Mikroprozessoren und Mikrorechner

Lehrheft 5

ITT Fachlehrgänge, Pforzheim

ITT

1., 2. und 3. Auflage, Juli 1977

Druck: Druckerei Seiter, 7535 Königsbach-Stein

Mikroprozessoren und Mikrorechner

Verfasser:
Dr. Jürgen Gerlach
C. D. Nabavi, B. Sc.
Forschungszentrum der
Standard Elektrik Lorenz AG
Stuttgart

7. Hardware- und Software-Hilfsmittel zum praktischen Arbeiten

Jede Gerätetechnik erfordert ihre besonderen Entwicklungshilfsmittel und Meßgeräte. Auch zum Entwickeln von Geräten, die einen Mikrocomputer enthalten, ist eine Reihe besonderer Hilfsmittel erforderlich, die erst dem Entwickler ein effektives Arbeiten ermöglichen. Es genügt nicht, sich die entsprechenden Bauelemente zu kaufen und diese dann nach den Schaltungsunterlagen des Herstellers zusammenzubauen. Man erhält auf diese Weise zwar sehr einfach einen Mikrocomputer, aber es wird schon Probleme geben festzustellen, ob dieser Computer funktioniert oder nicht. Da man nicht genau weiß, was im Innern der Schaltung passiert, kann man mit einer solchen Karte zunächst recht wenig anfangen. Auch ein Programm läßt sich hierauf nicht vernünftig entwickeln. Man könnte sich zwar vorstellen, daß man ein Programm schreibt und dieses z. B. von einem Halbleiterhersteller in RePROMs laden läßt und somit den Mikrocomputer zum „Laufen" bringt. Sollte aber dieses Programm einen Fehler enthalten, so ist es sehr schwer, diesen Fehler zu finden, da dieser Mikrocomputer wenig Eingriffs- und Diagnosemöglichkeiten hat. Deshalb ist zur Geräte- und Programmentwicklung ein viel flexibleres System mit entsprechenden Geräten und entsprechenden Hilfsprogrammen erforderlich.

Bei der Entwicklung eines Gerätes, das einen Mikrorechner enthält, muß zunächst das eigentliche Gerät mit Stromversorgung usw. entwickelt werden. Dann ist ein Programm zu erstellen, das dieses Gerät steuert. Hier steht der Entwickler oft vor dem Problem, daß das Labormodell für sich allein nicht vollständig getestet und damit häufig auch nicht vollständig entwickelt werden kann, bevor nicht das Programm komplett zur Verfügung steht. Andererseits kann aber vielfach auch das Programm nicht vollständig entwickelt und getestet werden, wenn nicht das komplette Gerät zur Verfügung steht, da ansonsten u. U. nicht alle Reaktionen des Gerätes in das Programm einbezogen werden können. Man wird deshalb das Labormodell mit konventionellen Mitteln so weit wie möglich entwickeln. Das gleiche gilt für das Programm. Man verwendet einen normalen Rechner und entwickelt darauf das Programm. Die Reaktionen des zu entwickelnden Gerätes versucht man mit dem normalen Rechner so weit wie möglich zu simulieren. Anschließend erfolgt dann die Phase der Integration von Gerät und Programm. In dieser Phase ist es häufig recht schwierig, Gerät und Programm einander anzupassen. Wenn eine Fehlfunktion auftritt, kann diese ihre Ursache in der Hardware oder in der Software haben. In diesem Falle sind dann Spürsinn und entsprechende Hilfsmittel erforderlich, damit man einen solchen Fehler finden kann. Nachfolgend sollen die Hilfsmittel, Geräte und Hilfsprogramme besprochen werden, die der Entwickler für die verschiedenen Aufgaben benötigt.

7.1 Laborausrüstung

Über die übliche Ausstattung eines Labors hinaus ist zum Arbeiten und Entwickeln mit Mikrocomputern ein vollständiges Mikrorechnersystem erforderlich. Eine Minimalkonfiguration eines solchen Systems wäre etwa:

- Der Mikrorechner selbst mit der erforderlichen Stromversorgung, Gehäuse für alle Baugruppen usw.
- Ein Fernschreiber oder ein äquivalentes Gerät mit Eingabetastatur, alphanumerischem Druckwerk, Lochstreifenleser und Lochstreifenstanzer mit der entsprechenden Anpassungsschaltung (Interface) zum Rechner
- Ein Schreib/Lesespeicher mit einer Kapazität von etwa 4 k Wörtern oder mehr, abhängig von der Größe der zu entwickelnden Programme und dem Speicherbedarf der vom Hersteller angebotenen Hilfsprogramme
- Fassungen für Festwertspeicher in ausreichender Menge für die Hilfsprogramme und für eigene Programme
- Ein Bedienungspult mit Schaltern und Anzeigelampen, das einen Zugriff zu dem Adreß- und dem Datenbus ermöglicht. Es sollte möglich sein, den Speicher zu laden und die Inhalte nachzuprüfen (LOAD und EXAMINE), den Mikrocomputer zu starten und zu stoppen und ihn im Einzelschrittbetrieb zu betreiben
- Zugriff zu einem Programmier- und Löschgerät für RePROMs, so daß die entwickelten Programme in Festwertspeicher geladen werden können. Dieses Gerät wird normalerweise bei der Programmentwicklung nicht ständig benutzt (Programme während dieser Phase im RAM), so daß sich mehrere Arbeitsgruppen ein solches Gerät teilen können. Außerdem bieten viele Hersteller von Speicherelementen und Distributoren hier Serviceleistungen an,

die bei geringer Auslastung eines eigenen Gerätes billiger sein können als eigene Investitionen. Auf alle Fälle benötigt ein solches Programmiergerät einen Lochstreifenleser, besser noch einen Anschluß an den Mikrocomputer, so daß die Programme nicht von Hand eingegeben werden müssen. Eine manuelle Eingabe ist nur für Programme bis etwa 100 Instruktionen praktikabel, darüber sind der Zeitaufwand und die Wahrscheinlichkeit von Eingabefehlern viel zu hoch.
Praktisch alle Hersteller von Mikroprozessoren stellen heute auch komplette Mikrorechnersysteme her. Solche Geräte haben Ähnlichkeit mit Prozeßrechnern und lassen sich im wesentlichen auch wie diese verwenden. Sie enthalten in einem geeigneten Gehäuse den kompletten Mikrorechner mit den erforderlichen Speichern, der Stromversorgung sowie Interface-Schaltungen zum Anschluß an Pheripheriegeräte und einen Bedienungspult. Diese Geräte haben den Vorteil, daß normalerweise die vom Hersteller gelieferten Programme ohne weiteres verwendet werden können. Solche Systeme sind außerdem häufig erweiterbar, so daß auch zusätzliche Geräte wie Floppy-Disks, Kasettenlaufwerke, alphanumerische und grafische Displays usw. angeschlossen werden können. Die zu entwickelnden Geräte können über zusätzliche freie Interface-Schaltungen an einen solchen Laborcomputer angeschlossen werden, so daß das Zusammenspiel zwischen Programm und Gerät erprobt werden kann. Manche Systeme bieten hier sehr weitgehende Möglichkeiten. Über ein besonderes Interface wird der Laborcomputer so mit dem zu entwickelnden Gerät verbunden, daß der Laborcomputer den Mikrocomputer im Gerät vollständig „emuliert", also ersetzt. Damit stehen alle Test- und Diagnosemöglichkeiten des Laborcomputers zur Hardware-Entwicklung zur Verfügung. In dem Maße, wie dann die Hardware „wächst", werden die Steuerfunktionen vom Gerät selbst ausgeführt. Eine solche Möglichkeit ist sehr wertvoll und kann die Entwicklungszeit drastisch kürzen.

7.2 Hilfsmittel zur Programmentwicklung

Bei den Übungsprogrammen für den ITT MP-Experimenter sind wir grundsätzlich so vorgegangen, daß wir über einen Programmablaufplan (Flußdiagramm) die Programmstruktur und den logischen Ablauf festgelegt haben. Dann haben wir ein sog. Quellenprogramm mit Hilfe der mnemonischen Befehle erstellt, das dann in den betreffenden Maschinencode umgesetzt wurde. Die so erstellten Programme wurden dann in den Rechner geladen und getestet. Dieses manuelle Verfahren ist, wie Sie sicher bestätigen können, sehr zeitaufwendig, mit Fehlermöglichkeiten behaftet und deshalb für das praktische Arbeiten – vor allem bei größeren Programmen – ungeeignet. Verschiedene dieser Arbeitsschritte können durch **Hilfsprogramme** automatisiert und dadurch wesentlich erleichtert werden.
In den ersten Entwicklungsphasen, der Systemanalyse, der Festlegung der Hardware-Konzeption sowie der Erstellung des Programmablaufplanes gibt es naturgemäß wenig Automatisierungsmöglichkeiten. Hier wird der erfahrene Ingenieur und Programmierer benötigt. Auch das Übersetzen des Ablaufplanes in die mnemonischen Befehle ist „Handarbeit".
Bei den weiteren Schritten kann dann eine Automatisierung eintreten. Eine Voraussetzung dafür ist, daß das Quellenprogramm zunächst auf einem vom Rechner lesbaren Datenträger, wie z. B. Lochstreifen oder Band, gespeichert wird. Diese Aufgabe wird durch das **Editorprogramm** (kurz: Editor) erleichtert. Der Editor verwandelt den Rechner in eine „intelligente" Schreibmaschine mit weitgehenden Textverarbeitungsmöglichkeiten. Der edierende Text wird dann auf einem Datenträger abgespeichert.
Dieser Datenträger wird dann mit einem weiteren Programm, dem **Assembler**, verarbeitet. Der Assembler übersetzt das Programm automatisch in die Maschinenbefehle und speichert das Maschinenprogramm, auch **Objektprogramm** genannt, wieder auf dem Datenträger. Zusätzlich wird für den Programmierer eine **Programmliste** erstellt, die Maschinen- und Quellenprogramm enthält. Das Objektprogramm wird von einem weiteren Hilfsprogramm, dem **Lader**, in den Rechner geladen. Das Programm kann jetzt ausgeführt und getestet werden.

