 12DO : Spring (jump) zur Adresse 12DO;
MOV B, C : Kopier den Inhalt des Registers C ins Register B;
ADI 3E : Addier zum Akku 3E.

Zur Eingabe dieser Befehle in den Speicher eines Mikrocomputers ist eine Tastatur wie bei einer Schreibmaschine erforderlich, denn es müssen ja die mnemonischen Abkürzungen eingegeben werden. Die Buchstaben des Mnemonic-Teils werden deshalb durch die Eingabetastatur im ASCII-Code (Abschnitt 3.3) an den Mikrocomputer weitergegeben.

Beispiel:

JMP 12DO → 4A 4D 50 20 31 32 44 30.

Die Befehle, die auf diese Weise in den Speicher eines Mikrocomputers gelangen, müssen anschließend mit Hilfe eines speziellen, im Mikrocomputer bereits vorhandenen Programmes in den Binär-Code umgewandelt werden. Häufig spricht man hierbei auch von einer Umwandlung in den Hex-Code, da dieser ja unmittelbar dem Binär-Code zugeordnet ist und eigentlich nur eine andere Schreibweise darstellt.

Beispiele:

JMP 12DO → C3DO12;
MOV B, C → 41;
ADI 3E → C63E.

Dieses Übersetzen in den Hex-Code kann entweder automatisch nach der Eingabe eines jeden Befehles geschehen oder nachdem das gesamte Programm eingegeben wurde.
Die Übersetzung erfogt z. B. anhand einer Tabelle, die sich im Speicher des Mikrocomputers befindet. Diese Tabelle enthält den zu jeder mnemonischen Abkürzung zugehörigen Hex-Code.
Bei den meisten Befehlen kommen zu dem Mnemonic-Code noch weitere Angaben hinzu. Diese machen Aussagen über betroffene Register, über Daten oder Adressen.

Diesen zweiten Teil nennt man die Operanden des Befehles.

Beispiele:

Mnemonic-Code (Operationscode)
Operanden
Kommentare

MOV A, B : Kopier den Inhalt von B in den Akku;
OUT 18 : Gib den Inhalt des Akkus auf den Port 18;
CMP E : Vergleich den Inhalt von E mit A;
MVI C,30 : Bring das Datum 30H ins Register C.

Eine Erweiterung des Mnemonic-Codes stellt die Assembler-Programmiersprache dar. Hierauf wird später noch kurz eingegangen.

Befehlsliste des MP 8080/85

In **Bild 6.4** sind alle Befehle des MP 8080 und die zusätzlichen Befehle des MP 8085 dargestellt.
Von den zusätzlichen Befehlen des MP 8085 (Bild 6.4d) sind nur die ersten zwei offiziell bekannt. Die zehn weiteren Befehle sind zwar zur Zeit vorhanden, es ist jedoch nicht auszuschließen, daß sie im Laufe der Zeit vom Hersteller modifiziert werden. Da der MP 8085, genau wie der MP 8080, von vielen verschiedenen Herstellern angeboten wird, kann es bezüglich dieser zusätzlichen zehn Befehle langfristig sogar zu unterschiedlichen Veränderungen kommen.
In der ersten Spalte der Befehlsliste sind die sogenannten Mnemonics, d. h. der Mnemonic-Code der Befehle in international gebräuchlicher Form, dargestellt. Die zweite Spalte gibt jeweils an, aus wieviel Byte der Befehl besteht.
Der Hex-Code des Befehles oder ein Verweis (s. T.) auf die rechts am Rand stehenden Tabellen ist in der Spalte 3 angegeben. Diese zweistellige Hex-Zahl stellt jeweils den Operationsteil des Befehles dar. In den Fällen, wo im ersten Befehlsbyte bereits variable Adressierungen enthalten sind, geben die zugehörigen Tabellen den entsprechenden Hex-Code wieder.

Beispiele:

MOV A, B → 78;
MVI H, MVI H, 5 F → 265 F.

Die Spalte 4 enthält die Abkürzungsbuchstaben derjenigen Kennzeichenbits (Flags), die bei der Befehlsausführung beeinflußt werden.
In den Spalten 5 und 7 sind die Anzahl der Taktzyklen wiedergegeben, die zum Abarbeiten des Befehles beim MP 8080 bzw. beim MP 8085 erforderlich sind. Bei bedingten Sprüngen oder Unterprogrammaufrufen variiert die erforderliche Anzahl der Taktzyklen in Abhängigkeit davon, ob die Bedingung erfüllt ist oder nicht. Wenn eine Bedingung nicht erfüllt ist, wird ein bedingter Sprung auch nicht ausgeführt, und der Mikroprozessor braucht die Sprungadresse nicht einzulesen. Hierdurch wird die Befehlsausführungszeit kürzer.

a)

BEFEHLSLISTE DES 8080/85

Mnemonic	Bytes	Hex. Code	Flagbeeinflussung *	Taktzyklen	Funktion des Befehls	Taktzyklen 8085
1 TRANSFERBEFEHLE						
1.1 Register→Register						
MOV r_1,r_2	1	s.T.	--	5	r_1,r_2 = A,B,C,D,E,H,L . Lade Reg. r_1 mit dem Inh. von Reg. r_2	4
XCHG	1	EB	--	4	Vertausche Inhalte der Registerpaare (D,E) und (H,L)	4
XTHL	1	E3	--	18	Vertausche Inh. d. Reg.-Paares (H,L) u. d. Inh. d. Wortes, das durch den Stapelzeiger adressiert ist	16
SPHL	1	F9	--	5	Lade Stapelzeiger mit dem Inh. d. Reg.-Paares (H,L)	5
1.2 Speicher, Peripherie→Register						
MOV r_1,M	1	s.T.	--	7	r_1 = A,B,C,D,E,H,L. Lade Reg. r_1 mit d. Inh. d. Speicherplatzes,das durch d. Inh. d. Reg.-Paares (H,L) adressiert ist	7
LDA adr	3	3A	--	13	Accu laden mit Inh. der Adresse adr	13
LDAX rp	1	s.T.	--	7	rp = B,D. Accu laden mit Inh. d. Speicherplatzes, der durch den Inh. d. Reg.-Paares rp adressiert ist	7
LHLD adr	3	2A	--	16	Lade Reg.-Paar (H,L) mit d. Inh. der Adressen (adr+1) und adr	16
POP rp PSW	1	s.T. F1	Z,S,P,C,A	10	rp = B,D,H,PSW. Reg.-Paar rp wird mit d. Wort geladen, das durch d. Stapelzeiger adressiert ist (PSW = Accu + Flags)	10
IN nr	2	DB	--	10	Accu wird mit d. Inh. d. Eingabekanals Nummer nr (nr≤255) geladen	10
1.3 Konstante→Registerpaar						
LXI rp,adr	3	s.T.	--	10	rp = B,D,H,SP. Lade Reg.-Paar rp mit wert adr	10
1.4 Register→Speicher, Peripherie						
MOV M,r_1	1	s.T.	--	7	r_1 = A,B,C,D,E,H,L. Inh. v. Reg. r_1 auf d. Speicherplatz abspeichern, d. durch d. Inh. d. Reg.-Paares (H,L) adressiert ist	7
STA adr	3	32	--	13	Accu-Inh. unter Adresse adr abspeichern	13
STAX rp	1	s.T.	--	7	rp = B,D. Accu in dem Byte abspeichern, das durch den Inh. d. Reg.-Paares rp_1 adressiert ist	7
SHLD adr	3	22	--	16	Reg.-Paar (H,L) unter (adr+1) und adr abspeichern	16
PUSH rp	1	s.T.	--	11	rp = B,D,H,PSW. Inh. d. Reg.-Paares rp wird in das Wort übertragen, das durch den Stapelzeiger adressiert ist	12
OUT nr	2	D3	--	10	Accu-Inh. wird auf Ausgabekanal (Nummer nr ≤ 255) ausgegeben	10
1.5 Konstante→Register						
MVI M ,konst	2	36	--	10	Bringe d. Wert d. Konstanten (konst ≤ 255) in d. Speicherplatz, der durch d. Inh. d. Reg.-Paares (H,L) adressiert ist	10
MVI r_1, konst	2	s.T.	--	7	r_1 = A,B,C,D,E,H,L. Lade Reg. r_1 mit Konstante (konst≤255)	7

* A = Auxiliary Carry (HilfsÜbertrag) , Z = Zero (Null) , S = Sign (Vorzeichen) , P = Parity (Parität) , C = Carry (Übertrag)

MOV	A	B	C	D	E	H	L	M
A	7F	78	79	7A	7B	7C	7D	7E
B	47	40	41	42	43	44	45	46
C	4F	48	49	4A	4B	4C	4D	4E
D	57	50	51	52	53	54	55	56
E	5F	58	59	5A	5B	5C	5D	5E
H	67	60	61	62	63	64	65	66
L	6F	68	69	6A	6B	6C	6D	6E
M	77	70	71	72	73	74	75	

z.B. → MOV M,D (D)→(M)

LDAX	B	D
	0A	1A

POP	B	D	H	PSW
	C1	D1	E1	F1

LXI	D16 ← 16Bit Datenwort
B	01
D	11
H	21
SP	31

STAX	B	D
	02	12

PUSH	B	D	H	PSW
	C5	D5	E5	F5

MVI	D8
A	3E
B	06
C	0E
D	16
E	1E
H	26
L	2E
M	36

Bild 6.4. Vollständige Befehlsliste des 8080/85; s. T. bedeutet: siehe in der Tabelle am rechten Rand nach a) die Transferbefehle

b)

2 ARITHMETISCHE OPERATIONEN						
INR r_1	1	s.T.	Z,S,P,A	5	r_1 = A,B,C,D,E,H,L. Zum Inh. d. Reg. r_1 wird 1 addiert	4
INR M	1	s.T.	Z,S,P,A	10	Zum Inh. d. durch Reg.-Paar (H,L) adressierten Bytes wird 1 addiert	10
DCR r_1	1	s.T.	Z,S,P,A	5	r_1 = A,B,C,D,E,H,L. Vom Inh. d. Reg. r_1 wird 1 subtrahiert	4
DCR M	1	35	Z,S,P,A	10	Vom Inh. d. durch Reg.-Paar (H,L) adressierten Bytes wird 1 subtrahiert	10
INX rp	1	s.T.	--	5	rp = B,D,H,SP. Der Inh. d. Reg.-Paares rp_1 wird um 1 erhöht	6
DCX rp	1	s.T.	--	5	rp = B,D,H,SP. Der Inh. d. Reg.-Paares rp_1 wird um 1 erniedrigt	5
ADD r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Inh. v. Reg. r_1 wird zum Inh. des Accus addiert	4
ADD M	1	86	Z,S,P,C,A	7	Der Inh. d. Speicherbytes, das durch d. Inh. d. Reg.-Paares (H,L) adressiert ist, wird zum Inh. d. Accus addiert	7
ADC r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Inh. d. Reg. r_1 u. Inh. d. Carry-Bits werden zum Inh. des Accus addiert	4
ADC M	1	8E	Z,S,P,C,A	7	Inh. d. Speicherbytes,das durch d. Inh. d. Reg.-Paares (H,L) adressiert ist u. d. Inh. d. Carry-Bits werden zum Inh. d. Accus addiert	7
DAD rp	1	s.T.	C	10	rp = B,D,H,SP. Inh. d. Reg.-Paares rp_1 u. d. Inh. d. Reg.-Paares (H,L) werden addiert. Ergebnis in (H,L)	10
SUB r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Inh. d. Reg. r_1 wird v. Accu-Inh. subtrahiert	4
SUB M	1	96	Z,S,P,C,A	7	Inh. d. Speicherbytes, das durch d. Reg.-Paar (H,L) adressiert ist, wird vom Accu subtrahiert	7
SBB r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Inh. v. Reg. r_1 u. Inh. des Carry-Bits werden v. Accu-Inh. subtrah.	4
SBB M	1	9E	Z,S,P,C,A	7	Inh. d. Speicherbytes, das durch d. Reg.-Paar (H,L) adressiert ist, u. Inh. des Carry-Bits werden vom Accu subtrahiert	7
ADI konst	2	C6	Z,S,P,C,A	7	Konstante (konst≤255) wird zum Inh. d. Accu addiert	7
ACI konst	2	CE	Z,S,P,C,A	7	Zum Accu-Inh. werden konst≤255 u. Carry-Bit addiert	7
SUI konst	2	D6	Z,S,P,C,A	7	konst≤255 wird vom Accu-Inh. subtrahiert	7
SBI konst	2	DE	Z,S,P,C,A	7	konst≤255 und das Carry-Bit werden v. Accu-Inh. subtrahiert	7
DAA	1	27	Z,S,P,C,A	4	Accu-Inh. wird in eine 2stellige Zahl umgewandelt	4

INR	A	B	C	D	E	H	L	M
	3C	04	0C	14	1C	24	2C	34

DCR	A	B	C	D	E	H	L	M
	3D	05	0D	15	1D	25	2D	35

INX	B	D	H	SP
	03	13	23	33

DCX	B	D	H	SP
	0B	1B	2B	3B

ADD	A	B	C	D	E	H	L	M
	87	80	81	82	83	84	85	86

ADC	A	B	C	D	E	H	L	M
	8F	88	89	8A	8B	8C	8D	8E

DAD	B	D	H	SP
	09	19	29	39

SUB	A	B	C	D	E	H	L	M
	97	90	91	92	93	94	95	96

SBB	A	B	C	D	E	H	L	M
	9F	98	99	9A	9B	9C	9D	9E

3 LOGISCHE OPERATIONEN						
CMA	1	2F	--	4	Accu-Inh. wird negiert	4
ANA r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Accu-Inh. u. d. Inh. d. Reg. r_1 werden UND-verknüpft	4
ANA M	1	A6	Z,S,P,C,A	7	Der Inh. d. durch Reg.-Paar (H,L) adressierten Bytes wird mit d. Accu-Inh. UND-verkn.	7
ANI konst	2	E6	Z,S,P,C,A	7	Accu-Inh. wird mit der konst≤255 UND-verknüpft	7
ORA r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Accu-Inh. wird mit d. Inh. d. Reg. r_1 ODER-verknüpft	4
ORA M	1	B6	Z,S,P,C,A	7	Inh. d. über Reg.-Paar (H,L) adressierter Bytes wird mit d. Accu-Inh. ODER-verknüpft	7
ORI konst	2	F6	Z,S,P,C,A	7	Accu-Inh. wird mit konst≤255 ODER-verknüpft	7
XRA r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Accu-Inh. wird mit d. Inh. d. Reg. r_1 Exclusiv-ODER-verknüpft	4
XRA M	1	AE	Z,S,P,C,A	7	Das über Reg.-Paar (H,L) adressierte Byte wird durch Exclusive-ODER mit d. Accu verkn.	7
XRI konst	2	EE	Z,S,P,C,A	7	Der Accu wird mit d. wert konst≤255 durch Exclusive-ODER verknüpft	7
CMP r_1	1	s.T.	Z,S,P,C,A	4	r_1 = A,B,C,D,E,H,L. Accu-Inh. wird mit d. Inh. d. Reg. r_1 verglichen	4
CMP M	1	BE	Z,S,P,C,A	7	Accu-Inh. wird mit d. Inh. d. durch Reg.-Paar (H,L) adressierter Bytes verglichen	7
CPI konst	2	FE	Z,S,P,C,A	7	Accu-Inh. wird mit konst≤255 verglichen	7

ANA	A	B	C	D	E	H	L	M
	A7	A0	A1	A2	A3	A4	A5	A6

ORA	A	B	C	D	E	H	L	M
	B7	B0	B1	B2	B3	B4	B5	B6

XRA	A	B	C	D	E	H	L	M
	AF	A8	A9	AA	AB	AC	AD	AE

CMP	A	B	C	D	E	H	L	M
	BF	B8	B9	BA	BB	BC	BD	BE

Bild 6.4. b) arithmetische und logische Operationen des 8080/85

c)

Mnemonic	Bytes	Hex. Code	Flagbeeinflussung *	Taktzyklen	Funktion des Befehls	Taktzyklen 8085
4 REGISTERANWEISUNGEN						
4.1 Akkumulator rotieren						
RLC	1	07	C	4	Accu-Inh. wird zyklisch um 1 Bit nach links verschoben. Bit 2^7 wird in das Carry-Bit geschrieben	4
RRC	1	0F	C	4	Accu-Inh. wird zyklisch um 1 Bit nach rechts verschoben. Bit 2^0 wird in das Carry-Bit geschrieben	4
RAL	1	17	C	4	Accu-Inh. wird um 1 Bit nach links geschoben. Bit 2^7 wird in das Carry-Bit und das Carry-Bit in das Bit 2^0 geschrieb.	4
RAR	1	1F	C	4	Accu-Inh. wird um 1 Bit nach rechts geschoben. Bit 2^0 wird in das Carry-Bit und das Carry-Bit in das Bit 2^7 geschrieb.	4
4.2 Übertragsbit-Anweisungen						
CMC	1	3F	C	4	Carry-Bit wird negiert	4
STC	1	37	C	4	Carry-Bit wird gesetzt	4
5 SPRUNGBEFEHLE						
5.1 Unbedingte Sprünge						
PCHL	1	E9	--	5	Programm wird an der Adresse fortgesetzt, die im Reg.-Paar (H,L) steht	6
JMP adr	3	C3	--	10	Programm wird an der Adresse fortgesetzt	10
5.2 Bedingte Sprünge						
JC adr	3	DA	--	10	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	7/10
JNC adr	3	D2	--	10	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	7/10
JZ adr	3	CA	--	10	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	7/10
JNZ adr	3	C2	--	10	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	7/10
JM adr	3	FA	--	10	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	7/10
JP adr	3	F2	--	10	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	7/10
JPE adr	3	EA	--	10	Bei Paritäts-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	7/10
JPO adr	3	E2	--	10	Bei Paritäts-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	7/10
6 UNTERPROGRAMMBEHANDLUNG						
6.1 Programmaufrufe						
Bei allen Aufrufbefehlen wird die Rückkehradresse im Stapelzeiger abgelegt						
CALL adr	3	CD	--	17	Programm wird bei der Adresse adr fortgesetzt	18
CC adr	3	DC	--	11/17	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	9/18
CNC adr	3	D4	--	11/17	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	9/18
CZ adr	3	CC	--	11/17	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	9/18
CNZ adr	3	C4	--	11/17	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	9/18
CM adr	3	FC	--	11/17	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	9/18
CP adr	3	F4	--	11/17	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	9/18
CPE adr	3	EC	--	11/17	Bei Paritäts-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	9/18
CPO adr	3	E4	--	11/17	Bei Paritäts-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	9/18
RST konst	1	s.T.	--	11	$0 \leq$ konst ≤ 7 Programm wird auf der Adresse 8xkonst fortgesetzt	12

RST	0	1	2	3	4	5	6	7
	C7	CF	D7	DF	E7	EF	F7	FF

Bild 6.4. c) Registeranweisungen und Verzweigungsbefehle des 8080/85

d)

Mnemonic	Bytes	Hex. Code	Flagbeeinflussung *	Taktzyklen	Funktion des Befehls	Taktzyklen 8085
6.2 Rücksprungbefehle						
RET	1	C9	--	10	Programm wird an der Adresse fortgesetzt, die in dem Wort steht, das über den Stapelzeiger adressiert ist	10
RC	1	D8	--	5/11	Bei Carry-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RNC	1	D0	--	5/11	Bei Carry-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RZ	1	C8	--	5/11	Bei Zero-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RNZ	1	C0	--	5/11	Bei Zero-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RM	1	F8	--	5/11	Bei Sign-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RP	1	F0	--	5/11	Bei Sign-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RPE	1	E8	--	5/11	Bei Paritäts-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
RPO	1	E0	--	5/11	Bei Paritäts-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stapelzeiger adressierten Wort steht	6/12
7 PROGRAMMUNTERBRECHUNG						
EI	1	FB	--	4	INTE-Flipflop wird gesetzt, der Mikroprozessor kann Unterbrechung annehmen	4
DI	1	F3	--	4	INTE-Flipflop wird rückgesetzt, der Mikroprozessor ignoriert Unterbrechungsanforderungen	4
8 SONSTIGE BEFEHLE						
HLT	1	76	--	7	Programm hält an, bis eine Unterbrechungsanforderung eintrifft	5
NOP	1	00	--	4	Leerbefehl	4
9 ZUSÄTZLICHE 8085 BEFEHLE						
RIM	1	20	--	4	Lesen der Unterbrechungsmaske incl. SID	SID I7.5 I6.5 I5.5 IE M7.5 M6.5 M5.5
SIM	1	30	--	4	Setzen der Unterbrechungsmaske incl. SOD	SOD SOE X R7.5 MSE M7.5 M6.5 M5.5
SHLX	1	D9	--	10	Inh. (L)→Inh. $(M)_{DE}$, Inh. (H)→Inh. $(M)_{DE+1}$	
LHLX	1	ED	--	10	Inh. $(M)_{DE}$→Inh. (L) , Inh. $(M)_{DE+1}$→Inh. (H)	
LDHI	2	28	--	10	Inh. (D,E) = Inh. (h,L) + Inh. (Byte 2)	
LDSI	2	38	--	10	Inh. (D,E) = Inh. (SP) + Inh. (Byte 2)	
DSUB	1	08	Z,S,P,C,A,X5,V	10	Inh. (H,L) = Inh. (H,L) - Inh. (B,C)	
ARHL	1	10	C	7	Verschiebe Inh. (H,L) nach rechts, L_0→Carry, H_7 H_6	
RDEL	1	18	C,V	10	Verschiebe Inh. (D,E) nach links, D_7→Carry, Carry E_0	
JNX5	3	DD	--	7/10	Sprung, wenn das Flag X5 nicht gesetzt ist	
JX5	3	FD	--	7/10	Sprung, wenn das Flag X5 gesetzt ist	
RSTV	1	CB	--	5/12	Restart 40_H, wenn Überlauf (Flag V)	
ZUSÄTZLICHE FLAGS: V wird gesetzt beim Überlauf des Zweierkomplements X5 wird bei einem Überlauf des INX-Befehles und bei einem Unterlauf des DCX-Befehles gesetzt						

Bild 6.4. d) Rücksprung- und sonstige Befehle des 8080/85

Die Spalte 6 enthält eine Kurzbeschreibung des Befehls. Einige der Befehle werden im folgenden noch näher erklärt werden. Eine detaillierte Beschreibung aller Befehle findet man in den Datenbüchern der Hersteller.
Demjenigen Leser, der selbst Mikroprozessorprogramme erstellen möchte, wird dringend empfohlen, sich die Kurzbeschreibung aller Befehle in Bild 6.4 anzusehen. Nur wenn man einen Überblick über die Möglichkeiten gewinnt, die sich bei der Anwendung der verschiedenen Befehle eröffnen, ist ein effektives Programmieren möglich.

Programmliste
Mit Hilfe der Befehle aus Bild 6.4 können nun Mikroprozessor-Programme geschrieben werden.
Nachdem man sich über die Problemstellung klar geworden ist, kann man — evtl. unter Zuhilfenahme eines Flußdiagrammes (Abschnitt 4.4) — an das Erarbeiten eines Lösungsweges gehen. Wie schon erwähnt, ist es hierbei wichtig, eine genaue Vorstellung von den Möglichkeiten zu haben, welche die Befehle bieten.
Gleichgültig, ob die Befehlseingabe im Hex- oder Mnemonic-Code erfolgen soll, die Programmliste wird immer zunächst im Mnemonic-Code geschrieben. Das **Bild 6.5** zeigt ein Beispiel für solch eine Programmliste.

Adr.	Hex-Code	Mnemonic-Code	Kommentare
101C	E60F	ANI 0F	⟨A⟩ = ⟨A⟩ UND 0F
101E	FE0A	CPI 0A	⟨A⟩ = 0A? (⟨A⟩ - 0A)
1020	DA2610	JC 1026	Spring, wenn Carry = H
1023	C637	ADI 37	⟨A⟩ = ⟨A⟩ + 37 H
1025	C9	RET	Zurück ins Hauptprogramm (Abschnitt 6.2)
1026	C630	ADI 30	⟨A⟩ = ⟨A⟩ + 30 H
1028	C9	RET	return (Abschnitt 6.2)

Bild 6.5. Beispiel für eine Programmliste. Dieses Programm wurde als Unterprogramm (Abschnitt 6.2) geschrieben und dient dazu, zu einer HEX-Zahl im Akku den zugehörigen ASCII-Code zu ermitteln und in den Akku zu schreiben.

Zunächst schreibt man in die dritte Spalte den Mnemonic-Code eines Befehles und in die vierte Spalte eine kurze Erklärung über die Wirkung des Befehles. Die erste Spalte enthält immer die Adresse des ersten Bytes eines jeden Befehles.
Als Kommentare werden in Bild 6.5 die Befehlsfunktionen erklärt, um dem Anfänger das Lesen des Programmes zu erleichtern. Ein geübter Programmierer dagegen wird im Kommentarfeld einer Befehlszeile Erläuterungen über die Funktion des Befehles innerhalb dieses speziellen Programmes geben (Bild 6.11).
Nachdem auf diese Weise das gesamte Programm erstellt worden ist, können mit Hilfe der Befehlsliste aus Bild 6.4 die Hex-Codes der einzelnen Befehle ermittelt und in Spalte 2 eingetragen werden. Dieser Schritt ist jedoch nur erforderlich, wenn der

Mikrocomputer im Hex-Code programmiert werden soll. Andernfalls übersetzt der Mikrocomputer selbst den Mnemonic-Code in den Hex-Code.
Der Hex-Code — er wird auch häufig Objektcode genannt — belegt dann eine entsprechende Anzahl von hintereinanderliegenden Speicherplätzen.

Beispiel: Belegung eines Speichers durch den Hex-Code aus Bild 6.5

Adr.:	101C	101D	101E	101F	1020	1021	1022	1023	1024	1025	1026	1027	1028
Hex-Code:	E6	0F	FE	0A	DA	26	10	C6	37	C9	C6	30	C9

6.2 Unterprogramme

Beim Programmieren kommt es häufig vor, daß eine bestimmte Befehlsfolge an mehreren Stellen des Programmes benötigt wird. In solchen Fällen ist es nicht erforderlich, diese Befehlsfolge immer wieder hinzuschreiben. Sie wird vielmehr in einen gesonderten Speicherbereich — z. B. ans Ende des Programmes — geschrieben und dann, an den Stellen des Programmes, wo sie benötigt wird, durch einen sogenannten Unterprogrammaufruf ersetzt. Die ausgesonderte Befehlsfolge nennt man Unterprogramm und die rufende Hauptprogramm (**Bild 6.6**).

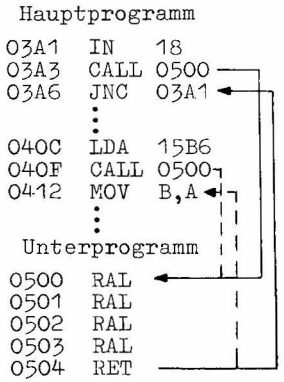

Bild 6.6. Zweimaliger Aufruf eines Unterprogramms

An der Stelle im Hauptprogramm, wo die Befehlsfolge des Unterprogrammes abgearbeitet werden soll, wird ein CALL-Befehl geschrieben. Dieser CALL-Befehl enthält im Adreßteil die Adresse des ersten Bytes des Unterprogrammes. Wenn nun der Mikroprozessor bei der Abarbeitung des Programms auf einen CALL-Befehl kommt, bewirkt dies einen Sprung zu der angegebenen Unterprogrammadresse. Bevor der Mikroprozessor diesen Sprung ausführt, merkt er sich jedoch noch die Adresse des nächsten Befehles hinter dem CALL-Befehl. Jedes Unterprogramm (UP) endet mit

einem RET-Befehl (return). Wenn der Mikroprozessor diesen Befehl erreicht, springt er zurück ins Hauptprogramm; und zwar zu derjenigen Adresse, die er sich beim Unterprogrammaufruf gemerkt hat, d. h. auf den unmittelbar hinter dem CALL stehenden Befehl. Die Wirkung dieses Verfahrens ist demnach so, als würde an der Stelle des CALL-Befehls das gesamte Unterprogramm — ohne den RET-Befehl — stehen.

Der Stapelzeiger

Beim Ausführen des ersten CALL-Befehles im Beispiel von **Bild 6.7** merkt sich der Mikroprozessor die Rücksprungadresse 03A6. Wo wird nun diese Adresse abgespeichert, damit der RET-Befehl auf sie zurückgreifen kann? Eine Möglichkeit besteht darin, spezielle Register hierfür vorzusehen. Der MP 8080/85 benutzt jedoch ein anderes Verfahren, das eine größere Flexibilität mit sich bringt.

Im Mikroprozessor 8080/85 gibt es ein spezielles 16-bit-Register, den sogenannten Stapelzeiger oder Stackpointer. Mit Hilfe dieses Stapelzeigers kann sich der Mikroprozessor Rücksprungadressen und auch Registerinhalte merken. Der Stapelzeiger selbst nimmt keine Rücksprungadresse auf, denn dann könnte er ja nur eine davon speichern. Im Stapelzeiger steht vielmehr eine RAM-Adresse (Arbeitsspeicher), von der ausgehend Rücksprungadressen abgespeichert werden können. Mehrere Rücksprungadressen sind immer dann erforderlich, wenn in einem Unterprogramm erneut ein CALL-Befehl steht, der ein weiteres Unterprogramm aufruft. Eine solche Unterprogrammverschachtelung kann durchaus über viele Stufen gehen. Doch hierzu später mehr.

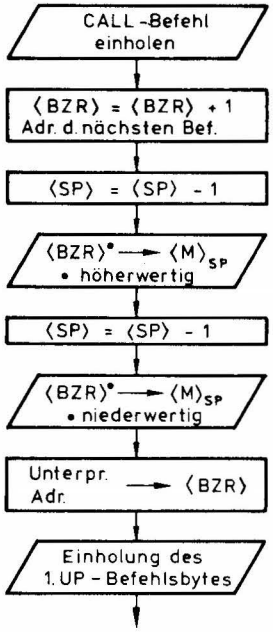

Bild 6.7. Abarbeitung eines CALL-Befehles (BZR = Befehlszählregister, SP = Stapelzeiger, M = Stapelspeicher)

Der Stapelzeiger kann in vielem so behandelt werden, wie ein Registerpaar. Es ist z. B. möglich, ihn mittels des LXI-Befehles mit einer 2-Byte-Konstanten zu laden. Diese Konstante hat dann die Bedeutung einer Adresse.