Auch für diese Testphase, die etwa 45 % der gesamten Entwicklungszeit erfordert, gibt es Hilfsprogramme, die **Debugging-Routines** (vom engl. bug = Wanze), die die Fehlersuche erleichtern. Sind Fehler gefunden, wird mit dem Editor das Quellenprogramm geändert, neu assembliert, geladen und wieder getestet. Dieser Vorgang wird so lange wiederholt, bis das

Programm fehlerfrei läuft. Je weniger manueller Arbeitsaufwand für einen solchen Zyklus erforderlich ist, desto schneller ist ein Programm entwickelt.
Der manuelle Arbeitsaufwand wird erheblich beeinflußt von der Rechnerausstattung. Ist ein schneller Massenspeicher, wie z.B. ein Plattenspeicher, verfügbar, so entfällt das zeitraubende Stanzen und Einlesen von Lochstreifen. Wenn mit Lochstreifen gearbeitet werden muß, so reduzieren schnelle Leser und Stanzer die zur Verarbeitung der Lochstreifen erforderliche Zeit auf ein Minimum. Ein Bildschirmgerät in Verbindung mit einem geeigneten Editor vervielfacht die Leistung beim edieren und korrigieren von Programmen.
Das in Abschnitt 7.1 beschriebene Minimalsystem läßt sich ohne weiteres zur Programmentwicklung verwenden. In diesem Falle arbeitet man mit einem Fernschreiber und einem Lochstreifen. Das heißt, der Zeitaufwand für einen Korrekturzyklus wird vor allem bei längeren Programmen sehr hoch. Man wird deshalb häufig die erwähnten Peripheriegeräte, wie schnelle Lochstreifenleser und -stanzer, Floppy-Disk, Bildschirmgerät usw., benötigen. Unter Floppy-Disk versteht man ein relativ schnelles und trotzdem preiswertes externes Speichermedium. Als Datenträger dient eine flexible Kunststoffscheibe mit einer magnetischen Schicht. Das Speichervermögen liegt etwa bei 2 Millionen bit. Wenn Massenspeicher, wie Platte oder Band, zur Verfügung stehen, sind auch entsprechende **Betriebssysteme** zur Datenverwaltung, zur Organisation von Datentransfers usw. erforderlich.
Es gibt jedoch auch Möglichkeiten, diese Investitionen zu vermeiden. Wenn im Labor ein Prozeßrechnersystem zur Verfügung steht, kann dieses auch zum Entwickeln der Programme für Mikrocomputer verwendet werden. Voraussetzung ist, daß ein **Cross-Assembler** zur Verfügung steht. Das ist ein Assembler, der Programme für einen anderen Rechner, als mit dem er selbst läuft, assembliert. Solche Cross-Assembler sind für eine Reihe von weitverbreiteten Prozeßrechner- und Mikrocomputertypen erhältlich.
Besonders schnelles Arbeiten ist möglich, wenn der Prozeßrechner und der Mikrocomputer durch eine Datenleitung miteinander verbunden werden, so daß die Programme einfach und schnell ausgetauscht werden können.
Eine weitere Möglichkeit besteht darin, den Großrechner der Firma (wenn ein solcher vorhanden ist) zur Programmentwicklung zu verwenden. Hierfür sind natürlich ebenfalls Cross-Assembler erforderlich, eventuell auch ein **Simulator**. Das ist ein Programm, das den Mikrocomputer auf dem Großrechner simuliert, so daß die Programme mit diesem simulierten Mikrorechner getestet werden können. Auf diese Art lassen sich Programme ohne Mikrocomputer entwickeln.
Wer keinen Rechner zur Verfügung hat, kann sich kommerzieller Time-Sharing-Dienste bedienen. Hier stehen die erforderlichen Programme zur Verfügung, so daß der Benutzer seine Programme entwickeln kann.
Zur Integration von Programm und Gerät wird man jedoch in den seltensten Fällen ohne ein einfaches Mikrorechner-Laborsystem auskommen, das dann jedoch die zusätzlichen Peripheriegeräte zur Beschleunigung des Arbeitsablaufes nicht erhalten muß.
Nachfolgend sollen die Aufgaben und die Arbeitsweisen der erwähnten Hilfsprogramme erläutert werden.

7.2.1 Editor

Der Editor ist ein Programm zur Textverarbeitung. Der zu verarbeitende Text wird in den Speicher des Rechners eingelesen. Dies geschieht über den Bedienungsfernschreiber, über Lochstreifen oder andere Datenträger. Solange der Text im Speicher des Rechners steht, sind beliebige Manipulationen am Text möglich.
Typische Editorfunktionen sind:
- Auslisten des Textes oder Textteilen
- Einfügen neuer Textzeilen an beliebigen Stellen
- Löschen beliebiger Textzeilen
- Änderungen innerhalb einer Textzeile (Löschen bzw. Einfügen von Buchstaben)
- Durchsuchen des gespeicherten Textes nach bestimmten Zeichenkombinationen (z.B. zur schnelleren Lokalisation einer fehlerhaften Stelle in einem längeren Programm)
- Umordnen des Textes (z.B. das Bewegen einiger Zeilen des Programms nach vorn oder hinten)

Der Editor hat zur Steuerung dieser Funktionen einen Satz von Befehlen, die über den Bedienungsfernschreiber eingegeben werden ebenso wie die Textänderungen selbst. Die Textzeilen werden häufig im Editor durchnummeriert, so daß Textstellen eindeutig gekennzeichnet werden können.
Besonders schnelles Arbeiten ist mit Sichtgeräten möglich, da hierbei der Text während des Arbeitens laufend auf dem Sichtgerät ausgelistet werden kann, so daß Änderungen leicht lokalisiert und kontrolliert werden können. Wenn der Text fertig ediert ist, wird er auf einem Datenträger (z.B. Lochstreifen) ausgegeben und kann dann mit dem Rechner weiterverarbeitet werden.
Für die Textausgabe wie auch für die Eingabe wird in Mikrorechnersystemen wie auch in Prozeßrechnersystemen überwiegend der ASCII-Code (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) verwendet. In diesem Code ist jedem Buchstaben und einer Reihe von Sonderzeichen jeweils eine 8-bit-Binärzahl zugeordnet. Die Zuordnung zeigt Tabelle 7.2.1.1.

Zeichen	ASCII hexadezimal	Binärcode
Wagenrücklauf	0 D	0 0 0 0 1 1 0 1
Zeilenvorschub	0 A	0 0 0 0 1 0 1 0
RUB OUT	7 F	0 1 1 1 1 1 1 1
!	2 1	0 0 1 0 0 0 0 1
"	2 2	0 0 1 0 0 0 1 0
#	2 3	0 0 1 0 0 0 1 1
$	2 4	0 0 1 0 0 1 0 0
%	2 5	0 0 1 0 0 1 0 1
&	2 6	0 0 1 0 0 1 1 0
'	2 7	0 0 1 0 0 1 1 1
(	2 8	0 0 1 0 1 0 0 0
)	2 9	0 0 1 0 1 0 0 1
*	2 A	0 0 1 0 1 0 1 0
+	2 B	0 0 1 0 1 0 1 1
,	2 C	0 0 1 0 1 1 0 0
−	2 D	0 0 1 0 1 1 0 1
.	2 E	0 0 1 0 1 1 1 0
/	2 F	0 0 1 0 1 1 1 1
:	3 A	0 0 1 1 1 0 1 0
;	3 B	0 0 1 1 1 0 1 1
<	3 C	0 0 1 1 1 1 0 0
=	3 D	0 0 1 1 1 1 0 1
>	3 E	0 0 1 1 1 1 1 0
?	3 F	0 0 1 1 1 1 1 1
[	5 B	0 1 0 1 1 0 1 1
/	5 C	0 1 0 1 1 1 0 0
]	5 D	0 1 0 1 1 1 0 1
↑	5 E	0 1 0 1 1 1 1 0
←	5 F	0 1 0 1 1 1 1 1
@	4 0	0 1 0 0 0 0 0 0
Leerstelle	2 0	0 0 1 0 0 0 0 0
0	3 0	0 0 1 1 0 0 0 0
1	3 1	0 0 1 1 0 0 0 1
2	3 2	0 0 1 1 0 0 1 0
3	3 3	0 0 1 1 0 0 1 1
4	3 4	0 0 1 1 0 1 0 0
5	3 5	0 0 1 1 0 1 0 1
6	3 6	0 0 1 1 0 1 1 0
7	3 7	0 0 1 1 0 1 1 1
8	3 8	0 0 1 1 1 0 0 0
9	3 9	0 0 1 1 1 0 0 1
A	4 1	0 1 0 0 0 0 0 1
B	4 2	0 1 0 0 0 0 1 0
C	4 3	0 1 0 0 0 0 1 1

Zeichen	ASCII hexadezimal	Binärcode
D	4 4	0 1 0 0 0 1 0 0
E	4 5	0 1 0 0 0 1 0 1
F	4 6	0 1 0 0 0 1 1 0
G	4 7	0 1 0 0 0 1 1 1
H	4 8	0 1 0 0 1 0 0 0
I	4 9	0 1 0 0 1 0 0 1
J	4 A	0 1 0 0 1 0 1 0
K	4 B	0 1 0 0 1 0 1 1
L	4 C	0 1 0 0 1 1 0 0
M	4 D	0 1 0 0 1 1 0 1
N	4 E	0 1 0 0 1 1 1 0
O	4 F	0 1 0 0 1 1 1 1
P	5 0	0 1 0 1 0 0 0 0
Q	5 1	0 1 0 1 0 0 0 1
R	5 2	0 1 0 1 0 0 1 0
S	5 3	0 1 0 1 0 0 1 1
T	5 4	0 1 0 1 0 1 0 0
U	5 5	0 1 0 1 0 1 0 1
V	5 6	0 1 0 1 0 1 1 0
W	5 7	0 1 0 1 0 1 1 1
X	5 8	0 1 0 1 1 0 0 0
Y	5 9	0 1 0 1 1 0 0 1
Z	5 A	0 1 0 1 1 0 1 0

Tab. 7.2.1.1
ASCII-Code

Wie aus dem Binärcode in Tab. 7.2.1.1 zu ersehen ist, ist das linksstehende bit immer 0. Es gibt auch Versionen des ASCII-Codes, bei denen das linke bit immer auf 1 gesetzt ist. In manchen Geräten wird das linksstehende bit als Prüf-bit für die Datenübertragung benutzt (Paritäts-bit).
Nachfolgend wollen wir die Anwendung des ASCII-Codes bei Lochstreifen als Datenträger näher betrachten:
Auf einem Lochstreifen werden Daten in Form von Gruppen eingestanzter Löcher festgehalten. Man spricht bei einem Lochstreifen von Kanälen und von Spalten. Die Kanäle verlaufen über die Länge des Streifens, die Spalten quer (Bild 7.2.1.1).