Die Adresse im Stapelzeiger bezeichnet das Ende eines Arbeitsspeicherbereiches, der auch Stapelspeicher (Stack) genannt wird.

Da eine Rücksprungadresse immer aus zwei Byte besteht, werden jeweils zwei Speicherplätze im Stapelspeicher für das Abspeichern einer Adresse benötigt. Hierzu erniedrigt der Mikroprozessor den Inhalt des Stapelzeigers um eins und speichert das höherwertige Byte ab, dann wird der Stapelzeiger erneut um eins erniedrigt und das niederwertige Byte unter dieser Adresse abgespeichert (Bild 6.7). Im Stapelspeicher (-Bereich) werden also die zu speichernden Daten zu niedrigeren Speicheradressen hin „aufgestapelt". Aus diesem Grunde wird in den Stapelzeiger zu Beginn eines Programmes meistens eine hohe — häufig auch die höchste — Adresse des Arbeitsspeichers eingeschrieben, damit zu niedrigeren Adressen hin Platz für den Stapel besteht.

Unterprogrammverschachtelung

Von einer Unterprogrammverschachtelung spricht man, wenn in einem Unterprogramm ein weiteres Unterprogramm aufgerufen wird.

Beim Aufruf des zweiten Unterprogrammes stapelt der Mikroprozessor die weitere Rücksprungadresse auf die erste.

In **Bild 6.8** wird sichtbar, daß der Stapelzeiger mit einer Adresse geladen werden kann, die um eins höher ist als die größte RAM-Adresse (Arbeitsspeicher), denn bevor über

Adr.	Befehl	⟨SP⟩*	RAM Adr.	RAM Inh.	Bemerkungen
0000	LXI SP,1100	1100			SP laden
⋮					
003C	CALL 024B	10FE	10FE	3F	Rücksprungadresse ins RAM
			10FF	00	laden und Sprung ins UP I
003F	XXXX	1100			
⋮					
	UP I				
024B	XXXX	10FE			
⋮			10FC	56	
0253	CALL 03A2	10FC	10FD	02	Sprung ins UP II
0256	XXXX	10FE	10FE	3F	
⋮			10FF	00	
	RET	1100			Rücksprung ins Hauptprogramm
	UP II				
03A2	XXXX	10FC			
⋮					
	RET	10FE			Rücksprung ins UP I

*Inhalt nach Ausführung des Befehles.

Bild 6.8. Beispiel für eine zweifache Unterprogrammverschachtelung; höchste RAM-Adresse = 10FF

den Stapelzeiger etwas abgespeichert wird, erniedrigt der Mikroprozessor den Inhalt um eins (Bild 6.7).
Aufgrund dieser Verfahrensweise zeigt der Stapelzeiger immer auf den Speicherplatz, in den zuletzt geschrieben wurde. Erst bei einem erneuten CALL-Befehl wird der Stapelzeiger dekrementiert und zeigt damit auf den nächsten freien Speicherplatz.
Bei der Ausführung eines RET-Befehles werden das Speicherbyte, auf welches der Stapelzeiger gerade zeigt, und dasjenige Byte, welches im nächsten Speicherplatz steht, zusammen als Rücksprungadresse gewertet. Bevor der Rücksprung ausgeführt wird, erhöht der Mikroprozessor den Stapelzeiger nochmals um eins, so daß er nun auf die evtl. vorhandene nächste Rücksprungadresse zeigt.

6.3 Programm-Entwicklungssysteme

In diesem Abschnitt wollen wir uns der Frage zuwenden, welche Hilfsmittel zur Entwicklung von Mikrocomputer-Programmen zur Verfügung stehen.
Wie **Bild 6.9** zeigt, kann hierzu prinzipiell jeder Computer herangezogen werden. Angefangen vom kleinsten System mit Hexadezimaltastatur, wie im Bild 6.3, bis hin zum Großrechner eines Rechenzentrums.
Ein wichtiger Unterschied besteht jedoch darin, ob das zu entwickelnde Programm in dem benutzten Computer ablauffähig ist oder nicht. Soll ein Programm, das mit Hilfe eines Computers entwickelt wird, in diesem gestartet werden können oder gar für ihn selbst bestimmt sein, so muß er den entsprechenden Mikroprozessor beinhalten. Wenn dagegen der zur Programmentwicklung benutzte Computer nicht über den

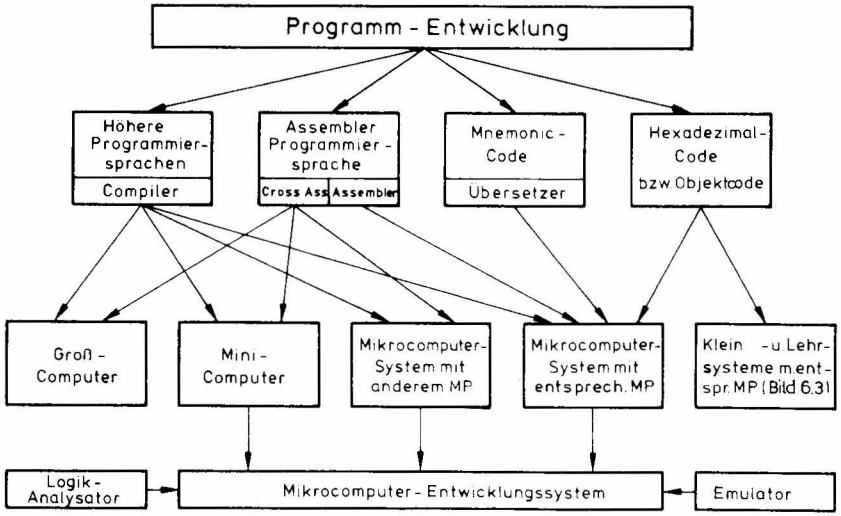

Bild 6.9. Möglichkeiten zur Entwicklung von Mikroprozessor-Programmen

gleichen Mikroprozessor verfügt, ist es erforderlich, das durch den Compiler oder den Cross Assembler übersetzte Programm — d. h. den Objektcode — über ein Speichermedium in den Mikrocomputer zu bringen, für den es geschrieben wurde.
Die Eingabe von Programmen im Hex-Code (Objektcode) und im Mnemonic-Code wurde bereits beschrieben.
Dieses Kapitel dagegen enthält Hinweise und Erklärungen zur Anwendung der Assembler-Programmiersprache und zur Benutzung von Hilfsprogrammen.
Der Einsatz höherer Programmiersprachen, bei denen z. B. ein in der Programmiersprache FORTRAN 80 geschriebenes Programm von einem speziellen Compiler in den Objektcode des MP 8080 übersetzt wird, soll hier nicht weiter behandelt werden.
Dem Vorteil der besseren Übersichtlichkeit beim Programmieren steht ein wesentlich längeres Objektcode-Programm entgegen — im Vergleich zum Fall der Programmierung des gleichen Problems in der Maschinensprache (Assembler). Hierdurch werden erheblich mehr Speicherplätze belegt und die Programmausführungszeiten drastisch verlängert.
Ein Programm-Entwicklungssystem heißt Mikrocomputer-Entwicklungssystem, wenn es durch verschiedene elektronische Testhilfen ergänzt wird. Zwei der wichtigsten Ergänzungen sind der Emulator (in-circuit-emulator) und der Logik-Analysator. Mit Hilfe dieser Testgeräte kann der neu zu entwickelnde Mikrocomputer an das Entwicklungssystem angeschlossen werden. Dies erlaubt umfangreiche Tests der Programme und der zu entwickelnden elektronischen Schaltung.
Die Entwicklungssysteme wurden früher ausschließlich von den Herstellern der Mikroprozessoren angeboten und enthielten den gleichen Mikroprozessor. Dies hat den Vorteil, daß die zu entwickelnden Programme direkt auf dem Entwicklungssystem ablauffähig sind und getestet werden können. Des weiteren ist der qualifizierte Anwender in der Lage, sich Programme für sein Entwicklungssystem zu schreiben, die seinen speziellen Bedürfnissen gerecht werden.
In letzter Zeit werden jedoch auch verstärkt Entwicklungssysteme von Meßgeräte-Herstellern angeboten. Diese Systeme bestehen im allgemeinen aus Minicomputern, und es stehen Cross-Assembler für alle gängigen Mikroprozessoren zur Verfügung. Ein wesentlicher Vorteil dieser Entwicklungssysteme ist zur Zeit vor allem in ihrer leichteren Bedienbarkeit zu sehen.
Die Entwicklung von Programmen läßt sich jedoch auch mit Hilfe sogenannter Personal-Computer durchführen. Ein Personal-Computer besteht aus einem Mikrocomputer, der, mit Tastatur und Bildschirm versehen, als Tischrechner Anwendung findet.
Wenn ein solcher Personal-Computer über einen Massenspeicher verfügt und dann noch durch einen Drucker und ein Speicherprogrammiergerät ergänzt wird, erhält man ein leistungsfähiges und äußerst preiswertes Programm-Entwicklungssystem.

Der Editor

Der Editor — oder auch Text-Editor — stellt ein spezielles Hilfsprogramm zum Einschreiben von Texten in den Anwenderspeicher des Computers dar.
Zunächst wird das Editor-Programm mit Hilfe des Monitors (residentes Grundpro-

gramm des Computers) vom Massenspeicher in den Anwenderspeicher gebracht und gestartet.
Danach übernimmt der Editor — d. h. das gestartete Programm — die Kontrolle über den Computer. Nun kann der Anwender die Befehlszeilen eines Programmes in den Speicher schreiben. Dies nennt man „Editieren eines Quellprogrammes" (s. a. Bild 6.11).
Damit später ein Assembler(-Programm) diese Befehlszeilen in gültige Mikroprozessor-Befehlsbytes übersetzen kann, müssen bei der Erstellung der Befehlszeilen bestimmte Konventionen beachtet werden. Die Vereinbarungen sind Bestandteil der Assembler-Programmiersprache.

Die Assemblersprache
Eine Assemblersprache bedient sich der mnemonischen Abkürzungen, wie sie bereits in Abschnitt 6.1 beschrieben wurden. Darüber hinaus bietet sie eine Reihe weiterer Erleichterungen beim Programmieren. Insbesondere können symbolische Marken und Namen benutzt werden. Dies hat zur Folge, daß beispielweise bei einem Sprungbefehl als Zieladresse ein Name angegeben wird, der an anderer Stelle, nämlich dem Sprungziel, als Marke vor dem entsprechenden Befehl steht.

Beispiel:

```
        JMP   ANTON

ANTON:  MVI   A, 0
```

Auf diese Weise ist es nicht erforderlich, Sprungadressen zu ändern, wenn Befehle eingeschoben werden. Beim Assemblieren (Übersetzen) fügt der Assembler (Übersetzer-Programm) die richtigen (absoluten) Adressen in den Hex-Code ein.

Aufbau einer Befehlszeile
Damit der Assembler eine Befehlszeile richtig übersetzen kann, müssen bestimmte Vereinbarungen oder Festlegungen getroffen werden. Eine Befehlszeile besteht maximal aus vier Feldern. Hiervon sind uns aus dem Vorangegangenen die letzten drei

Marke: oder Name	Operationscode bzw. Mnemoniccode	Operand oder Operanden	;Kommentar
	LXI	SP,1100H	;Hexadezimaldaten
ANFA:	MVI	B,55	;Dezimaldaten
	ANI	10111001B	;Binaerdaten
	LXI	H,PUFFER	;Datenmarke -> <HL>
	CPI	'*'	;Vergleich den Akku-Inhalt ;mit dem ASCII-Code von *
	JMP	ANFA	;Sprung zur Befehlsmarke
PUFFER:	NOP		

Bild 6.10. Aufbau von Befehlszeilen und Operanden-Strukturen

Felder bereits bekannt. Es kommt lediglich ganz links noch ein Feld für Marken oder Namen hinzu.
Einige Beispiele für den Aufbau von Befehlszeilen enthält **Bild 6.10**.Insbesondere im Operandenfeld wurden die wichtigsten Formen der Festlegung von Daten oder Adressen dargestellt.
Wichtig ist, daß die unterschiedlichen Angaben in einer Befehlszeile durch sogenannte Begrenzer getrennt werden.

Beispiele:
Leerzeichen (blank): Feldtrenner (JMP ANFA);
, : Trennt Operanden (MOV A, B);
' ' : Begrenzt ASCII-Zeichen (CPI 'A');
; : Beginn des Kommentarfeldes;
: : Begrenzt Marken (ANFA : XCHG).

Assembleranweisungen
Assembleranweisungen dienen zur Steuerung des Übersetzungsvorganges (Assemblieren) und werden wie Befehle in ein Programm eingebaut.
Bild 6.11 zeigt ein Beispiel für ein Quellprogramm. Es wurden das Unterprogramm aus Bild 6.5 benutzt und dem Hauptprogramm noch einige Assembleranweisungen hinzugefügt.

```
;Befehlsadresse (Adresse des ), Befehlsbytes)
;Objektcode (Hex-code)

;Speicherinhalt auf den Bildschirm bringen

        ORG   1000H     ;Assembleranweisung: Das erste Programm-
                        ;byte soll auf der Adresse 1000 stehen.
        LXI   H,1800H   ;Anfangsadresse des Speichers
BEGIN:  MOV   A,M       ;Speicherbyte -) Akku
        RRC             ;Höherwertige Tetrade des Akkus in
        RRC             ;die niederwertige bringen
        RRC             ;((A) 4 mal nach rechts rotieren)
        RRC
        CALL  HEXAS     ;Wandle HEX -) ASCII
        CALL  BILD      ;Linkes HEX-Zeichen -) Bildschirm
        MOV   A,M       ;Speicherbyte -) Akku
        CALL  HEXAS     ;Wandle HEX -) ASCII
        CALL  BILD      ;Rechtes HEX-Zeichen -) Bildschirm
        INR   L         ;Speicheradresse + 1
        JNZ   BEGIN     ;Nächstes Byte anzeigen
        JMP   RAUS      ;Rücksprung ins Betriebsprogramm
;U N T E R P R O G R A M M
;Dem HEX-Zeichen in der niederwertigen Tetrade des Akkus
;den ASCII-Code zuordnen -) Akku
HEXAS:  ANI   0FH       ;Höherwertige Tetrade durch UND-Ver-
                        ;knüpfung zu Null machen
        CPI   0AH       ;Ist das HEX-Zeichen ein Buchstabe?
        JC    ZAHL      ;Sprung, wenn Zahl
        ADI   37H       ;(A) + 37 = ASCII-Code der Buchstaben
        RET             ;Zurück ins Hauptprogramm
ZAHL:   ADI   30H       ;(A) + 30 = ASCII-Code der Zahlen
        RET             ;Zurück ins Hauptprogramm

BILD    EQU   1029H     ;Assembleranweisung: Weis der symbo-
                        ;lischen Adresse 'BILD' den Wert 1029 zu.
RAUS    EQU   5BH

        END             ;Assembleranweisung: Ende des Programms
```