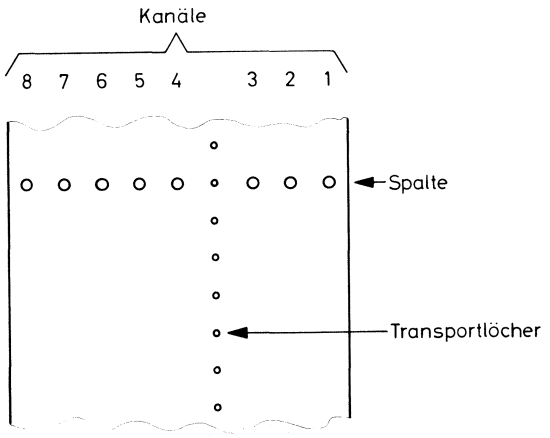


Bild 7.2.1.1
Einteilung des 8-Kanal-Lochstreifens

Bei Mikro- und Prozeßrechnern werden fast ausschließlich 8-Kanal-Lochstreifen verwendet. Die 8 Kanäle werden den 8 Stellen eines ASCII-Codewortes zugeordnet, so daß in einer Spalte ein Zeichen steht (Bild 7.2.1.2).

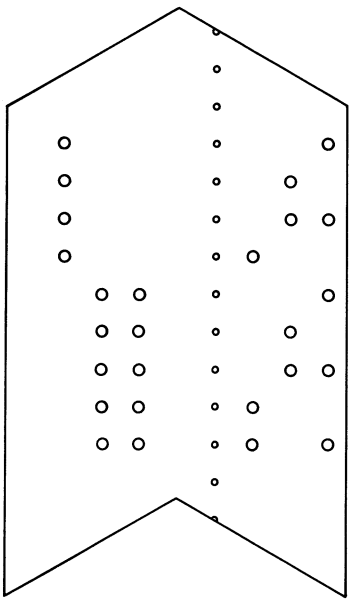


Bild 7.2.1.2
Lochstreifen mit ASCII-Zeichen

Bei Lochstreifen werden also bit 0 in Kanal 1 und bit 7 in Kanal 8 gestanzt.

7.2.2 Assembler

In den bisherigen Programmbeispielen haben wir bereits eine sogenannte Assemblersprache verwendet, ohne diese Sprache näher zu definieren und zu begründen. Befehle, die im reinen Maschinencode angegeben werden, sind natürlich weitgehend unleserlich und unverständlich. Aus diesem Grunde haben wir praktisch von Anfang an mit mnemonischen Abkürzungen für die einzelnen Befehle gearbeitet. Wir haben unsere Programme zunächst mit diesen verhältnismäßig leicht verständlichen und zu merkenden Ausdrücken aufgeschrieben. Danach haben wir dann mit Hilfe einer Tabelle die eigentlichen Maschinenbefehle angegeben. Dieser Übersetzungsvorgang ist im Grunde eine rein mechanische Tätigkeit, die sich ohne weiteres automatisieren läßt.
Wir wollen jetzt noch einen Schritt weitergehen. Wenn z.B. ein längeres Programm mit mehreren Sprungadressen gegeben ist, so wird es schwierig, sich diese Adressen zu merken. In einem solchen Falle ist es von Vorteil, diesen Adressen Namen zu geben, also sog. symbolische Adressen zu verwenden. Diese Namen, die die Funktion eines Sprunges oder eines Operanden ausdrücken, erleichtern die Lesbarkeit eines Programms. Wir wollen dies an einem Programmbeispiel näher erläutern (bezogen auf den hypothetischen Mikrorechner):

```
            ; PROGRAMM ZUM DURCHSUCHEN EINER LISTE
            ; ALLE ELEMENTE DER LISTE WERDEN
            ; MIT EINEM SOLLWERT VERGLICHEN
            ; BEI ÜBEREINSTIMMUNG STEHT NUMMER
            ; DES ELEMENTES IN R2
.ORG 2 0
              SUBR R2, R2
              LOAD R3, LISTE    ; LADE STARTADR.
WIEDER:       INCR R2
              LOAD R1, @ R3 ↑   ; HOLE ELEMENT
              SUBM R1, SOLL     ; VERGLEICHE
              JMPZ ENDE         ; GEFUNDEN
              MOVE R1, R3       ; FERTIG?
```

```
        SUBM R1, MAP1
        JMPZ FEHLER
        JUMP WIEDER
FEHLER: HALT            ; IN WIRKLICHKEIT WÜRDE
ENDE:   HALT            ; DAS PROGRAMM HIER FORTGESETZT
LISTE:  4 0             ; ADRESSE DER LISTE
SOLL:   1 2             ; SOLLWERT
MAP1:   4 6             ; MAX. ADRESSE PLUS 1
.END:
```

Dieses Programm wird z.B. mit Hilfe des Editors geschrieben und dann im ASCII-Code auf einem Datenträger gespeichert. Es handelt sich dabei um das **Quellenprogramm**. In diesem Programm finden wir bekannte Rechnerbefehle wie z.B. SUBR R2, R2, die übersetzt werden sollen. Daneben gibt es Ausdrücke wie .ORG 2 0 und Begleittext wie z.B. die Überschrift. Der Ausdruck .ORG 2 0 ist kein Rechnerbefehl sondern ein Befehl des Programmierers an den Assembler. Hierdurch wird dem Assembler mitgeteilt, daß die Adresse 2 0 die Startadresse des Programms sein soll. Auch bei dem Ausdruck .END handelt es sich um einen Befehl an den Assembler, der das Programmende anzeigt. Auch das Semikolon ; ist aus dieser Gruppe. Dieses Sonderzeichen im Programm signalisiert dem Assembler, daß aller Text von nun an bis zum Ende der Zeile Kommentar ist, der nicht übersetzt werden darf. Jeder Assembler hat einer Reihe solcher Befehle, die den Übersetzungsvorgang steuern. Für die vielen in der Praxis verwendeten Assembler gibt es allerdings keine einheitliche Nomenklatur, so daß man sich mit dem jeweiligen Assembler vertraut machen muß. Der erste Rechnerbefehl in dem Programm ist:

SUBR R2, R2

Damit der Assembler diesen Befehl übersetzen kann, benötigt er eine Tafel aller mnemonischen Ausdrücke, die **Tafel der permanenten Symbole**. In dieser Tafel sind alle mnemonischen Rechnerbefehle, z.B. in Form ihrer ASCII-Darstellung, enthalten sowie die zugehörige Übersetzung in den Maschinencode.
Der Eintrag in die permanente Symboltafel für den SUBR-Befehl könnte so aussehen:

Adresse	Inhalt	Kommentar
n	5 3	; S
n + 1	5 5	; U
n + 2	4 2	; B
n + 3	5 2	; R
n + 4	2 0	; OP-Code

Zunächst sind hintereinander die ASCII-Werte für SUBR abgespeichert und dann der zugehörige OP-Code, nämlich 2 0. Die Übersetzung des ersten Befehles verläuft folgendermaßen:

Das Assemblerprogramm beginnt, eine neue Zeile einzulesen. Das Ende der vorhergehenden Zeile hat er an dem Zeichen „Wagenrücklauf – neue Zeile" erkannt. Zunächst kommen die Leerstellen (ASCII-Code 2 0). Diese werden ignoriert. Wenn dann das erste Wort kommt, wird es vom Assembler bis zur Leerstelle gelesen. Nun durchsucht der Assembler die Symboltafel, ob das soeben eingelesene Wort enthalten ist. Ist eine Übereinstimmung gefunden, ist damit der OP-Code und die erforderliche Weiterverarbeitung des Befehles festgelegt. In unserem Beispiel steht jetzt der OP-Code 2 0 fest. Außerdem ist festgelegt, daß die nächsten Zeichen bis zum Komma das DST-Register (hier R2), die Zeichen nach dem Komma das SRC-Register (ebenfalls R2) spezifizieren. Diese Registerspezifikationen können genauso durch Nachsuchen in der permanenten Symboltafel übersetzt werden. Dabei findet der Assembler, daß dem DST-Register R2 der Maschinencode 2 und dem SRC-Register der Maschinencode 8 zugeordnet ist. Eine ODER-Verknüpfung aller bisher gefundenen Teilergebnisse ergibt den endgültigen Maschinencode $2\ A_{16} \triangleq 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0_2$.
Beispiel:

OP-Code 0 0 1 0 0 0 0 0
DST-Register 0 0 0 0 0 0 1 0
SRC-Register V 0 0 0 0 1 0 0 0
0 0 1 0 1 0 1 0_2 = 2 A_{16} (Ergebnis der ODER-Verknüpfung)