Bild 6.11. Beispiel für ein Programm (s. a. Bild 6.5), das mit Hilfe eines Editors für einen Assembler geschrieben wurde

```
                        ;Befehlsadresse (Adresse des 1. Befehlsbytes)
                        ;Objektcode (Hex-code)

                        ;Speicherinhalt auf den Bildschirm bringen

0000                    ORG  1000H     ;Assembleranweisung: Das erste Programm-
                                       ;byte soll auf der Adresse 1000 stehen.
1000 211018             LXI  H,1800H   ;Anfangsadresse des Speichers
1003 7E         BEGIN:  MOV  A,M       ;Speicherbyte -) Akku
1004 0F                 RRC            ;Höherwertige Tetrade des Akkus in
1005 0F                 RRC            ;die niederwertige bringen
1006 0F                 RRC            ;((A) 4 mal nach rechts rotieren)
1007 0F                 RRC
1008 CD1C10             CALL HEXAS     ;Wandle HEX -) ASCII
100B CD2910             CALL BILD      ;Linkes HEX-Zeichen -) Bildschirm
100E 7E                 MOV  A,M       ;Speicherbyte -) Akku
100F CD1C10             CALL HEXAS     ;Wandle HEX -) ASCII
1012 CD2910             CALL BILD      ;Rechtes HEX-Zeichen -) Bildschirm
1015 2C                 INR  L         ;Speicheradresse + 1
1016 C20310             JNZ  BEGIN     ;Nächstes Byte anzeigen
1019 C35B00             JMP  RAUS      ;Rücksprung ins Betriebsprogramm
                    ;U N T E R P R O G R A M M
                    ;Dem HEX-Zeichen in der niederwertigen Tetrade des Akkus
                    ;den ASCII-Code zuordnen -) Akku
101C E60F       HEXAS: ANI  0FH        ;Höherwertige Tetrade durch UND-Ver-
                                       ;knüpfung zu Null machen
101E FE0A               CPI  0AH       ;Ist das HEX-Zeichen ein Buchstabe?
1020 DA2610             JC   ZAHL      ;Sprung, wenn Zahl
1023 C637               ADI  37H       ;(A) + 37 = ASCII-Code der Buchstaben
1025 C9                 RET            ;Zurück ins Hauptprogramm
1026 C630       ZAHL:   ADI  30H       ;(A) + 30 = ASCII-Code der Zahlen
1028 C9                 RET            ;Zurück ins Hauptprogramm

1029            BILD    EQU  1029H     ;Assembleranweisung: Weis der symbo-
                                       ;lischen Adresse 'BILD' den Wert 1029 zu.
005B            RAUS    EQU  5BH

0000                    END            ;Assembleranweisung: Ende des Programms



                    SYMBOL TABLE

BEGIN   1003    BILD    1029    HEXAS   101C    RAUS    005B
ZAHL    1026
```

Bild 6.12. Vom Assembler erstellte Programmliste (listing) auf der Basis des Programms aus Bild 6.11

Der Assembler

Das Assembler-Programm hat die Aufgabe, aus einer Befehlszeile des Quellprogrammes den entsprechenden Objektcode zu erzeugen. Hierbei dienen die Assembler-Anweisungen zur Steuerung des Übersetzungsvorganges.

Die Übersetzung geschieht meistens in zwei Schritten. Zunächst wertet der Assembler die Anweisungen, die Operanden und die Marken aus. Hierbei wird insbesondere eine sogenannte Symbol-Tabelle erstellt, die eine Zuordnung der absoluten Adressen zu den symbolischen Namen enthält (**Bild 6.12**).

Im zweiten Durchgang werden dann die mnemonischen Abkürzungen in den entsprechenden Binär-Code umgewandelt. Der hierbei entstehende Objektcode (Hex-Code) wird im Anwenderspeicher des Entwicklungssystems abgelegt.

7 Elektronischer Aufbau eines Mikrocomputers

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Bereits im Kapitel 5 wurden einige wichtige Komponenten von Mikrocomputern kurz beschrieben.

In diesem Kapitel folgen grundlegende Aspekte im Hinblick auf den elektronischen Aufbau von Mikrocomputern (siehe auch [1]).

7.1 Der Mikroprozessor

Die meisten heute erhältlichen Mikroprozessoren haben einen 8-bit-Datenbus und werden deshalb auch 8-bit-Mikroprozessoren genannt.

Seit 1971 wurden viele verschiedene Mikroprozessoren entwickelt. Insbesondere Verbesserungen in der Herstellungstechnologie führten zu immer komplexeren Systemen (**Bild 7.1**).

Leider verfügen fast alle Mikroprozessoren über einen anderen Befehlssatz und eine differierende Anschlußbelegung, so daß sie untereinander nicht kompatibel sind.

Typenbezeichnung	Minimale Befehlszyklus-zeit/µs	Menge der Grundbefehle	Menge der Allzweckregister	Lieferbar seit	Hersteller
CDP1802	2,5	91	16x16bit	1977	RCA
2650AN	1,5-2,4	75	7	1977	Valvo-Signetics
MD46802	0,8	72	2	1980	Mitel
6500	1-2	56	1	1976	Rockwell
6800	0,5-1	72	2	1975	Motorola
6802	1	72	2	1976	"
6809	1	59	2-3	1980	"
F8-3850	2	67	65	1974	Fairchild
Z80	1-1,6	158	14	1977	Zilog
8008	12,5-20	48	6	1971	Intel
8080	1,5-2	72	7	1974	"
8085	0,8-1,3	74	7	1977	"
8088	0,4	105	8	1979	"
SC/MP	2	46	2	1975	National Semic.
NSC800	1	158	14	1980	"
TMS9980	0,4	69	Register im RAM	1978	Texas Instruments

Bild 7.1. Gegenüberstellung einiger 8-bit-Mikroprozessoren. Die meisten werden in NMOS-Technik angefertigt; lediglich der 8008 wurde in PMOS-Technik hergestellt, dagegen werden der CDP1802, der MD46802 (kompatibel mit dem 6802) und der NSC800 in CMOS-Technik gefertigt. Eine innere 16-bit-Struktur besitzen die Mikroprozessoren MC6809, 8088, TM99980.

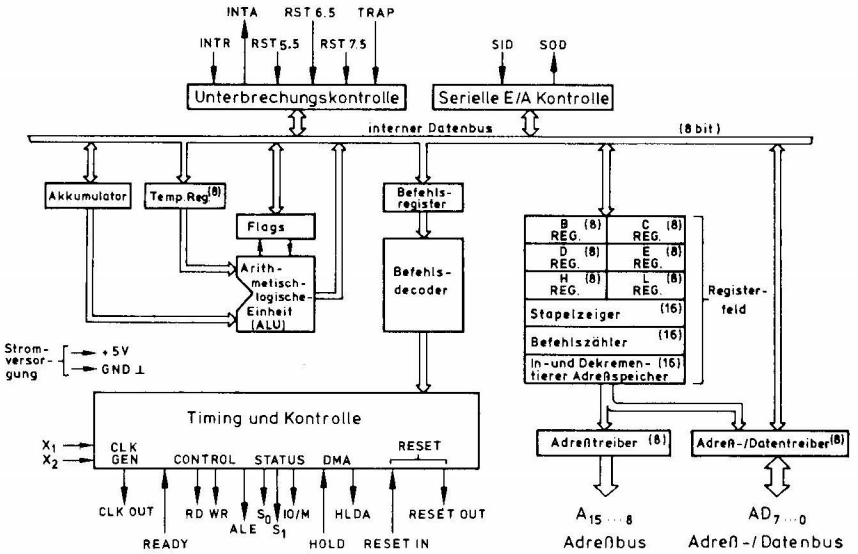

Bild 7.2. Innerer Aufbau des MP8085. Die Ein- und Ausgangssignale werden später beschrieben.

Auch im folgenden wird deshalb — wie bisher — der Mikroprozessor 8080 bzw. 8085 exemplarisch behandelt.
Während der MP 8080 noch in der alten NMOS-Technik gefertigt wird und deshalb drei Versorgungsspannungen (+12V, +5V, —5V) benötigt, erfolgt die Herstellung des MP 8085 in der neueren (depletion-load-) NMOS-Technik mit nur +5V Versorgungsspannung (**Bild 7.2**).

Die Fortschritte in der Herstellungstechnologie führten zu weiteren Vorteilen des MP 8085 gegenüber dem MP 8080:

- Der MP 8085 ist um 50 % schneller (3 MHz) als der MP 8080,
- hat zusätzlich einen seriellen Eingang (SID) und einen seriellen Ausgang (SOD),
- besitzt vier weitere Unterbrechungseingänge (TRAP, RST5.5, RST6.5, RST7.5),
- benötigt keinen Taktgeber — und keinen Systemsteuerbaustein,
- hat darüber hinaus aber den gleichen Befehlssatz wie der MP 8080 und einige Befehle mehr.

Da der MP 8085 auch nur 40 Anschlußbeinchen (pins) wie der MP 8080 hat, können nicht alle Ein-/Ausgabekanäle gleichzeitig nach außen geführt werden. Dies führte zur Anwendung des „Zeitmultiplex-Verfahrens" in Bezug auf den Datenbus und den niederwertigen Teil des Adreßbusses. Auf acht Pins des MP 8085 erscheint deshalb kurzzeitig das niederwertige Adreßbyte, und dann erst stehen sie für den Datenbus zur Verfügung.

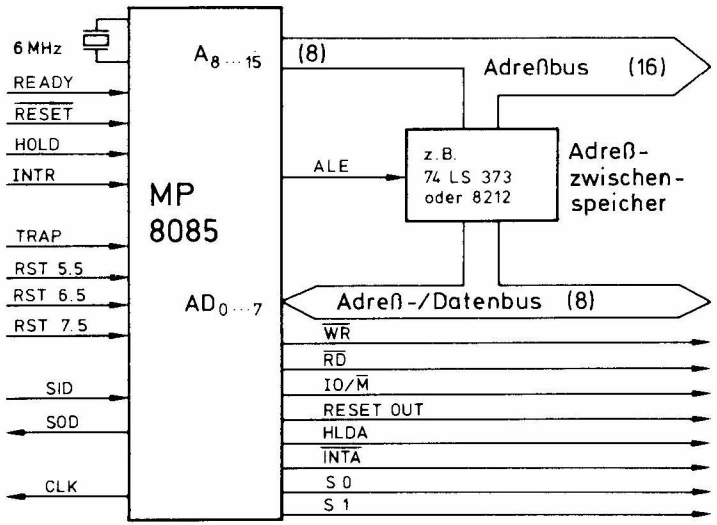

Bild 7.3. Grundverschaltung des MP8085. Der untere Teil des Adreßbusses wird durch das Demultiplexen des AD-Busses gewonnen. Wenn über der, den Kanal kennzeichnenden Buchstabenkombination ein Querstrich steht, so bedeutet dies "low aktiv", d. h., ein L-Signal charakterisiert diesen Zustand.

Die im **Bild 7.3** angegebenen Signale bedeuten im einzelnen:

READY: H → MP arbeitet
L → MP bleibt nach der Aussendung einer Adresse stehen.
RESET: L → MP startet bei Adresse 0000
RESET OUT: H → Reset Quittung
ALE: Signal zu Beginn eines neuen Maschinenzyklus (Adreßübernahme).
HOLD: H → MP anhalten, die Busse in den hochohmigen Zustand bringen und auf HLDA ein H ausgeben (Quittierung).
INTR: H → Unterbrechungsanforderung (interrupt)
INTA: L → Unterbrechungsquittung
TRAP, RST5.5, RST6.5, RST7.5: H → Unterbrechungsanforderungen
SID: Serieller Daten-Eingang
SOD: Serieller Daten-Ausgang
CLK: Mikroprozessor-Taktausgang (3 MHz)
S0: Statussignal 0
S1: Statussignal 1
RD: L → Lesen (read), der Datenbus ist auf Eingabe geschaltet.
WR: L → Schreiben (write), der Datenbus ist auf Ausgabe geschaltet.
IO/M: H → Ein-/Ausgabe (in/out) Operation. Zugriff zu peripheren Einheiten.
L → Speicherzugriff (memory).

Der Baustein 74LS373 im Bild 7.3 dient zur Zwischenspeicherung des niederwertigen Adreßbytes, das zu Beginn eines jeden Maschinenzyklus (Bild 7.4) auf dem Adreß/

Datenbus kurz ausgesandt wird. Auf diese Weise erhält man ein komplettes Bussystem bestehend aus 16 Adreßleitungen, 8 Datenleitungen und einigen Steuerleitungen (Steuerbus).
Die durch den Quarz erzeugte Frequenz (typisch 6 MHz) wird im Mikroprozessor halbiert und ergibt damit den internen Takt, der gleichzeitig auf dem Ausgang CLK (clock) verfügbar ist.

Signalverläufe am MP8085 (Timing)
Die beschriebenen Ein-/Ausgabekanäle des MP 8085 können immer nur einen der beiden Zustände L oder H annehmen, dies jedoch zu den unterschiedlichsten Zeitpunkten. Eine Gegenüberstellung der zeitlichen Verläufe von L- und H-Zuständen mehrerer Kanäle nennt man Timing-Diagramm.
Es dürfte klar sein, daß die Signale auf den 38 Kanälen (40-2 für +5V/Masse) eines Mikroprozessors recht komplex sein können und der Mikrocomputer nur richtig arbeitet, wenn die angeschlossenen Bausteine das Timing des Mikroprozessors berücksichtigen. Deshalb sollen hier einige wichtige und grundlegende zeitliche Abläufe kurz skizziert werden.
Im **Bild 7.4** ist als Beispiel der Signalverlauf für einen OUT-Befehl dargestellt. Für den gesamten Ablauf werden drei Maschinenzyklen (M1, M2, M3) und 10 Takte benötigt. Diese 10 Takte entsprechen dem in der Befehlsliste angegebenen Wert. Die Dauer einer Taktperiode T errechnet sich aus der Quarzfrequenz f wie folgt:

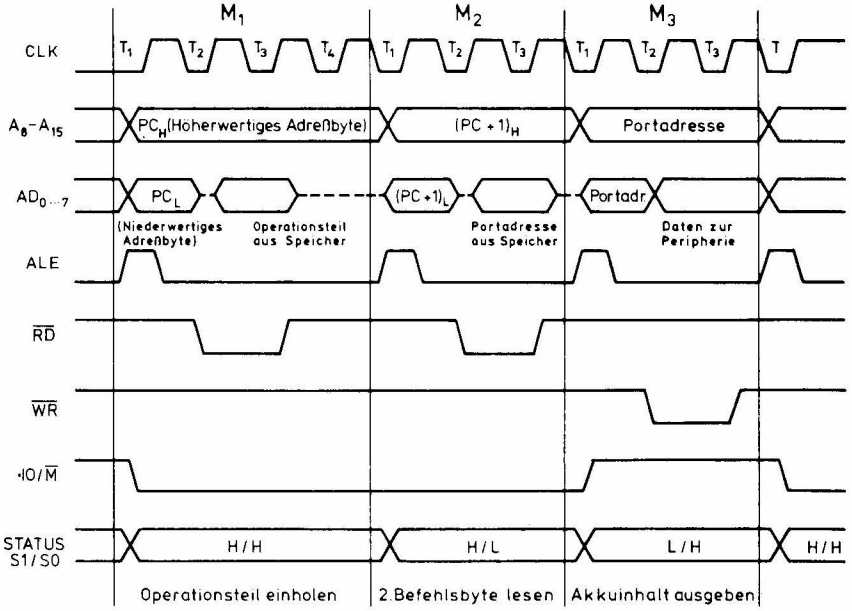

Bild 7.4. Grundlegendes Timing des MP8085 für den Befehl OUT

Beispiel:
$T = 2/f$.
$T = 2/6\ \text{MHz} = 0{,}33 \cdot 10^{-6}$ s.

Ein Maschinenzyklus beginnt immer mit der Aussendung einer Adresse und ist normalerweise drei Takte lang, nur M1 besteht meistens aus vier oder sechs Takten. Ein gesamter Befehlszyklus kann aus bis zu fünf Maschinenzyklen und maximal insgesamt 18 Takten (CALL-Befehl) bestehen. Es sind insgesamt sieben verschiedene Arten von Maschinenzyklen möglich. Welcher Zyklus gerade abläuft, läßt sich anhand der Status- und Kontrollsignale (IO/M, S1, S0, RD, WR, INTA) feststellen.

Der erste Maschinenzyklus dient immer zum Holen des Operationsteiles eines Befehles aus dem Speicher (opcode fetch). Hierzu wird der Inhalt des Befehlszählers (PC = program counter) auf den Adreßbus gebracht; das höherwertige Byte auf den oberen Halbbus ($A_{8\ldots15}$) und das niederwertige Byte auf den Adreß-Datenbus ($AD_{0\ldots7}$). Das ALE-Signal sorgt dann dafür, daß das niederwertige Byte außerhalb des Mikroprozessors zwischengespeichert wird. Danach erfolgt eine Umschaltung des AD-Busses auf Dateneingabe und die Aussendung eines Lesesignales (RD = L). Da der Ausgang IO/M ebenfalls im L-Zustand ist, sendet der über die Adresse angesprochene Speicherplatz den Operationscode des OUT-Befehles (D3) an den MP 8085 (Bild 7.4).

Die Interpretation des Operationsteiles (während T4) zeigt dem Mikroprozessor, daß noch ein weiteres Befehlsbyte, nämlich die Portadresse, eingeholt werden muß. Dies geschieht unter Erhöhung des Befehlszählers um eins im zweiten Maschinenzyklus.

Im letzten Maschinenzyklus schaltet dann der Ausgang IO/M auf H (Peripheriezugriff) und der Schreibausgang (WR) auf L, und es wird der Inhalt des Akkumulators auf den Datenbus gebracht. Die 1-Byte-Portadresse wird dabei als höher- und niederwertiges Adreßbyte ausgegeben.

7.2 Die Speicher

Der Speicher eines Mikrocomputers soll digitale Informationen aufnehmen und behalten, d. h., es handelt sich hierbei ausschließlich um digitale Speicher. Die Größe und Schnelligkeit der Speicher sind wesentliche Kriterien für die Leistungsfähigkeit eines Mikrocomputers.
Doch zunächst ist es erforderlich, einige wichtige Begriffe zu klären:

Speicherkapazität:
Menge der Bits oder Bytes, die ein Speicher aufnehmen kann, z. B. 1 kB = 1024 x 8 bit.

Zugriffszeit:
Zeit, die zwischen der Anforderung und der Verfügbarkeit eines Speicherplatzinhaltes vergeht.

Einschreiben:
Einbringen von Informationen in eine Speicherzelle unter Löschung der alten Information.

Lesen:
Entnahme eines Speicherplatzinhaltes ohne Löschung.

Wahlfreier Zugriff:
Alle Speicherplätze sind in beliebiger, ungebundener Reihenfolge erreichbar (random access), z. B. RAM, ROM.

Reihenfolge Zugriff:
Z. B. Bindung an die Reihenfolge, in der die Daten eingeschrieben wurden (Magnetband).

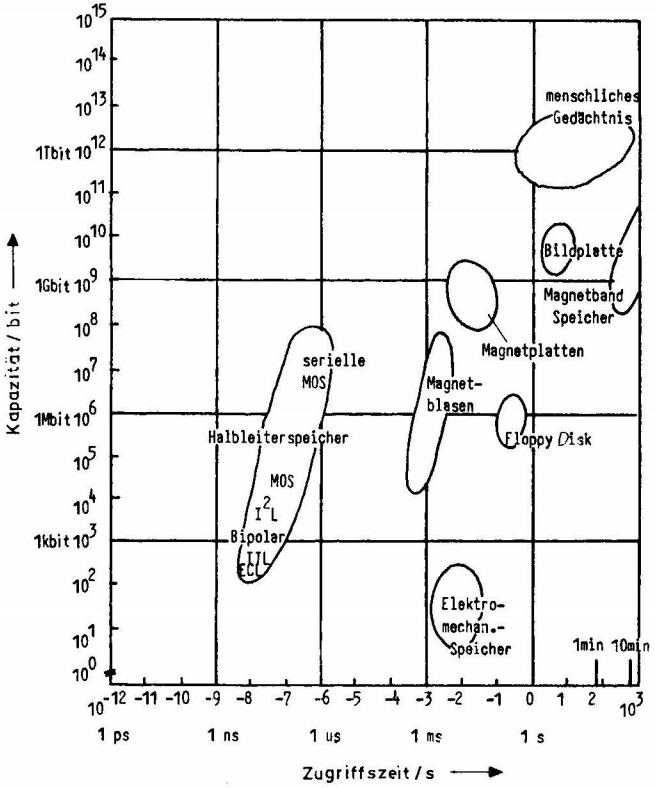

Bild 7.5. Vergleich unterschiedlicher Speichermedien

Permanenz:
Nach einem Energie-(Strom-)Ausfall bleibt die Information entweder erhalten (permanente Speicher: ROM, Magnetspeicher) oder ist gelöscht (temporäre Speicher: RAM, Register).

Anstelle der Begriffe permanenter oder temporärer Speicher werden häufig auch die Begriffe nichtflüchtiger- (nonvolatile) oder flüchtiger Speicher benutzt.
Ein Vergleich verschiedener Speichermedien (**Bild 7.5**) zeigt, daß die Kapazität und die Zugriffszeit von Speichern in einem viele Zehnerpotenzen überstreichenden Bereich variieren kann.

Speicherorganisation
Die internen Halbleiterspeicher eines Mikrocomputers besitzen einen wahlfreien Zugriff, d.h. jeder einzelne Speicherplatz ist direkt adressierbar. Zur Bezeichnung der Größe und der Organisation eines solchen Speichers werden meistens die Anzahl der Speicherplätze und die Anzahl der Zellen pro Speicherplatz angegeben. Wobei die Anzahl der pro Speicherplatz parallel verarbeitbaren Bits (Speicherzellen) auch als Wortlänge bezeichnet wird.

Beispiele:
256 × 1: 256 Speicherplätze mit jeweils einer Zelle.
1024 × 4: 1024 Speicherplätze mit einer Wortlänge von 4 bit.
2048 × 8: 2-kB-Speicher mit einer Kapazität von 16 384 bit = 16 kbit.

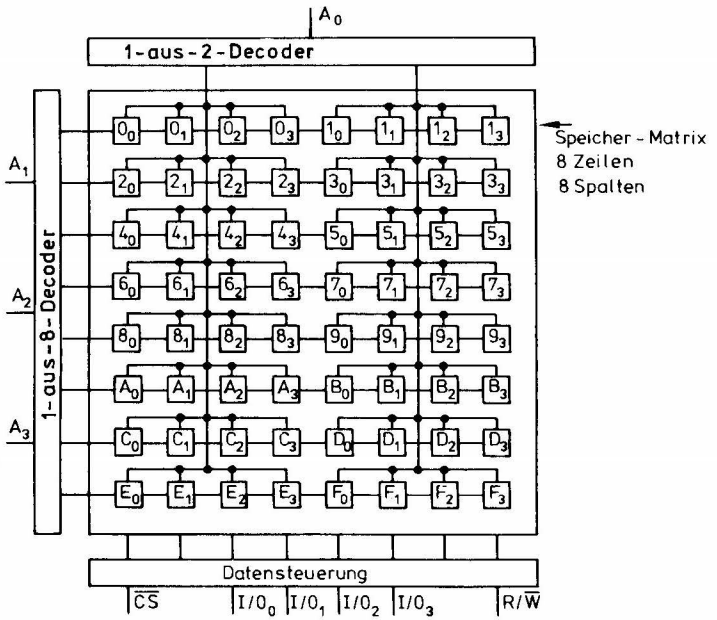

Bild 7.6. Prinzipieller Aufbau eines 16 × 4 Schreib-/Lesespeichers mit quadratischer Matrix

Ein Speicher, der 16 Plätze mit einer Wortlänge von 4 bit haben soll und damit 64 Zellen benötigt, kann als quadratische Matrix mit der Organisation 8 x 8 hergestellt werden. Wie **Bild 7.6** zeigt, sind dann jeweils vier Zellen einer Zeile zu einem Speicherplatz zusammenzufassen. Insbesondere die großen Speicher (1 bis 2 kB) haben häufig diese Organisation.

Für die Spaltenauswahl sind entsprechend weniger Adreßleitungen als für die Zeilenauswahl erforderlich, da jeweils eine der Wortlänge entsprechende Anzahl von Zellen in einer Zeile gleichzeitig angesprochen und mit den I/O-Leitungen verbunden wird.

Beispiel:
Bei 4 bit Wortlänge, wie in Bild 7.6, werden für die Spaltenauswahl zwei Adreßleitungen weniger benötigt, da zur Darstellung von vier Adressen 2 bit ($4 = 2^2$) erforderlich sind.

Der Eingang CS dient zur Steuerung der Tristate-Gatter (Abschnitt 3.3) in den Ein-/Ausgabeleitungen. Hierdurch ist ein dirkter Anschluß an die Busse des Mikrocomputers möglich.

7.2.1 Schreib-/Lesespeicher (RAM)

RAM = Random Access Memory = Schreib-/Lesespeicher mit wahlfreiem Zugriff. Wenn die Versorgungsspannung weggenommen wird, geht der Speicherinhalt verloren. RAMs finden vor allem Anwendung als Arbeitsspeicher.

Statische Speicher
Bei statischen Speichern werden die Speicherelemente durch zwei rückgekoppelte

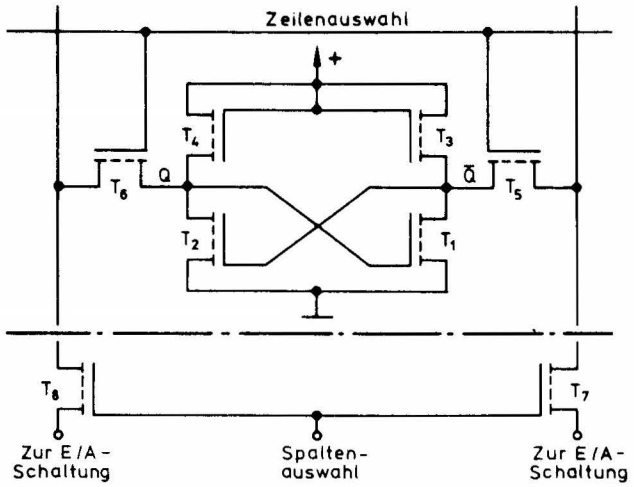

Bild 7.7. Statische MOS-Speicherzelle mit Zeilen- und Spaltenauswahl

Inverter (bistabile Kippstufe, Flipflop) gebildet, wobei den beiden stabilen Zuständen jeweils die logische Information L oder H zugeordnet wird. In der Prinzipschaltung von **Bild 7.7** werden die beiden Lastwiderstände wegen eines geringeren Platzbedarfs durch zwei Transistoren T3 und T4 mit fester Vorspannung ersetzt.

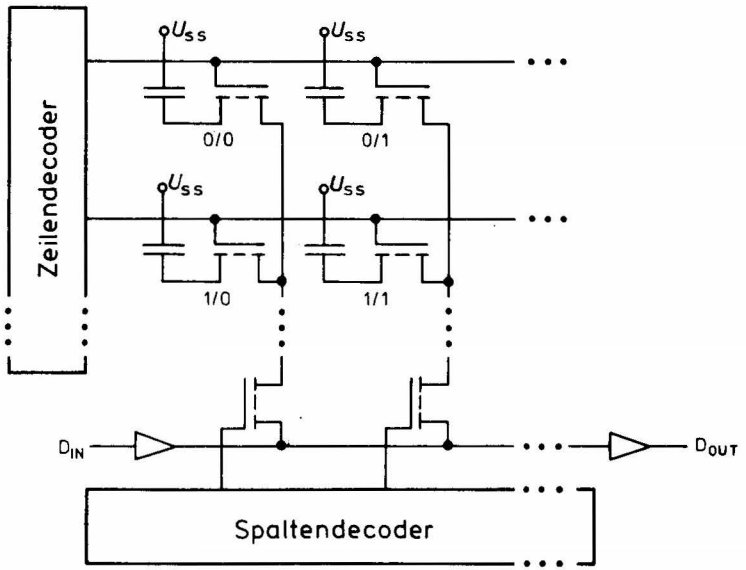

Bild 7.8. Bit-orientierter dynamischer Speicher mit 1-Transistor-Speicherzelle

Dynamische RAM-Speicher
Bei dynamischen Speicherelementen wird die Information als Ladung in einem Kondensator gespeichert. Das in **Bild 7.8** gezeigte Prinzipschaltbild stellt eine sogenannte 1-Transistor-Zelle dar, bei der die Information abhängig von der Kondensatorladung als Potentialsprung auf der Datenleitung erscheint, sobald der Transistor T1 über die Zeilenauswahlleitung in den leitenden Zustand versetzt wird. Wegen der unvermeidbaren Leckströme muß die Speicherinformation (Kondensatorladung) periodisch regeneriert werden (refresh).

7.2.2 Festwertspeicher (ROM)

ROM = Read Only Memory = Festwertspeicher mit wahlfreiem Lesezugriff.
Der Speicherinhalt wird bereits bei der Herstellung eingeschrieben und kann nicht mehr verändert, sondern nur ausgelesen werden. Beim Ausfall der Betriebsspannung bleibt der Inhalt erhalten.
Der Aufbau einer Speicherzelle ist ähnlich wie bei einem RAM, nur wird hier ein Transistor dauernd leitend oder nichtleitend geschaltet. Diese Verschaltung erfolgt bei der Herstellung durch eine Programmiermaske. Für jedes zu setzende Bit bleibt in der

Maske ein Fenster offen. Hierdurch wird mit Hilfe des üblichen Fotoverfahrens das Gate des entsprechenden Transistors mit der Betriebsspannung verbunden und erzeugt somit dauernd ein H-Signal.

Programmierbare Festwertspeicher (PROM)
Ein PROM ist ein ROM, das vom Anwender (einmal) programmiert werden kann. Hierzu benötigt man ein PROM-Programmiergerät.
Die Speicherzelle ist ähnlich aufgebaut wie die eines MOS-ROMs, nur enthält sie zusätzlich ein Sicherungselement (Sicherung, fuse), das von außen — durch einen starken Strom — zerstörbar ist. Bei der Programmierung werden durch Anlegen einer höheren Spannung gezielt Sicherungen durchgebrannt oder nicht; dadurch wird der Speicher unwiderruflich programmiert.