Auch die Befehle an den Assembler werden durch Vergleich mit der Tafel der permanenten Symbole identifiziert. Diese Befehle werden dann natürlich nicht in Maschinenbefehle übersetzt, sie steuern vielmehr den Ablauf der Übersetzung.
Auf diese Art und Weise kann das gesamte Programm übersetzt werden, mit Ausnahme der symbolischen Adressen. Die Maschinenbefehle können vom Assembler unmittelbar verarbeitet werden, da im Assembler eine Liste aller Rechnerbefehle enthalten ist. Die vom Benutzer definierten Namen können in dieser Liste natürlich nicht enthalten sein, sie werden erst in dem zu übersetzenden Programm spezifiziert. Deshalb muß vor der eigentlichen Übersetzung ein weiterer Schritt durchgeführt werden, nämlich das Anlegen einer Liste mit allen im zu übersetzenden Programm vorkommenden symbolischen Adressen. Bei diesem **ersten Schritt** oder **ersten Durchlauf** wird das gesamte Programm eingelesen. Die einzelnen Befehle werden nicht vollständig übersetzt, es wird lediglich überprüft, wie viele Speicherwörter (Bytes) jeder Befehl benötigt. Damit kann der Assembler die Adresse festlegen, an der später ein bestimmter Befehl abgespeichert wird. In unserem Beispiel kommt der Befehl SUBR R2, R2 in die Adresse 2 0. Da es sich um einen 1-Byte-Befehl handelt, kommt der Befehl LOAD R3, LISTE in Adresse 2 1. Dieser Befehl benötigt 2 Bytes, so daß der nächste Befehl in Adresse 2 3 kommt. Dieser Adresse 2 3 ist nun eine symbolische Adresse, nämlich WIEDER zugeordnet. Der Assembler erkennt dies am Doppelpunkt, die Zeichen links vom Doppelpunkt stellen die symbolische Adresse dar. Diese Zeichen werden nun wie die permanenten Symbole in eine Liste eingetragen. Dieser Eintrag wäre in unserem Beispiel:

Adresse	Inhalt	Kommentar
.	.	
.	.	
m	5 7	; W
m + 1	4 9	; I
m + 2	4 5	; E
m + 3	4 4	; D
m + 4	4 5	; E
m + 5	5 2	; R
m + 6	2 3	; zugeordneter Adressenwert
m + 7	:	; nächstes Symbol

Im ersten Durchlauf werden so sämtliche symbolischen Adressen und die ihnen zugeordneten absoluten Adressen in eine Symboltafel eingetragen. Nun kann in einem **zweiten Durchlauf** die eigentliche Übersetzung durchgeführt werden.
Wir wollen dies am Beispiel der nächsten Instruktionen nachvollziehen. Zunächst wird der LOAD-Befehl eingelesen und in der Symboltafel aufgesucht. Damit steht der OP-Code 8 0 und die weitere Behandlung des Befehles fest. Die nächsten Zeichen bis zum Komma spezifizieren das DST-Register, in unserem Falle R3 und somit den Code 3 (binär 0 0 0 0 0 0 1 1). Die Zeichen hinter dem Komma spezifizieren die Adressierungsart. Bei diesem Befehl ist kein Sonderzeichen zur Angabe einer Adressierungsart vorhanden, d.h., es handelt sich um absolute Adressierung mit Code 4 (binär 0 0 0 0 0 1 0 0). Damit ist dieses Rechnerwort festgelegt. Die ODER-Verknüpfung der Teilergebnisse ergibt den endgültigen Befehl Inhalt von Adresse 2 1, nämlich 8 7 (binär 1 0 0 0 0 1 1 1). Damit ist dieser Befehl aber noch nicht vollständig übersetzt. Absolute Adressierung bedeutet, daß im nächsten Byte (Adresse 2 2) nicht der nächste Befehl steht, sondern die Adresse des Operanden für den LOAD-Befehl. Diese Adresse folgt unmittelbar auf das Komma. Wenn hier eine Zahl stehen würde (z.B. LOAD R3, 1 6), würde die Zahl 1 6 unmittelbar in die nächste Adresse assembliert. In unserem Programmbeispiel steht hier ein Symbol, nämlich LISTE. Der Assembler durchsucht die Symboltafel nach diesem Symbol und findet dort, daß LISTE den Wert 3 2 hat. Den Wert 3 2 für Liste hat der Assembler im ersten Durchlauf festgelegt. Damit wird 3 2 in das nächste Speicherwort assembliert. Der vollständig übersetzte Befehl lautet also:

Adresse	Inhalt	Befehl
.	.	
.	.	
2 1	8 3	LOAD R3, LISTE
2 2	3 2	
.	.	
.	.	

Die vom Benutzer definierten Symbole werden also vom Assembler folgendermaßen behandelt:

1. Durchlauf
Zunächst sucht der Assembler die Definition aller Symbole (in unserem Fall wird die Definition durch einen Doppelpunkt gekennzeichnet) und legt eine Symboltafel an.

2. Durchlauf
Wenn im Programm ein Symbol verwendet wird, ersetzt der Assembler jedes Symbol durch den in der Symboltafel zu findenden Zahlenwert und führt damit die Übersetzung durch.

Nachfolgend noch einige Zusatzbemerkungen zu den symbolischen Adressen:
– Für jedes Symbol ist in der Symboltafel ein Platz bestimmter Größe reserviert, d.h., die Anzahl der Zeichen im Symbol ist begrenzt. Eine gängige Größe ist maximal 6 Zeichen pro Symbol.
– Die Länge der Symboltafel ist bestimmt durch die Anzahl der vom Programmierer verwendeten Symbole. In einem praktischen Mikrorechnersystem ist die permanente Symboltafel ein Teil des Assemblerprogramms, das z.B. vom Hersteller in einem ROM geliefert werden kann. Die Tafel der Benutzersymbole muß auf alle Fälle vom Assembler in einem Schreib/Lesespeicher angelegt werden. In einem Rechnersystem, das zur Programmentwicklung verwendet werden soll, muß ein genügend großer Schreib/Lesespeicher enthalten sein, damit der Assembler die erforderliche Symboltafel anlegen kann.

Der Hauptvorteil von symbolischen Adressen macht sich erst in Verbindung mit einem Assemblerprogramm bemerkbar. Ist z.B. ein Programm mit absoluten Adressen übersetzt und wird dann festgestellt, daß an einer Stelle zusätzliche Instruktionen eingefügt werden müssen, so verschieben sich natürlich alle nachfolgenden Adressen. Daraufhin muß das ganze Programm durchgesehen werden, damit die Adressenänderung berücksichtigt werden kann. Wenn nur symbolische Adressen verwendet werden, führt der Assembler diese Adressenkorrektur automatisch durch. Es wird im Quellenprogramm lediglich die Instruktion eingefügt. Im ersten Durchlauf legt der Assembler automatisch die korrigierte Symboltafel an und übersetzt somit im zweiten Durchlauf richtig. Beim praktischen Programmieren werden deshalb fast ausschließlich symbolische Adressen verwendet.
In diesem Zusammenhang soll noch einmal auf den Unterschied zwischen dem **Wert** einer symbolischen Adresse und ihrem **Inhalt** hingewiesen werden. Die symbolische Adresse LISTE hat den Wert 3 2. Überall wo der Assembler das Symbol LISTE findet, ersetzt er es durch die Zahl 3 2. Der Inhalt dieser symbolischen Adresse dagegen ist in unserem Programmbeispiel 4 0. Dieser Inhalt hat nichts zu tun mit dem Wert des Symbols.
Der Befehl

```
8 7 LOAD R3, LISTE
3 2
```

verwendet absolute Adressierung, d.h., der Inhalt der Adresse LISTE, also die Zahl 4 0, wird in R3 geladen. Dieser Befehl ist identisch mit dem Befehl:

```
8 7 LOAD R3, 3 2
3 2
```

Im Unterschied hierzu würde der Befehl

```
                    8 3 LOAD R3, # LISTE
                    3 2
```

die Zahl 3 2 in R3 laden. Dieser Befehl ist identisch mit dem Befehl:

```
                    8 3 LOAD R3, # 3 2
                    3 2
```

Eine weitere Aufgabe des Assemblers neben der Übersetzung des Programms ist das Ausgeben einer Programmliste. Eine solche Programmliste ist nachfolgend dargestellt:

```
                              ; PROGRAMM ZUM DURCHSUCHEN EINER LISTE
                              ; ALLE ELEMENTE DER LISTE WERDEN
                              ; MIT EINEM SOLLWERT VERGLICHEN
                              ; BEI ÜBEREINSTIMMUNG STEHT NUMMER
                              ; DES ELEMENTES IN R2
                    .ORG 20
20      2A                    SUBR R2, R2
21      87                    LOAD R3, LISTE     ; LADE STARTADR.
22      32
23      62        WIEDER:     INCR R2
24      8D                    LOAD R1, @ R3 ↑    ; HOLE ELEMENT
25      A5                    SUBM R1, SOLL      ; VERGLEICHE
26      33
27      E1                    JMPZ ENDE          ; GEFUNDEN
28      31
29      0D                    MOVE R1, R3        ; FERTIG?
2A      A5                    SUBM R1, MAP1
2B      34
2C      E1                    JMPZ FEHLER        ; NICHT GEFUNDEN
2D      30
2E      E0                    JUMP WIEDER
2F      23
30      00        FEHLER:     HALT               ; IN WIRKLICHKEIT WÜRDE DAS
31      00        ENDE:       HALT               ; PROGRAMM HIER FORTGESETZT
32      40        LISTE:      4 0                ; ADRESSE DER LISTE
33      12        SOLL:       1 2                ; SOLLWERT
34      46        MAP1:       4 6                ; MAX. ADRESSE PLUS 1
                    .END
```

Das Quellenprogramm mit allen Kommentaren wird ausgedruckt, zusätzlich stehen ganz links die Adressen und rechts daneben die Maschinenbefehle.
Diese Liste ist ein wichtiges Hilfsmittel zum Programmtest, da aus ihr der Zusammenhang zwischen dem Quellenprogramm und dem tatsächlichen Inhalt des Speichers hervorgeht.