Lösch- und programmierbare Festwertspeicher (EPROM)
EPROM = Erasable Programmable Read Only Memory
Die Speicherzelle enthält einen MOS-Transistor mit einem aus zwei Schichten bestehenden Gate. Dieses Gate stellt einen Kondensator dar, der aufgrund seiner guten Isolierung eine durch Injektion von Elektronen hervorgerufene Aufladung über lange Zeit behält (etwa 10 Jahre). Im ungeladenen Zustand hat die Zelle die Information H und im geladenen Zustand L. Das „Laden" erfolgt durch eine höhere Spannung (etwa 25 V) mittels eines Programmiergerätes (Bild 8.1).
Zum Löschen eines programmierten Speichers wird dieser dem Licht einer UV-Lampe (254 nm Wellenlänge) ausgesetzt. Das UV-Licht dringt durch das Fenster des Bausteingehäuses und fällt auf den Halbleiter-Chip. Hierdurch kann die Ladung der Kondensatoren am Gate abfließen, und alle Zellen haben wieder H-Zustand.
Insbesondere diese Bausteine werden heute vielfach als Programmspeicher in Mikrocomputerschaltungen eingesetzt.

Beispiel:
2716 → EPROM mit 2048 x 8 bit = 2kB Speicherkapazität,
27128 → EPROM mit 16384 x 8 bit = 16kB Speicherkapazität.

7.3 Ein-/Ausgabe-Einheiten

Der Mikroprozessor — ergänzt durch Speicher — stellt eine autonome Rechen- und Steuereinheit dar. Diese Einheit ist jedoch praktisch solange wertlos, wie es keine Möglichkeit gibt, einen Kontakt mit der Umwelt herzustellen.
Ein minimaler Kontakt würde beispielsweise darin bestehen, daß diese Einheit ständig Informationen an die Umwelt abgibt (z. B. Computer-Uhr). Darüber hinaus ist es jedoch meistens auch erforderlich, Informationen an den Mikroprozessor zu geben, damit auf seinen Programmablauf Einfluß genommen werden kann (z. B. Eingabe einer Weckzeit in eine Computer-Uhr).
Zur Kontaktaufnahme mit dem Mikroprozessor sind spezielle Ein-/Ausgabekanäle erforderlich. Fortschrittliche Mikroprozessoren verfügen manchmal selbst schon über einige solcher Kanäle (z. B. SID und SOD beim MP 8085).

Im allgemeinen werden jedoch mehr Kanäle benötigt, und es ist deshalb erforderlich, spezielle Ein-/Ausgabe-Bausteine einzufügen, als Bindeglieder zwischen dem Mikroprozessor und der Umwelt. Diese Bausteine haben die Aufgabe, Vermittler (interface) zwischen dem für den Datenaustausch zuständigen Datenbus und einer peripheren Einheit (Drucker, Anzeige, Tastatur, Maschine etc.) zu sein.
Damit der Datenbus einwandfrei arbeiten kann, müssen auch die Eingabe- und Ausgabe-Kanäle — ebenso wie die Speicher — über entsprechende Schalter (Tristate-Gatter) zum richtigen Zeitpunkt an den Datenbus angekoppelt werden (**Bild 7.9**).

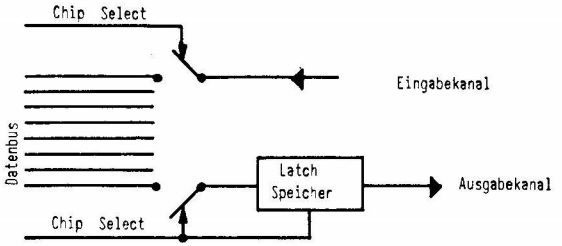

Bild 7.9. Anschluß von Ein-/Ausgabekanälen an den Datenbus

Bei einem Ausgabe-Kanal wird meistens ein Zwischenspeicher vorgesehen, da die auszugebende Information auf dem Datenbus nur für sehr kurze Zeit (weniger als eine Mikrosekunde, Bild 7.4) erscheint. Am Ausgabe-Kanal bleibt dann die Information solange erhalten, bis sie durch einen erneuten Ausgabevorgang überschrieben wird.

Ein-/Ausgabe-Baustein 8255
Eine große Flexibilität und Leistungsfähigkeit läßt sich durch die Verwendung spezieller, hochintegrierter und allgemein verwendbarer Ein-/Ausgabe-Bausteine erreichen. Solche Bausteine enthalten meistens mehrere, frei programmierbare Schnittstellen zum Datenbus. Jede Schnittstelle besteht aus acht Kanälen — entsprechend den 8 bit des Datenbusses — und wird häufig als „Port" bezeichnet.
In **Bild 7.10** ist ein zur MP 8080-Familie gehörender Port-Baustein dargestellt. Zum Anschluß an den MP 8085 sind neben dem Datenbus noch vier Steuerleitungen und im einfachsten Fall noch drei Adreßleitungen erforderlich.
Die Steuersignale WR und RD (write/read) dienen zur Festlegung der Datenflußrichtung auf dem internen Datenbus.
Das Reset-Signal bewirkt eine Programmierung der drei Ports A, B, C auf Eingabe. Hierdurch wird erreicht, daß kurz nach dem Einschalten des Mikroprozessors — bevor der Port-Baustein programmiert werden kann — keine undefinierten Ausgaben auf einem Port erfolgen können. Dies könnte sonst zu schweren Störungen oder gar zu Zerstörungen an peripheren Geräten führen. Die Bausteinauswahl (CS) kann durch die UND-Verknüpfung zwischen dem IO/M-Signal und einer Adreßleitung erfolgen. Beachtet werden muß hierbei jedoch, daß die Adreßbits A_0 und A_1 für eine andere Aufgabe reserviert sind.

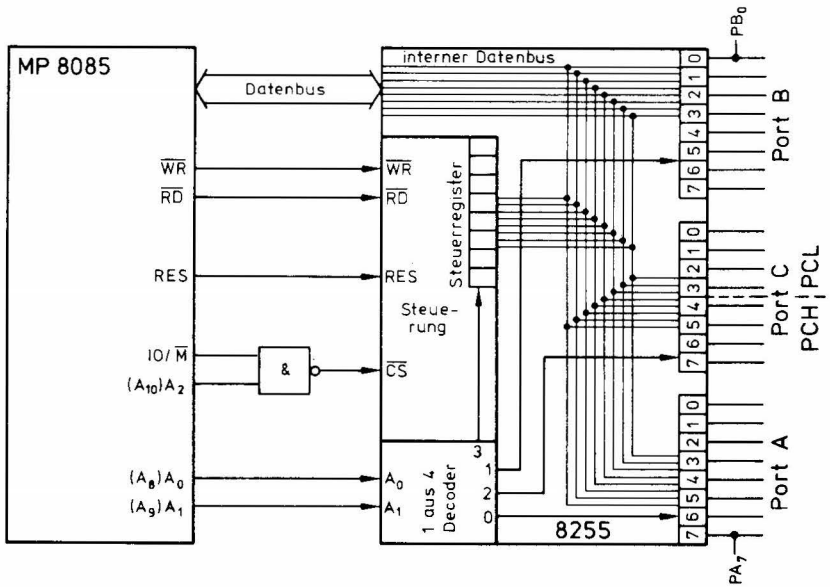

Bild 7.10. Anschluß des programmierbaren Port-Bausteines 8255 an den MP8085

Die Adreßbits A_0 und A_1 dienen — über einen internen „1-aus-4-Decoder" — zur Auswahl der im Baustein 8255 vorhandenen Register. Je nach dem, welche zweistellige Binärzahl über A_0, A_1 anliegt, wird eines der vier Register mit dem internen Datenbus verbunden. Die rechte Tabelle in **Bild 7.11** zeigt die entsprechende Zuordnung.
Das Steuerregister (SR) nimmt ein Datenwort auf, dessen einzelne Bits den Betriebszustand des 8255 festlegen. Die drei Ports (PA, PB, PC) enthalten jeweils einen Speicher (Register) zur Sicherung von Ausgabedaten.
Wie schon erwähnt, bringt ein Reset-Signal alle drei Ports in den Eingabe-Zustand. Nach dem Einschalten des Mikroprozessors erzeugt im allgemeinen eine spezielle Schaltung automatisch ein Reset-Signal. Hierdurch sind alle Ports zunächst auf Eingabe programmiert.
Das Festlegen der Aufgabe einzelner Kanäle oder ganzer Ports — bedingt durch den

CS	WR	RD	RES	
X	X	X	H	Programmierung auf Eingabe
H	X	X	L	Baustein vom
L	H	H	L	Datenbus getrennt
L	H	L	L	Lesen, Port ---▸ MP, Eingabe
L	L	H	L	Schreiben, MP ---▸ Port, Ausgabe

A_0	L	H	L	H
A_1	L	L	H	H
Auswahl	PA	PB	PC	SR

Bild 7.11. Steuersignale des 8255 und Auswahl der internen Register.
PA = Port A, PB = Port B, PC = Port C, SR = Steuerregister

konkreten Einsatz des Mikrocomputers — erfolgt durch die ersten Befehle des Mikrocomputerprogrammes, das ja nach einem Reset, mit der Adresse 0 (MP 8080/85) beginnend, abgearbeitet wird.

Beispiel:
MVI A, 8AH : Steuerwort in den Akku bringen,
OUT 7 : Steuerwort ins Steuerregister schreiben.

Das Aussenden der Adresse 0707H im obigen Beispiel erzeugt ein Bausteinauswahlsignal für den 8255 in Bild 7.10, denn IO/M und A_2 haben den Zustand H (7 = LHHH). Da auch die Adreßbits A_0 und A_1 im Zustand H sind, wird im 8255 das Steuerregister mit dem Datenbus verbunden, und es nimmt den Wert 8A auf. Nun ist der Port-Baustein 8255 programmiert.
Das im obigen Beispiel benutzte Steuerwort 8A programmiert Port A — sowie die niederwertige Tetrade des Ports C (PCL) — auf Ausgabe und Port B — sowie die höherwertige Tetrade des Ports C (PCH) — auf Eingabe.

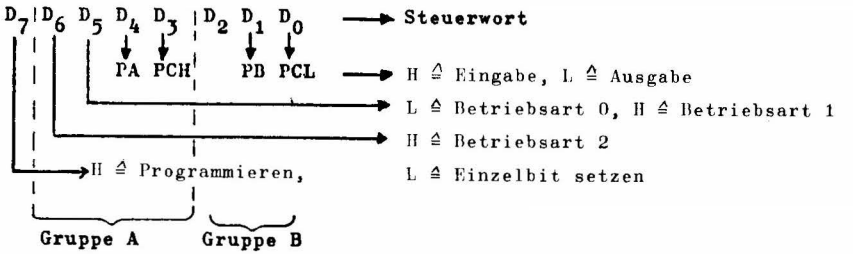

Bild 7.12. Steuerwort für den 8255

Die Bedeutung der einzelnen Bits des Steuerwortes wird in **Bild 7.12** erklärt. Das im letzten Beispiel angewandte Steuerwort 8A entspricht der Binärzahl HLLLHLHL. Dies sagt aus, daß eine Programmierung erfolgen soll (D_7 = H), daß die Betriebsart 0 gewünscht wird und legt weiterhin die schon beschriebenen Datenflußrichtungen der einzelnen Ports fest. Die Gruppen A und B können auf verschiedenen Betriebsarten programmiert werden (siehe Datenbücher). Bei der schon beschriebenen Betriebsart 0 sind die Ports als einfache Ein-/Ausgabe-Schnittstellen zu benutzen. Die Betriebsart 1 gestattet eine getastete Ein-/Ausgabe; hierbei werden die Ports A oder B in Verbindung mit Abtastimpulsen oder Quittungs-Signalen des Ports C benutzt. Eine getastete Zweiweg-Bus-Ein-/Ausgabe ist schließlich in der Betriebsart 2 möglich.
Die Adressen der Ports des Bausteines in Bild 7.10 — sie werden im folgenden Beispiel benutzt — ergeben sich aus Bild 7.11 zu:
Port A = 04, Port B = 05, Port C = 06.
Diese Adressen sind allerdings nicht eindeutig, denn auch jede andere Adresse eines IN-oder OUT-Befehles, bei der A_2 = H ist, selektiert den Baustein in Bild 7.10.
Eine eindeutige Bausteinauswahl wird in Bild 8.3 gezeigt.
Wenn mehrere 8255 an den Datenbus anzuschließen sind, kann jeweils eines der

Adreßbits A_2 bis A_7 die Bausteinauswahl bewirken; auf diese Weise können sechs Port-Bausteine 8255 gleichzeitig benutzt werden. In diesem Fall ist jedoch darauf zu achten, daß immer nur eines der Adreßbits $A_{2\ldots7}$ den Zustand H annimmt, da sich sonst mehrere Ports um den Datenbus „streiten".

Beispiel:
Zum Vergleich zwischen Hard- und Software-Lösungen soll die im **Bild 7.13** dargestellte UND-Verknüpfung durch die Benutzung der Ein-/Ausgabekanäle des Bausteines in Bild 7.10 realisiert werden. Zu diesem Zweck sind die acht Eingänge an Port B (Adresse 05) und die Ausgänge an Port A (Adresse 04) anzuschließen.

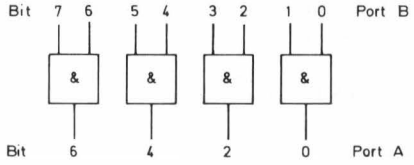

Bild 7.13. UND-Verknüpfung zwischen 4 x 2 Eingangsgrößen mit Gatter-Bausteinen

Das folgende Programm liest immer wieder die Zustände an den acht Eingängen in den Akku ein (IN 05) und führt dann eine UND-Verknüpfung paarweise durch. Hierzu werden die eingelesenen Zustände einmal im Register B aufbewahrt (MOV, B, A), dann alle Bits um eine Position nach links verschoben (RRC) und schließlich die Inhalte der Register B und A UND-verknüpft (ANA B). Im Akku steht dann das Ergebnis der UND-Verknüpfung zwischen den Eingängen 0,1 1,2 2,3 3,4 usw.

```
1000 DB05   IN    05H   ; Port B (8 bit) lesen → Akku (A7 . . . A0)
1002 47     MOV   B, A  ; Akku-Inhalt ins Register B kopieren (B7 . . . B0)
1003 0F     RRC         ; ⟨ A ⟩ verschieben: A0 A7 A6 A5 A4 A3 A2 A1
1004 A0     ANA   B     ; UND-Verknüpfung mit ⟨ B ⟩ : B7 B6 B5 B4 B3 B2 B1 B0
1005 D304   OUT   04H   ; Ausgabe des Ergebnisses: PA6 PA4 PA2 PA0
1007 C30010 JMP   1000H ; Sprung zum Anfang
```

Die Verknüpfungen 1,2 3,4 usw. werden nicht gebraucht (Bild 7.13), deshalb enthalten nur die vier geraden Bitnummern (0, 2, 4, 6) die Ausgangs-Zustände der Gatter aus Bild 7.13; diese erscheinen dann auf den Portleitungen (Port A) PA_0 bis PA_6.

8 Mikrocomputer-Anwendung

Prof. Dr. rer. nat. *Dieter Hannemann,* FH Bochum (Abt. Gelsenkirchen)

Aus der Sicht des Aufgabenbereiches können bei der Anwendung von Mikrocomputern zwei Fälle unterschieden werden:
1. Verwendung fertiger Platinen;
2. Entwicklung spezieller Platinen.