Diese Liste wird häufig in einem dritten Durchlauf des Assemblers erstellt, d.h., das Quellenprogramm muß ein drittes Mal eingelesen werden. Im Prinzip ist es möglich, die Liste schon beim zweiten Durchlauf, also bei der eigentlichen Übersetzung auszugeben. Das setzt aber voraus, daß 2 unabhängige Ausgabegeräte zur Verfügung stehen, eines für das übersetzte Programm (Objektprogramm) und eines für die Liste. Verwendet man z.B. einen Fernschreiber mit Lochstreifenstanzer, so sind normalerweise Stanzer und Druckwerk mechanisch gekoppelt. In diesem Fall gibt man im zweiten Durchlauf das übersetzte Programm auf Lochstreifen aus, der gleichzeitig entstehende Ausdruck ist wertlos, die Liste wird im dritten Durchlauf ausgedruckt. Dieses 3malige Einlesen des Quellenprogramms erfordert bei einem langsamen Leser und bei längeren Programmen erhebliche Zeit.
Eine weitere wichtige Aufgabe des Assemblers ist die Meldung von Programmfehlern. Dabei ist zu beachten, daß ein Assembler grundsätzlich nur Verstöße gegen die definierte Assemblersprache entdecken kann, wie z.B. Syntaxfehler, falsche Befehlsbezeichnungen usw., nicht jedoch logische Fehler im Programm. Eine Fehlermeldung eines guten Assemblers enthält

möglichst viele Informationen über den Fehler, z.B. um welche Art von Fehler es sich handelt, an welcher absoluten Adresse im Programm der Fehler vorkommt, wie viele Instruktionen hinter der letzten symbolischen Adresse der Fehler liegt usw. Diese Informationen helfen dem Programmierer, den Fehler mit Hilfe des Editorprogramms rasch zu lokalisieren und zu korrigieren. Nachfolgend sind typische Fehlerarten, die der Assembler melden kann, angegeben:
- Undefiniertes Symbol. Im Programm wird ein Symbol verwendet, das nirgends definiert ist, so daß der Assembler seinen Wert nicht kennt
- Symbol mehrmals definiert. Hierbei entstehen Zuordnungsschwierigkeiten, da manchmal die letzte Definition als gültig angenommen wird. Doppeldefinitionen können zu schwer lokalisierbaren Programmfehlern führen
- Illegale Adressierungsart
- Illegales Zeichen
- Illegale Oktalzahl, z.B. 9 in einer Oktalzahl
- Zu viele oder zu wenige Operanden
- Zahl zu groß, z.B. 300_{10} in 8 bit
- Programmende-Zeichen fehlt
- Symboltafel voll. Der verfügbare Speicherplatz reicht nicht aus für die Symboltafel

Was bisher erläutert wurde, sind die Grundfunktionen eines Assemblers. Daneben haben in der Praxis Assembler häufig eine Reihe von Extras, die das Arbeiten leichter und sicherer machen:
- Daten können in verschiedenen Darstellungen angegeben werden, z.B. Zahlen in dezimal, hexadezimal, oktal, Text in ASCII-Code oder als Zeichen
- Der Assembler kann arithmetische und logische Ausdrücke als Operanden akzeptieren, d.h., der Assembler wertet den Ausdruck aus und verwendet das Ergebnis als Operanden
- Es wird automatisch eine geeignete Adressierungsart gewählt. Hierbei ist wichtig, daß in der Programmliste klar und eindeutig zu erkennen ist, welche Adressierungsart gewählt wurde, und daß der Programmierer die automatische Wahl gegebenenfalls ausschalten kann
- Makroinstruktionen können definiert werden. Makroinstruktionen sind Befehle, die vom Benutzer definiert werden und die einer ganzen Reihe von Maschinenbefehlen entsprechen. Sie werden verwendet, um häufig wiederkehrende Operationen mit einem Befehl im Programm ausführen zu können. Jedesmal wenn der Assembler diesen Befehl im Programm findet, substituiert er ihn durch den entsprechenden Maschinencode. Man spricht daher auch von In-Line-Subroutines im Gegensatz zu echten Subroutines, wo der Code nur einmal vorhanden ist, man aber mehrere Sprungbefehle braucht, um diesen Code mehrmals auszuführen. Makros sind nur sinnvoll, wenn auch Argumente in der Makroinstruktion definiert werden können, die dann an der entsprechenden Stelle in den Maschinencode umgesetzt werden. Makros können wie Subroutines verschachtelt werden. Die Makromöglichkeiten eines Assemblers lassen sich dadurch beurteilen, wie viele Makros definiert werden können, wie lang die Makros sein können, wie viele Argumente erlaubt sind und wie viele Niveaus verschachtelbar sind

Diese und andere Möglichkeiten eines Assemblers können das Arbeiten mit dem Assembler erleichtern und beschleunigen. Viel wichtiger ist aber, daß die Grundfunktionen des Assemblers zuverlässig ausgeführt werden, daß das Format leicht zu benutzen ist und daß der Programmierer die Möglichkeit hat, die verschiedenen Felder in einem Befehl so anzuordnen, wie er es für sinnvoll hält. Es sollte nicht sein, daß ein Assembler einen Befehl zurückweist, weil irgendein Feld einer Instruktion in der falschen Spalte beginnt.

7.2.3 Ladeprogramm

Dieses Programm hat die Aufgabe, das Objektprogramm von einem externen Speichermedium, z.B. einem Lochstreifen, in den Schreib/Lesespeicher des Mikrorechners zu laden. Diese Grundfunktion ist sehr einfach. Ladeprogramme, die nur diese Aufgabe erfüllen, sind deshalb auch sehr einfache Programme.
Daneben gibt es Ladeprogramme, die noch zusätzliche Möglichkeiten bieten. Manche Assembler produzieren einen Maschinencode, in dem die Startadresse nicht endgültig festgelegt ist (Relocatable Code). Erst beim Ladevorgang werden die Adressen im Programm so

festgelegt, daß es in einen bestimmten Bereich geladen werden kann. Diese Methode erleichtert das gemeinsame Laden von verschiedenen Programmen, die unabhängig voneinander assembliert werden.
Noch einen Schritt weiter geht das **Linkage Editing** beim Laden. Hier führt das Ladeprogramm die Verbindung von mehreren Objektprogrammen untereinander aus, die z.B. gemeinsame symbolische Adressen (Global Variables) verwenden. Linkage Editing erfordert entsprechende Assembler und Lader, die in der Lage sind, die benötigten Informationen auszutauschen.

7.2.4 Fehlersuchprogramme (Debugging-Routines)

Ohne besondere Fehlersuchprogramme bietet ein Mikrorechner-Laborsystem nur Fehlersuchmöglichkeiten, die z.B. auch der ITT MP-Experimenter bietet. D.h. Single-Step-Betrieb, um schrittweise durch ein Programm zu gehen und so die Speicherinhalte zu kontrollieren und gegebenenfalls zu korrigieren. Eine einfache Testmethode ist, an geeigneten Stellen im Programm HALT-Befehle einzubauen und nach einem HALT-Befehl Speicher- und Registerinhalte sowie den Prozessorstatus zu überprüfen. Danach kann durch Eingabe des ursprünglichen Befehles der HALT-Befehl ersetzt werden und das Programm bis zum nächsten HALT-Befehl überprüft werden. Diese Methode ist natürlich sehr zeitraubend.
Ein Fehlersuchprogramm erleichtert diese Arbeit erheblich. Dabei wird das Fehlersuchprogramm gemeinsam mit dem Anwenderprogramm geladen, d.h., der Speicher des verwendeten Mikrorechners muß groß genug sein, um beide Programme gleichzeitig fassen zu können. Das Fehlersuchprogramm erlaubt eine Fehlersuche vom Fernschreiber aus, ohne dafür Schalter oder Lampen zu benötigen. Die wichtigsten Funktionen, die eine gute Debugging-Routine ausführen muß, sind nachfolgend aufgeführt:
- Auslisten und Ändern der Inhalte von Arbeitsregister und Flags
- Auslisten und Ändern der Inhalte beliebiger Speicheradressen
- Starten von Programmen bei beliebigen Adressen
- Einfügen von künstlichen Haltepunkten in das Programm
- Fortsetzen des Programms nach einem Haltepunkt

Der Programmtest mit einer solchen Debugging-Routine verläuft folgendermaßen: Nach dem ersten Laden Programms werden mit der Debugging-Routine die Inhalte einiger Speicheradressen überprüft. Anhand der Programmliste besteht so die Möglichkeit festzustellen, ob alle Programmteile geladen sind. Dann wird nach der einen Gruppe von Befehlen ein Haltepunkt eingebaut und das Programm von der Debugging-Routine gestartet. Wenn das Programm den Haltepunkt erreicht hat, wird das Anwenderprogramm unterbrochen und die Kontrolle der Debugging-Routine übergeben. Der Programmierer kann jetzt Speicher- und Registerinhalte überprüfen und so feststellen, ob das Programm bis jetzt ordnungsgemäß funktioniert hat. Ist das der Fall, kann der Haltepunkt im Programm weiterbewegt und die Ausführung des Programms fortgesetzt werden. Ist das Programm nicht in Ordnung, kann eine Korrektur sofort mit Hilfe der Debugging-Routine erfolgen. Nach einer Korrektur ist es sinnvoll, möglichst sofort das Quellenprogramm zu berichtigen und neu zu assemblieren, so daß wieder eine Programmliste, die auf dem neuesten Stand ist, zur Verfügung steht.
Man bewegt also den Haltepunkt schrittweise durch das gesamte Programm, falls erforderlich an kritischen Stellen von Befehl zu Befehl, bis das Programm vollständig getestet ist.

Eine Grundvoraussetzung für diese Art von Programmtest ist, daß das Programm in dieser Phase in einem Schreib/Lesespeicher gespeichert wird, da in einem Festwertspeicher keine künstlichen Haltepunkte eingebaut werden können und auch keine Korrekturen mit der Debugging-Routine durchführbar sind. Erst wenn das Programm getestet und in Ordnung ist, kann es in Festwertspeicher abgespeichert werden.
Neben diesen Grundfunktionen, die für eine brauchbare Debugging-Routine unerläßlich sind, gibt es noch Versionen, die weitergehende Möglichkeiten bieten:
- Eingabe von Befehlen in Assemblersprache. Ein in der Debugging-Routine enthaltener Miniassembler führt die Übersetzung in den Maschinencode durch
- Verschiedene Haltepunkte, so daß z.B. in jeden Ast einer Verzweigung ein Haltepunkt gesetzt werden kann

– Mehrfaches Vorbeifahren an einem Haltepunkt. Diese Möglichkeit ist besonders wertvoll, um Schleifen zu testen, die sehr häufig durchlaufen werden und deren Aussprungbedingung vielleicht nicht korrekt ist. Hier läuft ein Programm eine zu spezifizierende Anzahl mal an einem Haltepunkt vorbei, bis dann die Kontrolle zur Debugging-Routine geht
– Suchen im Speicher nach gegebenen Inhalten. Diese Möglichkeit ist besonders gut geeignet, um festzustellen, wo die Daten z.B. bei einem Adressierungsfehler geblieben sind oder wo ein Befehl sitzt, der ständig einen anderen ändert

Die erweiterten Möglichkeiten können das Fehlersuchen, vor allem bei komplizierten Fehlern beträchtlich erleichtern. Eine gute Debugging-Routine ist ein sehr wichtiges Programm, das hilft, eine sehr zeitraubende Phase der Programmentwicklung zu beschleunigen.