Der erste Anwendungsfall tritt vor allem auf, wenn nur geringe Stückzahlen (z. B. 100 pro Jahr) gebraucht werden oder wenn fertige Platinen vorliegen, die im konkreten Fall den Bedarf genau abdecken.
Hierzu gibt es von einer Reihe von Anbietern mehr oder weniger umfangreiche Baugruppensysteme — häufig auf der Basis einzelner Eurokarten (100 x 160 mm^2). Mit diesen einzelnen Platinen lassen sich dann Mikrocomputer für die unterschiedlichsten Anwendungen zusammenstellen (**Bild 8.1**).
Von der zweiten Möglichkeit, nämlich der Entwicklung eigener, spezieller Mikrocomputerschaltungen, wird vor allem dann Gebrauch gemacht, wenn große Stückzahlen oder sehr spezielle Anforderungen zu realisieren sind. Hier treten dann im besonderen Maße wirtschaftliche Überlegungen in den Vordergrund.

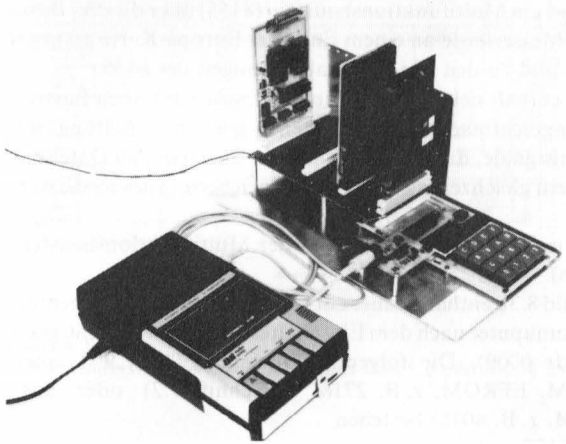

Bild 8.1. Mikrocomputer auf der Basis des in Bild 6.3 dargestellten Minimalsystemes MICO 85 (Werkbild: PTFE [4]).
Auf einer Busplatine können bis zu 13 weitere Baugruppen angeordnet werden.
Die erste senkrechte Platine (rechts) ist die MC-Baugruppe (Abschnitt 8.1); dahinter steht eine 16-kB-RAM-Platine, und ganz links ist eine EPROM-Programmier-Platine angeordnet.

Im folgenden wird anhand von Beispielen gezeigt, wie eine universelle Mikrocomputer-Platine aufgebaut sein kann und wie hiermit eine Motorsteuerung —in ihren wesentlichen Elementen — zu realisieren ist.

8.1 Mikrocomputer-Platine

Die hier beschriebene MC-Platine entstammt dem in Bild 8.1 dargestellten System (MICO 85 [1,4]).

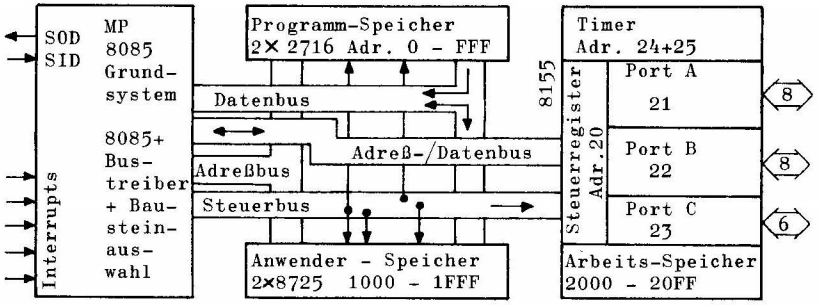

Bild 8.2. Aufbau eines Mikrocomputers am Beispiel des in Bild 8.1 gezeigten Systemes (s. a. Bild 8.3)

Die Struktur des Mikrocomputers ist in Bild 8.1 dargestellt. An das Grundsystem (s. a. Bild 8.3a) sind die Speicher und ein Multifunktionsbaustein (8155) über die drei Busse angeschlossen. Eine 96polige Messerleiste an einem Ende der Europa-Karte gestattet den Kontakt zu allen Bussen und zu den Ein-/Ausgabeleitungen des 8155.

Das Grundsystem (Bild 8.3a) enthält neben dem Mikroprozessor 8085 noch Bustreiber — damit viele Bausteine angeschlossen werden können — und eine Schaltung zum Erzeugen der Bausteinauswahlsignale, damit immer nur ein Baustein den Datenbus benutzt. Der Bustreiber „3" dient gleichzeitig zur Zwischenspeicherung des niederwertigen Adreßbytes.

An dem Grundsystem sind vier Speicherbausteine und der Multifunktionsbaustein 8155 angeschlossen (Bild 8.3b).

Der Speicherbaustein „6" in Bild 8.3b enthält immer ein nicht flüchtiges (permanentes) Programm, damit der Mikrocomuputer nach dem Einschalten eine sinnvolle Tätigkeit aufnehmen kann (Start bei Adr. 0000). Die folgenden Speicher „7" bis „9" können aus Festwertspeichern (ROM, EPROM, z. B. 2716, Abschnitt 7.2) oder aus Schreib-/Lesespeichern (RAM, z. B. 4016) bestehen.

Der Multifunktionsbaustein 8155 enthält:

— den Arbeitsspeicher (Notizblock des Computers),
— einen Timer (programmierbarer Zähler), der über die Teiler „12" und „13" einen 750-Hz-Takt bekommt, und
— drei Ein-/Ausgabeschnittstellen (Port A, Port B, Port C).

a)

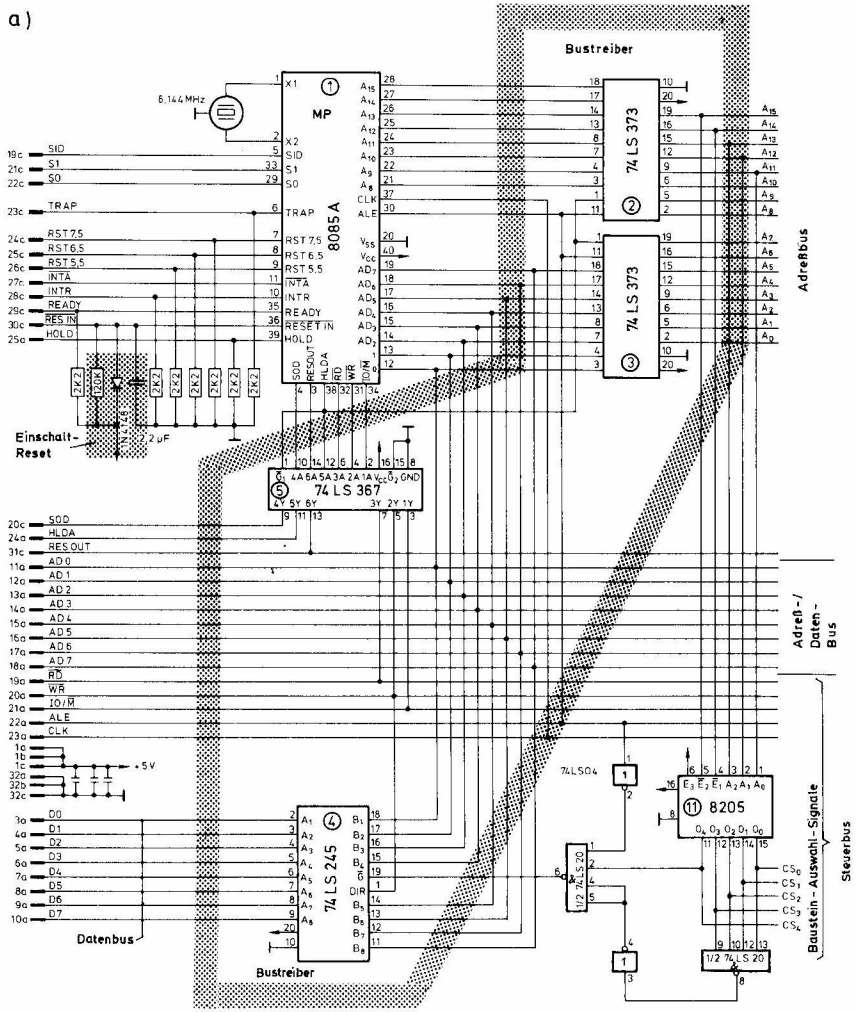

Bild 8.3.
a) Grundsystem der MC-Platine aus Bild 8.1 und Bild 8.2

An die Ports können z. B. eine Tastatur, eine Anzeige-Einheit und eine zu steuernde Maschine angeschlossen werden.

b)

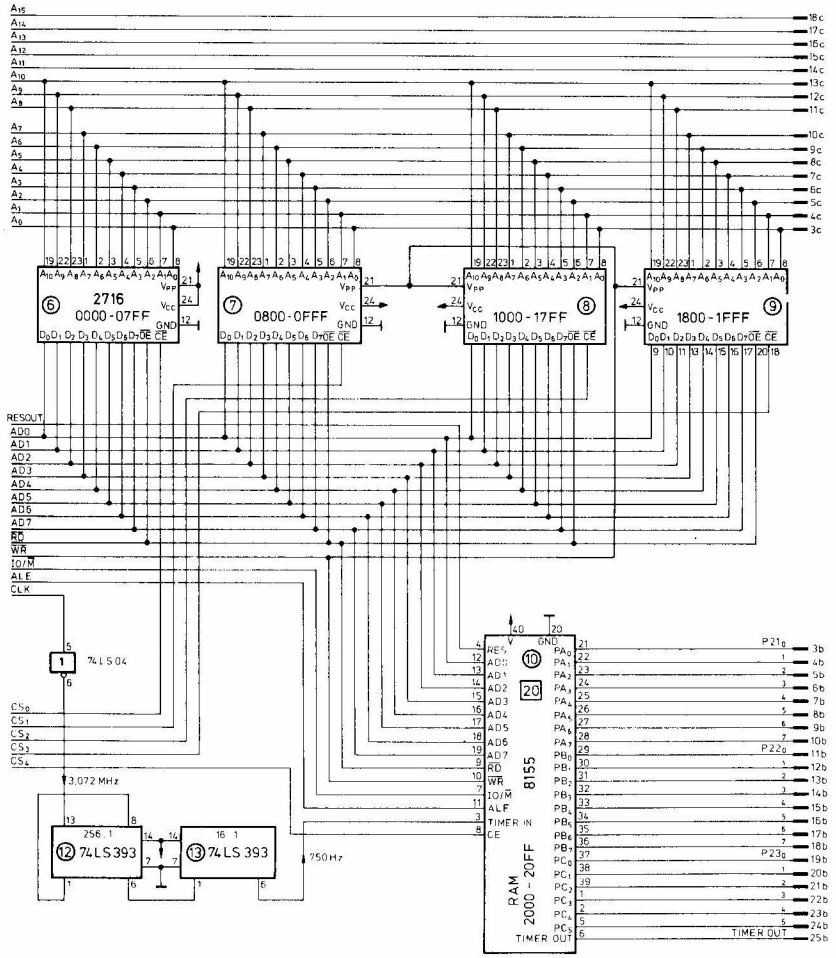

Bild 8.3.
b) Anschluß von Speichern und Ein-/Ausgabeports an das Grundsystem aus Bild 8.3a

8.2 Eingabetastatur

Der Anschluß von Tastern oder Schaltern an einen Mikrocomputer befähigt den Menschen, einen direkten Einfluß auf die internen Vorgänge des Mikrocomputers auszuüben. Diese Wirkung ist jedoch nur zu erreichen, wenn die angeschlossenen Taster oder Schalter im Programmablauf ständig abgefragt (polling) werden.

Beispiele:
Eingaben an den Mikrocomputer:
— Signal für Programmverzweigungen,
— Anforderung spezieller Aktivitäten,
— Eingabe von Kommandos und Befehlen,
— Speicherinhalte verändern.

Für eine möglichst universelle Kommunikation mit dem Mikrocomputer sind im einfachsten Fall 16 Taster erforderlich (Bild 6.3 und Bild 8.1).
Hierbei ist jedem der 16 Taster eine andere Hexadezimalzahl zugeordnet. Der Druck auf eine Taste bedeutet somit das Aussenden eines 4-bit-Wortes (Tetrade) an den Mikrocomputer.
Damit der Mikrocomputer dieses Wort empfängt und richtig „versteht", muß er durch ein entsprechendes Programm dazu veranlaßt werden, alle 16 Taster abzufragen und — wenn eine Taste gedrückt wurde — ihren Wert (0 . . . F bzw. LLLL . . . HHHH) in ein Register zu schreiben. Was dann mit diesem Wort oder dieser Zahl weiter geschehen soll, oder welche Bedeutung sie hat, wird ebenfalls durch eine vorherige Programmierung festgelegt.
Eine Minimaltastatur, bestehend aus 16 Tastern, wird häufig als quadratische Matrix gemäß **Bild 8.4** aufgebaut. Aufgrund dieser speziellen Anordnung sind nur acht E/A-Kanäle zur Abfrage von 16 Tastern erforderlich.

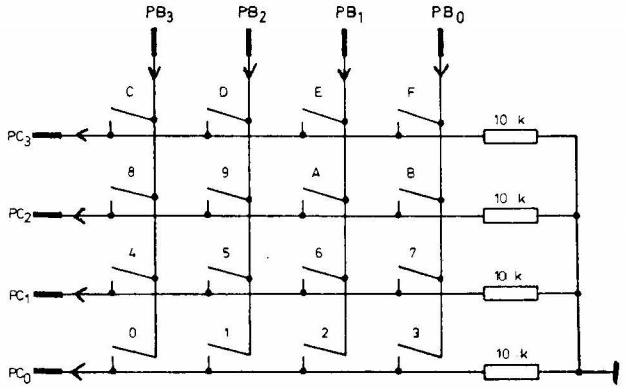

Bild 8.4. Hexadezimal-Tastatur gemäß Bild 8.1

Der hexadezimale Wert, welcher einer Taste zugeordnet wird, ist durch das Programm bestimmt.
In Bild 8.4 wurden die Zeilenleitungen an die niederwertige Tetrade des Ports C eines 8155 (Bild 8.3b) und die Spaltenleitungen an die niederwertigen Kanäle des Ports B angeschlossen. Bei geöffneten Tastern nehmen die Eingänge $PC_{0\ldots3}$ infolge derPull-down-Widerstände den L-Zustand an. Wenn nun auf einen der Ausgänge $PB_{0\ldots3}$ ein H gegeben wird und darüber hinaus in der selben Spalte eine Taste gedrückt ist, erscheint auf der entsprechenden Zeile ein H.

Beispiel:
Der Taster A sei geschlossen (Bild 8.4).
MVI A, 2 ; LLLL LLHL,
OUT PB ; PB_1 = H,
IN PC ; Akku-Bit 2 = H.
Auf dem Eingang PC_1 (Bild 8.4) erscheint ein H.

Durch die Ausgabe eines H auf allen vier Spaltenleitungen nacheinander und das jeweilige Einlesen der vier Zeilenleitungen ist eindeutig feststellbar, welche Taste gedrückt wurde.

8.3 Anzeige-Einheit

Wie in Bild 8.1 zu sehen ist, kann an die Ports der MC-Platine (Bild 8.3b) auch eine Anzeige-Einheit angeschlossen werden.
Die Anzeige-Einheit besteht aus acht Siebensegment-LED-Bausteinen, mit denen sich das Hexadezimalalphabet (0 . . . F) darstellen läßt. Zur Entlastung des Mikrocomputers wurde auf direktes Ansteuern der Anzeige-Elemente verzichtet. Stattdessen kommen hochintegrierte Spezialbausteine zur Anwendung (**Bild 8.5**).