7.2.5 Simulator

Ein Simulator ist ein **Cross-Computer-Programm**, also ein Programm, das einen bestimmten Rechner auf einem anderen Rechner simuliert. Es gibt Simulatorprogramme, die auf einem Großrechner einen Mikrorechner simulieren. Damit können auf dem Großrechner Programme für Mikrorechner entwickelt werden.
Bekanntlich erhält auch der ITT MP-Experimenter Simulationsprogramme, mit denen z.B. der einfache Rechner (System 4) und der hypothetische Rechner (System 5) simuliert werden.

7.2.6 Monitorprogramme und Betriebssysteme

Ein Mikrorechner ohne Programme ist selbst für die einfachsten Funktionen nicht brauchbar. Er kann keine Peripheriegeräte betreiben und in vielen Systemen noch nicht einmal die Schalterzustände einlesen und Lampen zur Anzeige bringen. Man kann dann Programme nur dadurch eingeben, daß man entsprechend programmierte ROMs in Fassungen steckt. Es gibt normalerweise nur eine Startadresse, an der die Programmausführung bei Betätigen der Starttaste beginnt. In den angesprochenen Mikrorechner-Laborsystemen sind deshalb Monitorprogramme enthalten, die es erlauben, vom Fernschreiber aus Befehle an den Rechner zu geben, andere Programme zu laden und zu starten. Häufig führen diese Monitorprogramme auch verschiedene Funktionen einer Debugging-Routine aus.
Ein einfaches Beispiel für ein Monitorprogramm für den ITT MP-Experimenter ist das System 6. Hier handelt es sich um ein Programm für den MP 8080, das es ermöglicht, Speicher- und Registerinhalte anzuschauen und zu ändern, Programme zu starten usw. Dieser Monitor ist eine Verbindung zwischen den am Experimentiergerät vorhandenen Peripheriegeräten (Schalter und Lampen) und dem 8080-Mikrorechner. Das Programm liest die Schalter, interpretiert die Funktionen, führt sie aus und betreibt die entsprechenden Anzeigelampen. Prinzipiell führt es die gleichen Funktionen aus, als wenn es die Befehle von einem Fernschreiber bekommen würde und die Ergebnisse dort ausdrucken würde.
Wenn in einem Rechnersystem Massenspeicher wie Band oder Platte vorhanden sind, werden die Aufgaben des Monitors vielfältiger. Das Monitorprogramm muß den Datentransfer zwischen Peripheriegeräten und Rechner steuern und die Organisation der Daten auf den Speichermedien durchführen. Dazu führt das Monitorprogramm ein Inhaltsverzeichnis (Directory) für die Daten auf jedem Medium, aus dem hervorgeht, welchen Namen eine Datei hat, wo sie auf der Platte oder dem Band steht, wie lang die Datei ist, welcher Art die Daten sind usw. Der Monitor, man spricht bei solchen Programmen häufig auch von Betriebssystemen (Operating Systems), steuert das Laden der Programme in den Rechner, das Abspeichern von Daten auf den Massespeichern usw. Das Laden eines Assemblerprogramms ist dann z.B. durch einen einfachen Befehl wie RUN ASSEMBLER möglich.
Solche Betriebssysteme sind heute von verschiedenen Herstellern für ihre Mikrorechnersysteme erhältlich. Leistungsfähige Betriebssysteme, wie sie bei Großrechnern üblich sind, die Time-Sharing-Betrieb, Rechnerkostenabrechnung usw. durchführen, werden bei Mikrorechnern nicht eingesetzt.

7.2.7 Hochsprachen, Compiler

Der Vorteil der Programmierung in einer Assemblersprache ist, daß der Programmierer die Leistungsfähigkeit eines Mikrorechners voll ausschöpfen kann, da er die Operationen des Rechners im Detail kontrolliert. Die Erfahrungen haben jedoch gezeigt, daß Programmierung in Assemblersprachen zeitraubend ist und viel Erfahrung erfordert. Bei Großrechnern und auch bei Prozeßrechnern geht deshalb der Trend schon lange zu den problemorientierten Hochsprachen, wie FORTRAN, ALGOL, PL1 und viele andere. Diese Sprachen verwenden eine Syntax, die mehr der normalen Sprache angenähert ist und deren Befehlsvorrat den Problemstellungen angepaßt ist, für die Sprachen entwickelt wurden. Diese Sprachen reflektieren nicht mit den Eigenschaften eines bestimmten Rechners, sie sind vielmehr standardisiert und im wesentlichen unabhängig von einem bestimmten Rechnertyp.
Die Übersetzung eines in Hochsprache geschriebenen Programms in den Maschinencode wird mit einem besonderen Übersetzungsprogramm, einem **Compiler**, durchgeführt. Solche Compiler werden in zunehmendem Maße auch für Mikrorechner angeboten, so daß auch für Mikrorechner Programme in Hochsprachen geschrieben werden können. Vorteile dieser Programmierung sind:
- Schnelleres Arbeiten
- Weniger Programmiererfahrungen erforderlich
- Bessere Lesbarkeit der Programme
- Einfachere Struktur der Programme

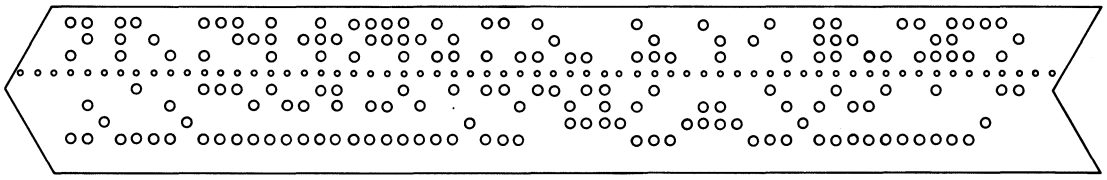
Der Nachteil der Programmierung in Hochsprachen ist, daß ein solches Programm normalerweise mehr Speicher benötigt als ein Assemblerprogramm für die gleiche Funktion und deshalb langsamer läuft. Bei einem konkreten Projekt sind deshalb Programmierkosten gegenüber Speicherkosten abzuwägen. Dieser Kompromiß kann je nach Stückzahl, Programmlänge, Produktlebensdauer usw. ganz unterschiedlich ausfallen. Der generelle Trend geht mit zunehmenden Programmierkosten und abnehmenden Speicherkosten sicher in Richtung eines zunehmenden Einsatzes von Hochsprachen.

Fragen zu Abschnitt 7.

1. Welche Laborgrundausrüstung (Minimalsystem) wird zur Entwicklung von Mikrorechnersystemen benötigt?

2. Was versteht man unter einem Editor?

3. Lesen Sie folgenden Lochstreifen!



4. Was verstehen Sie unter einem Assemblerprogramm?

5. Wodurch wird bei Verwendung eines Assemblers erreicht, daß nur die in mnemonischer Schreibweise angegebenen Befehle in den Maschinencode übersetzt werden und nicht der noch zusätzlich angegebene Kommentar sowie sonstige Bemerkungen?

6. Was verstehen Sie unter einer Tafel der permanenten Symbole?

7. Auf welche Weise werden die in einem edierten Quellenprogramm angegebenen symbolischen Adressen mit Hilfe eines Assemblers in absolute Adressen umgewandelt?

8. Welchen Wert und welchen Inhalt hat die symbolische Adresse MAP1 in dem Programm auf Seite 5.10?

9. Welche grundsätzliche Aufgabe hat ein Ladeprogramm?

10. Für welchen Zweck werden Debugging-Routines eingesetzt?

11. Welchen Zweck haben Monitorprogramme?

12. Für welchen Zweck werden Compiler benutzt?

8. Datentransfer in Mikrorechnersystemen

In diesem Abschnitt soll der Austausch von Daten zwischen den Baugruppen eines Mikrorechners und zwischen Mikrorechner und Peripherie näher erläutert werden. Diese Frage hat eine große Bedeutung für die praktische Auslegung von Systemen mit Mikrorechnern.

8.1 Bussystem eines Mikrorechners

Das Bussystem eines Mikrorechners verbindet die Baugruppen des Systems – Mikroprozessor, Speicher, Interface-Schaltungen – miteinander, so daß ein Datenaustausch zwischen den Baugruppen des Systems möglich ist. Die Struktur dieses Bussystems bestimmt die interne Kommunikation im Mikrorechner.
In einem Mikrorechner müssen im wesentlichen folgende Daten zwischen den Baugruppen des Systems übertragen werden:

1. Holen der Instruktion (des Befehles)
- Ausgabe der Adresse der Instruktion, also des Inhaltes des Programmzählers, zum Programmspeicher
- Übertragung der Instruktion vom Speicher in das Befehlsregister im Steuerwerk
- Übertragung von Signalen zur Steuerung des Zeitablaufes des Transfers zwischen Steuerwerk und Programmspeicher

2. Holen und Abspeichern von Operanden (Daten)
- Ausgabe der Adresse des Datenwortes zum Datenspeicher
- Übertragung des Datenwortes vom oder zum Datenspeicher
- Übertragung von Signalen zur Steuerung von Richtung und Zeitablauf des Datentransfers zwischen CPU und Datenspeicher

3. Dateneingabe, Datenausgabe
- Ausgabe von Signalen zur Auswahl einer Interface-Schaltung
- Übertragung eines Datenwortes zwischen CPU und Interface-Schaltungen
- Übertragung von Signalen zur Steuerung von Richtung und Zeitablauf des Transfers zwischen CPU und Interface-Schaltungen

Jedem dieser verschiedenen Datenpfade kann im Mikrorechner ein eigener Bus zugeordnet werden. In diesem Fall spricht man von **Dedicated Busses.**
Dieses Konzept benötigt eine Vielzahl von Verbindungen im System. Diesem Nachteil steht der Vorteil gegenüber, daß eine teilweise Überlappung der einzelnen Datentransfers erfolgen und so eine besonders hohe Arbeitsgeschwindigkeit des Rechners erreicht werden kann.
Dieses Konzept wird allerdings nur in Ausnahmefällen angewandt, da die Vielzahl der Verbindungen auch eine Vielzahl von Anschlüssen an den integrierten Schaltkreisen erfordert. Aus diesem Grunde wird sehr häufig ein Datenbus für verschiedene Aufgaben verwendet. Mit zusätzlichen Steuersignalen wird angezeigt, welche Daten gerade über den Bus übertragen werden. Bei diesem System spricht man von **Shaved Busses.**
Die Breite eines Busses, also die Anzahl der Leitungen, wird bestimmt von der Wortlänge des zu übertragenden Signals. Hat z. B. der Programmzähler 16 bit, d. h., der Mikrorechner kann

insgesamt 2^{16} = 64 k Wörter im Programmspeicher adressieren, so muß der Bus zur Ausgabe der Adresse eines Befehles 16 bit breit sein.
Um die Anschlüsse an den integrierten Schaltungen zu reduzieren, wird manchmal ein solches Datenwort in mehreren Schritten über einen schmaleren Bus übertragen, z. B. in 2 Hälften zu je 8 bit. Dieser Multiplexbetrieb verlangsamt natürlich die Übertragung. Zusätzlich sind Register erforderlich, um die zuerst übertragenen Daten zwischenzuspeichern, bis das vollständige Wort zur Verfügung steht.
Sehr häufig wird derselbe Bus zur Übertragung von Daten in 2 Richtungen benutzt, z. B. derselbe Bus zur Übertragung von Daten vom Speicher zur CPU und von der CPU zum Speicher. Das bedeutet, daß der Bus von verschiedenen Stellen aus angesteuert werden muß. Bedingung ist hierfür, daß TM-State-Logikschaltungen verwendet werden. Das sind Schaltungen, deren Ausgänge außer dem H- und L-Pegel noch einen weiteren Zustand mit hoher Impedanz haben. In diesem Zustand sind die Ausgangstreiber ausgeschaltet und die Ausgänge folgen dem Pegel des Busses, ohne diesen nennenswert zu belasten. Bei diesem Verfahren muß durch eine entsprechende Steuerung unbedingt sichergestellt werden, daß zu einem bestimmten Zeitpunkt nur **ein** Baustein den Bus ansteuert. Alle anderen Bausteine müssen sich im Hochimpedanzzustand befinden, da sich andernfalls Kurzschlüsse ergeben können.
Bei Mikroprozessoren mit 4 und 8 bit Wortlänge wird häufig folgendes Bussystem verwendet:
– Ein Adreßbus mit 12 bis 16 bit Breite zur Adressierung von Programmspeicher, Datenspeicher und zur Selektion von Peripheriegeräten
– Ein Datenbus, der in beiden Richtungen benutzt wird, zur Übertragung der Instruktionen vom Programmspeicher, der Daten vom und zum Datenspeicher sowie der Daten von und zu den Interface-Schaltungen der Peripheriegeräte. Die Breite dieses Datenbusses entspricht der Wortlänge des Rechners
– Ein Steuerbus zur Übertragung der erforderlichen Steuersignale. Gelegentlich wird auch der Datenbus mitbenutzt, um zusätzliche Steuersignale und Informationen über den Rechnerstatus im Zeitmultiplex mit den Daten zu übertragen. In diesem Fall werden dann zusätzliche Register benötigt um diese Informationen zwischenzuspeichern

Ein typischer Vertreter dieser Gruppe von Mikroprozessoren ist der MP 8080. Sein Bussystem zeigt Bild 8.1.1.

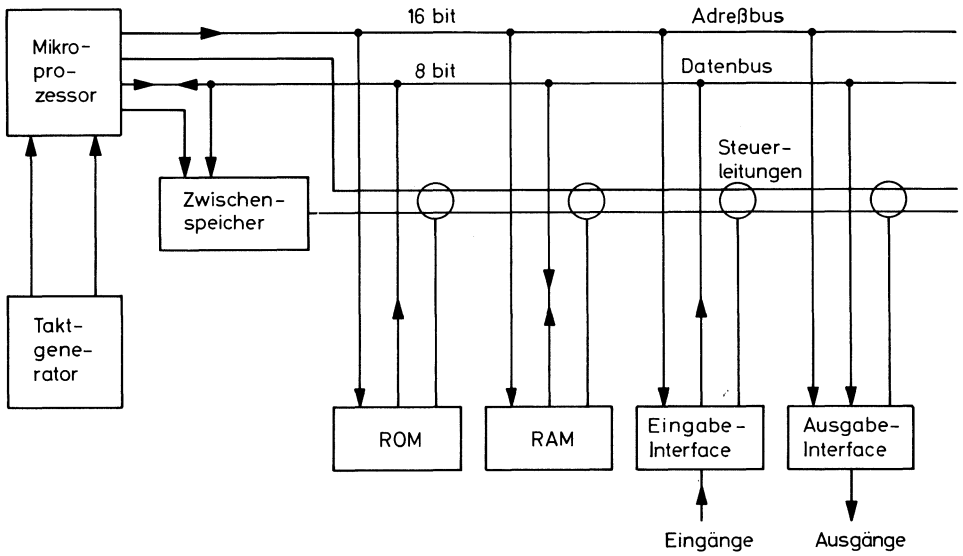


Bild 8.1.1.
Vereinfachtes 8080-System

Bei diesem System wird die Möglichkeit benutzt, zusätzliche Statusinformationen über den Datenbus zu übertragen. Ein Steuersignal (SYNC) zeigt an, wenn diese Statusinformation übertragen wird. Dieses Bussystem und die Funktion der Steuersignale wird im nächsten Abschnitt eingehend behandelt.
Bei Mikroprozessoren mit Wortlängen von 12 bis 16 bit wird häufig ein gemeinsamer Daten/Adreßbus benutzt. Über diesen Bus werden im Zeitmultiplex die Adressen für Programm- und Datenspeicher sowie die Instruktionen und Daten übertragen. Zusätzlich ist wieder ein

Steuerbus vorhanden. Bei diesen Mikroprozessoren werden häufig Speicherelemente und Interface-Schaltkreise mit integrierten Adreßregistern verwendet. Diese Register werden mit einem zusätzlichen Steuersignal getaktet, wenn auf dem Bus die Adressen übertragen werden. In Bild 8.2.1 sind 2 typische Konfigurationen dieser Gruppe dargestellt.

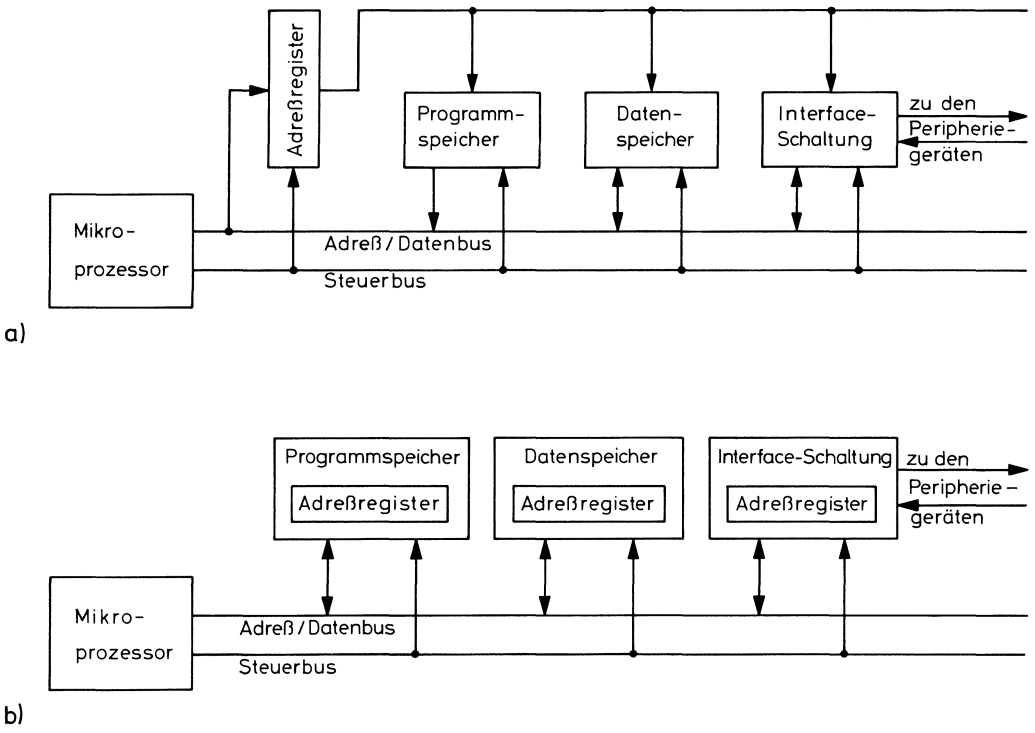


Bild 8.1.2
Typische Buskonfiguration bei 12- bis 16-bit-Mikrorechnern
a) Ausführung mit separatem Adreßregister
b) Ausführung mit integrierten Adreßregistern

Nachfolgend soll der Datentransfer zwischen Mikroprozessor und Speicher sowie zwischen Mikroprozessor und Peripheriegeräten genauer behandelt werden.