Jeder dieser Bausteine (7212MIPL) enthält vier Speicher, Decoder und Treiber zum Anschluß von vier Anzeige-Elementen. Die vier Datenspeicher, welche jeweils die 4 bit einer darzustellenden Tetrade aufnehmen, sind an Port $A_{0\ldots3}$ ($P21_{0\ldots3}$) der MC-Platine angeschlossen und können über die zwei DSC (digit select code)-Leitungen angewählt werden ($2^2 = 4$). Über zwei generelle Auswahlleitungen ($CS_{1\ldots2}$), angeschlossen an Port A_6 und A_7, ist festlegbar, welcher Steuerbaustein jeweils gemeint ist.

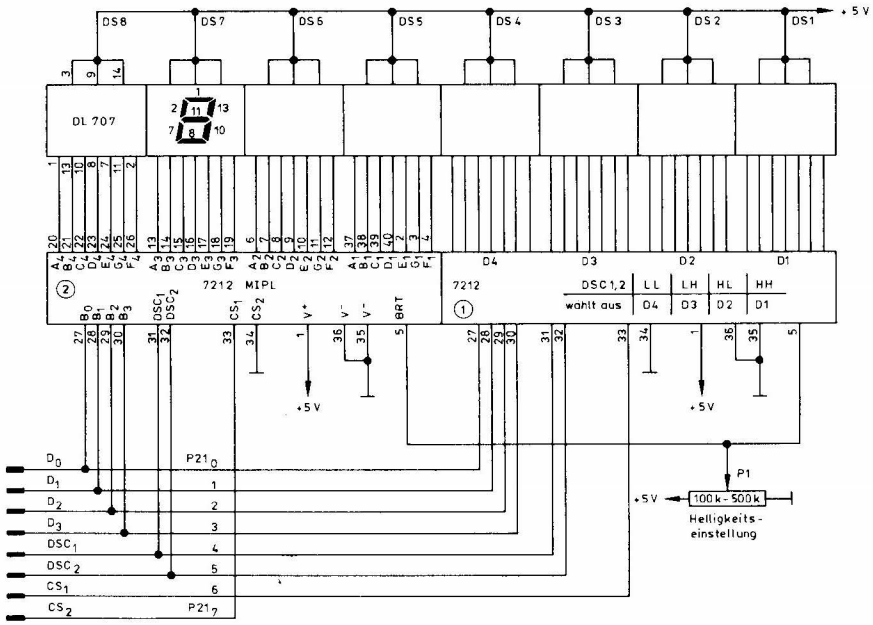

Bild 8.5. 8-stellige Anzeigeeinheit für das Minimalsystem in Bild 6.3 und Bild 8.1

8.4 Motorsteuerung

Zur Verdeutlichung der Möglichkeiten einer Maschinensteuerung mit dem bis hierher beschriebenen System soll kurz auf eine Motorsteuerung eingegangen werden.
Das **Bild 8.6** zeigt im oberen Teil einen Gleichstrommotor, dessen Anschlüsse über jeweils zwei Schalttransistoren (T_1, T_3 bzw. T_2, T_4) an die positive Motorbetriebsspannung (U_M) und die Masse (Minuspol) angeschlossen werden kann.
So lange nur die beiden oberen oder nur die beiden unteren Transistoren durchgeschaltet sind, läuft der Motor nicht. Wenn dagegen zwei diagonal angeordnete Transistoren durchschalten, läuft der Motor in der einen oder anderen Drehrichtung (Vertauschung der Polarität am Motor). Unzulässig ist es, zwei übereinander liegende Transistoren durchzuschalten, da dann ein Kurzschluß entsteht. Damit dies nicht eintreten kann, und zur Erzeugung der nötigen Basissteuerströme, wurde ein Interface — bestehend aus vier invertierenden Treibern mit offenem Kollektor (Bild 3.7) — vorgesehen.

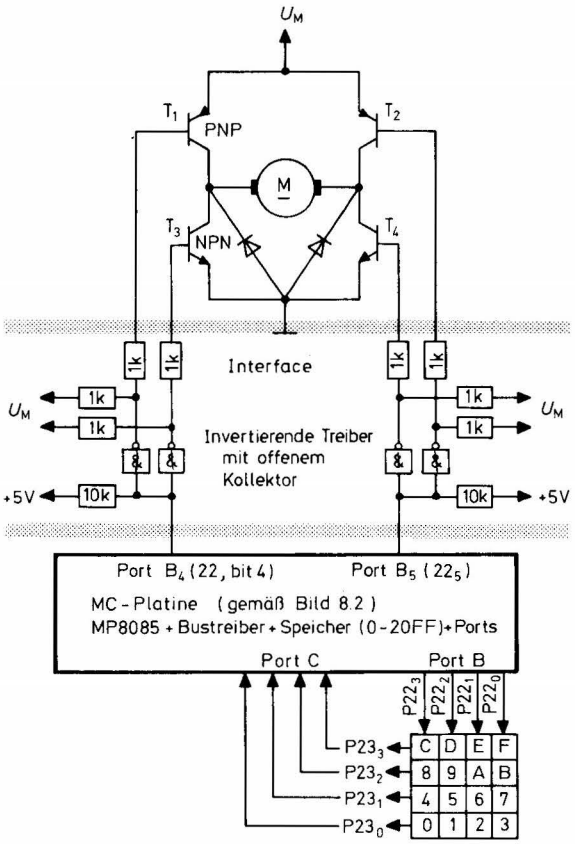

Bild 8.6. Motorsteuerung mit MC-Platine (Bild 8.2) und Tastatur (Bild 8.4).
T1 = T2 = BD 680 (80V/4A), T3 = T4 = BD 679 (80V/4A), Treiber = 7406 (30V/40mA).

Auf diese Weise bleiben nur noch zwei Steuerleitungen übrig. Wenn beide Leitungen H-oder L-Potential haben, steht der Motor. Hat dagegen eine Leitung den Zustand H und die andere L, so läuft der Motor entweder im Rechts- oder Linkslauf, je nach dem an welcher Leitung der logische H-Pegel anliegt.
Im unteren Teil von Bild 8.6 ist noch einmal die MC-Platine aus Bild 8.2 dargestellt. Zwei Kanäle des Ports B ($PB_{4\ldots5}$) sind mit den beiden Motorsteuerleitungen verbunden (während an den anderen Kanälen z. B. eine Tastatur und eine Anzeige angeschlossen sind).
Durch Veränderung der logischen Zustände (L oder H) auf den beiden Motorsteuerleitungen können vom Mikrocomputer die unterschiedlichsten Betriebszustände erreicht werden. Zur Kontrolle dieser Betriebszustände sind an andere Kanäle des

Mikrocomputers noch entsprechende Sensoren anschließbar (Drehzahlgeber, Lichtschranken, Strommesser etc.).

Start des Motors
Wenn die Schaltung aus Bild 8.6 (MICO 85, Bild 8.1 [4]) benutzt wird, bewirken die folgenden Befehle, daß der Motor zu laufen beginnt.

```
IN   22H   ; Zustände der Kanäle des Ports B in den Akku einlesen;
ORI  10H   ; ODER-Verknüpfung des Akkus mit 10H. Ergebnis: Bit 4 = H;
ANI  ODFH  ; UND-Verknüpfung von A mit DFH. Ergebnis: Bit 5 = L;
OUT  22H   ; Ausgabe von A auf Port B. Ergebnis: der Motor läuft links herum;
JMP  0078H ; Rücksprung in das Monitor-Programm.
```

Wenn zur Realisierung dieser Steuerung ein universelles System wie in Bild 6.3 oder Bild 8.1 benutzt wird, so enthält der erste Speicherbaustein („6" in Bild 8.3b: 0000-07FF) ein sogenanntes Monitor-Programm.
Dieses Grundprogramm kann durch die Eingabe von Kommandos über die Hexadezimal-Tastatur (Bild 8.4) zu verschiedenen Dienstleistungen herangezogen werden:
— Programmbytes in einen bestimmten Anwendungsspeicherbereich schreiben,
— Speicherinhalte inspizieren und verändern,
— Programme starten und
— Programme testen etc.

Die Eingabe eines Kommandos kann in einfachster Weise dadurch erfolgen, daß jedem Tastenwert (0 . . . F) ein anderes Kommando zugeordnet wird. Dann genügt der Druck auf eine Taste, um eine bestimmte Fähigkeit des Monitors zu aktivieren.
Darüber hinaus ist es bei einem leistungsfähigen System möglich, vorhandene Unterprogramme (im Monitor) auch dem Anwender nutzbar zu machen:
— Eingaben von der Tastatur annehmen,
— Ergebnisse auf die Anzeigen bringen etc.

Dies alles bedeutet — angewandt auf das letzte Beispiel-Programm —, daß der Hex-Code des Programmes (DB 22 F6 10 E6 DF D3 22 C3 78 00) z. B. in den Speicherbereich ab 1000H eingegeben und dann gestartet wird.

Beispiel:
Es werden nacheinander die folgenden Tasten ⟨ ⟩ gedrückt (MICO 85):

⟨ 1 ⟩:	Kommando 1 (Bytes in den Speicher schreiben).
⟨ 1 ⟩⟨ 0 ⟩⟨ 0 ⟩⟨ 0 ⟩:	Ab der Adresse 1000H einschreiben.
⟨ D ⟩⟨ B ⟩⟨ 2 ⟩. . .:	Alle 11 Byte des Programms eingeben.
⟨ RESET ⟩:	Einschreiben beenden, weitere Kommandos annehmen.
⟨ 3 ⟩:	Kommando 3, Sprung auf die Adr. 1000H und damit Start des dort eingeschriebenen Programmes.

Nach Ausführen der vier Befehle des obigen Programmes (Start des Motors) wird ein Rücksprung (JMP 0078H) in den Monitor vorgenommen. Daraufhin können erneut Kommandos gegeben werden.

Drehrichtungsumkehr

Abschließend wird an einem kurzen Programm gezeigt, wie über die Tastatur der Motor zum Rechts-, Linkslauf oder zum Stillstand gebracht werden kann. Hierzu wird das Tastatur-Abfrageprogramm des Monitors (CALL 001B) verwandt. Der Druck auf die Taste „0" heißt Linkslauf, auf „1" heißt Rechtslauf und auf jede andere Taste heißt Stillstand.

```
1000  DB22    IN    22H    ; Port B einlesen, Ergebnis im Akku
1002  E6CF    ANI   OCFH   ; UND-Verknüpfung des Akkus mit CF → Bit 4 und
                           ; 5 = L
1004  47      MOV   B, A   ; Akkuinhalt in Register B sichern
1005  CD1B00  CALL  001BH  ; Tastatur abfragen, Tastenwert im Akku
1008  FEFF    CPI   OFFH   ; Keine Taste gedrückt? (Kennzeichen FF)
100A  CA0510  JZ    1005H  ; Tastatur erneut abfragen
100D  0E10    MVI   C, 10H ; Bit 4 = H für Linkslauf setzen
100F  FE00    CPI   00     ; Linkslauf ? (Taste 0)
1011  CA1D10  JZ    101DH  ; Wenn ja, Steuersignale geben
1014  0E20    MVI   C, 20H ; Bit 5 = H für Rechtslauf setzen
1016  FE01    CPI   01     ; Rechtslauf ? (Taste 1)
1018  CA1D10  JZ    101DH  ; Wenn ja, Sprung!
101B  0E30    MVI   C, 30H ; Beide Steuerbits = H für Stopp
101D  78      MOV   A, B   ; Port-B-Wert zurück in den Akku
101E  81      ADD   C      ; Entsprechende Steuerbits H setzen
101F  D322    OUT   22H    ; Ausgabe auf Port B
1021  C30010  JMP   1000H  ; Sprung an den Anfang
```

8.5 Schrifttum

[1] Hannemann, D.: Einführung in die Mikrocomputer-Technik. 3. Aufl. Essen: Verlag W. Girardet, 1983

[2] Hannemann, D.: Wie erlernt man den Umgang mit Mikroprozessoren? Elektro-Anzeiger 13 (1982) S. 12-13

[3] Hannemann, D.: Testhilfe für Mikroprozessorsschaltungen, Elektronik-Applikation 14 (1982) H. 12, S. 47-50

[4] PTFE: Systembeschreibung MICO 85. Hermann-Löns-Str. 6, 4650 Gelsenkirchen-Buer, Tel. (0209)39 76 47

[5] Siemens: Mikrocomputer-Bausteine, Datenbücher

9 Der Transistor als Leistungsschalter

Dipl.-Ing. *Ingo J. Arnold,* BBC-AG, Heidelberg

In der Leistungselektronik werden im Bereich unter 100 A und 800 V in zunehmendem Maße Transistoren eingesetzt, wo bisher Röhren und Thyristoren ihren Einsatz fanden. Die Anwendung „Schalt"-Transistor erfolgt weniger als Ersatz für mechanische Schaltkontakte, z. B. bei Schützen und Relais, sondern dort, wo Stell- und Regelaufgaben zu erfüllen sind. Im Stromversorgungsteil und in der Horizontalablenkung von Fernsehempfängern, bei Netzgeräten und zur Drehzahlregelung von Gleichstrom- und Drehstrommotoren sind einige Schwerpunkte der Anwendung zu finden.

Mit diesen Leistungs-Stellgliedern sollen jeweils die Ausgangsgrößen Spannung oder Strom gesteuert bzw. verstellt werden. In den meisten Anwendungen soll diese Verstellung analog bzw. linear und stetig erfolgen; trotzdem wird dazu sehr vorteilhaft ein Wirkprinzip verwendet, das auf dem unlinearen Verhalten eines Schalter-Stellgliedes beruht, siehe auch Abschnitt 13.3.

Bild 9.1 veranschaulicht die Wirkungsweise eines einfachen (Einweg-) Schalterstellgliedes. Die aus dem Wechselspannungsnetz gewonnene Gleichspannung U_G wird mit dem Schalter V (Bild 9.1a) periodisch voll, aber zeitbegrenzt zum Ausgang durchge-

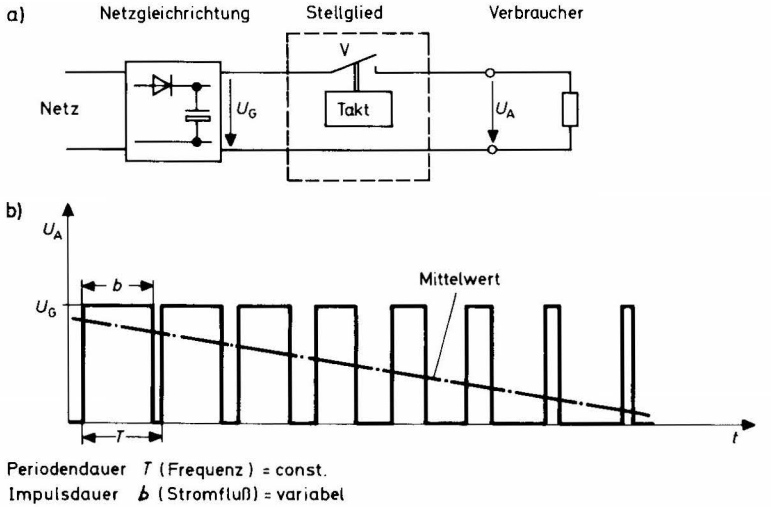

Bild 9.1. Gleichstromsteller mit Schalt-Stellglied
a) prinzipieller Aufbau;
b) Ausgangsspannung