8.2 Datentransfer zwischen Mikroprozessor und Speicher

Programm und Datenspeicher eines Mikrorechners werden je nach Größe der Speicher aus einer Anzahl von einzelnen Speicherelementen aufgebaut. Die Adressierung einer einzelnen Speicherzelle erfolgt dann in 2 Stufen:
Der Mikroprozessor gibt auf dem Adreßbus die Adresse aus. Aus dieser Adresse muß zunächst abgeleitet werden, welcher Speicherblock, also welcher Baustein, adressiert wird. Hierzu wird aus der Adresse ein Chip-Enable-Signal abgeleitet, das dem entsprechenden Steuereingang als Speicherbaustein zugeführt wird. Dieses Signal schaltet den betreffenden Baustein ein oder aus. Nur wenn am Eingang der richtige Pegel anliegt, wird die Adresse decodiert, und es können dann Daten gelesen oder geschrieben werden. Danach wird über die Adreßleitungen am Speicherbaustein eine einzelne Zelle adressiert. Folgendes Beispiel soll dies verdeutlichen:
Bei einem Rechner mit 16 bit Adreßbusbreite wird ein 5-k-Programmspeicher mit 8 bit Wortlänge benötigt. Zur Verfügung stehen hierfür 5 ROMs mit 1 k x 8 bit. Zusätzlich wird ein 1-k-Schreib/Lesespeicher mit ebenfalls 8 bit Wortlänge benötigt. Zur Verfügung stehen 8 RAMs mit 1 k x 1 bit Kapazität. Alle Speicherbauelemente haben 10 Adreßeingänge (2^{10} = 1024 = 1 k) und einen Chip-Enable-Eingang. Für dieses Beispiel kann folgende Anordnung zur Adreßdecodierung verwendet werden: Die bit 0 bis 9 des Adreßbusses werden verbunden mit den Adreßeingängen der Speicherbauelemente. Die bit 10 bis 15 werden als Chip-Enable-Signale verwendet (Bild 8.2.1).

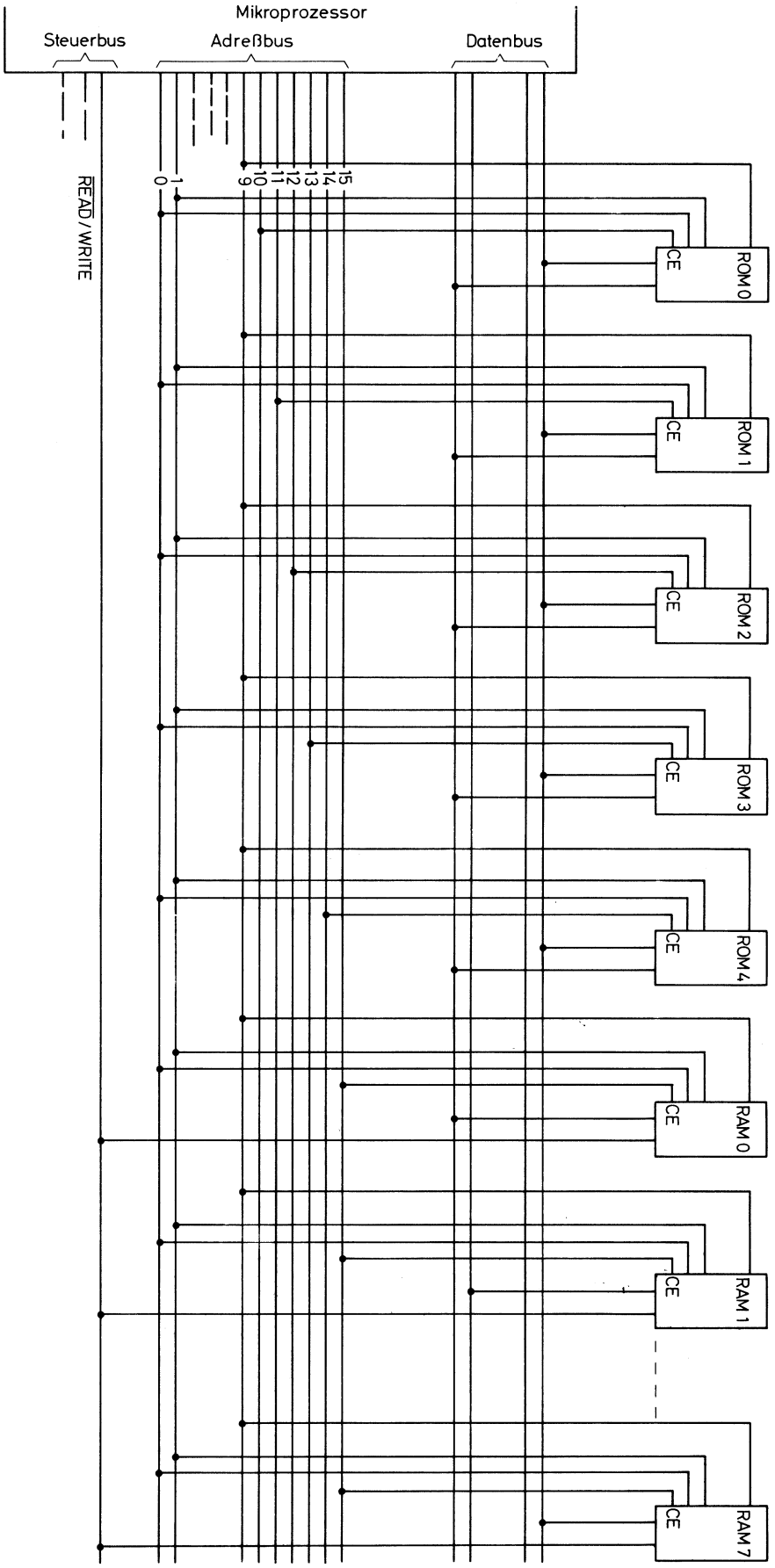


Bild 8.2.1
Einfache Adreßdecodierung für 1-k-Blöcke ROM und RAM (Chip-Enable-Signale)

Für diese Anordnung ergibt sich folgende Adressenzuordnung:

ROM 0:	1 0 0 0 0 0 0 0 0 0 0 0 –	1 1 1 1 1 1 1 1 1 1 1 1
ROM 1:	1 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 1 1 1 1 1 1 1 1 1 1
ROM 2:	1 0 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 0 1 1 1 1 1 1 1 1 1 1
ROM 3:	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 0 0 1 1 1 1 1 1 1 1 1 1
ROM 4:	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 0 0 0 1 1 1 1 1 1 1 1 1 1
RAM:	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1

Die 8 RAMs werden bei dieser Anordnung entsprechend Bild 8.2.2 zusammengeschaltet.

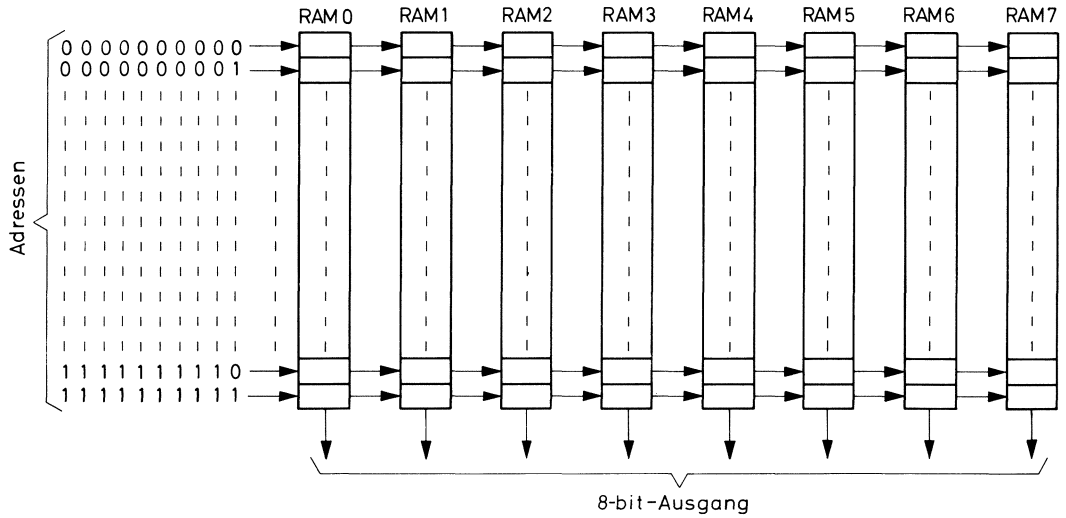


Bild 8.2.2
Zusammenschaltung der acht 1-k x 1-bit-RAMs

Bild 8.2.2 zeigt, daß die Adreßeingänge aller 8 RAMs parallelgeschaltet werden, so daß letztlich ein RAM mit 1 k x 8 bit entsteht.
Der Vorteil dieser Anordnung ist, daß keine zusätzlichen Bauelemente zur Adreßdecodierung benötigt werden. Der Nachteil ist, daß der Adreßbereich des Mikroprozessors nur zu einem kleinen Teil ausgenutzt werden kann. Alle Adressen, die in den bit 10 bis 15 mehr als eine 1 aufweisen, sind in dieser Anordnung nicht verwendbar, da sonst mehr als ein Baustein gleichzeitig selektiert würde. Aus diesem Grunde wird diese Art der Adreßdecodierung bei kleinen Systemen mit geringem Speicherbedarf eingesetzt und ist dort sehr wirtschaftlich.
Sollen mehr Speicher verwendet werden, müssen die Leitungen 10 bis 15 des Adreßbusses ganz oder teilweise decodiert werden. Eine solche Anordnung zeigt Bild 8.2.3.
Aus den Adreßbusleitungen 10 bis 12 werden die Chip-Enable-Signale abgeleitet, indem diese Adreßleitungen mit einem 8-aus-3-Decoder decodiert werden (Decoder 1). Die Ausgänge dieses Decoders führen direkt die Chip-Enable-Signale. In dem gewählten Beispiel werden von den 8 zur Verfügung stehenden Ausgangsleitungen nur 6 benötigt.
Die Adreßleitungen 13 bis 15 werden ebenfalls in einem 8-aus-3-Decoder decodiert (Decoder 2). Eines der Ausgangssignale von Decoder 2 wird verwendet, um Decoder 1 einzuschalten (Chip-Enable-Signal für Decoder 1). Die restlichen 7 Ausgangssignale sind noch frei für Erweiterungen. Insgesamt kann mit diesen 8 Ausgangsleitungen einer aus 8 Decodern selektiert werden. Jeder dieser 8 Decoder kann wieder einen aus 8 Speicherblöcken à 1 k selektieren, so daß insgesamt einer aus 64 Speicherblöcken à 1 k selektiert werden kann. Mit dieser Anordnung ist im voll ausgebauten Zustand also der volle Adreßbereich von 64 k, der durch die 16 Adreßleitungen gegeben ist, ausgeschöpft. Für das gewählte Beispiel ergibt sich folgende Adressenzuordnung:

ROM 0:	0 0 0 0 0 0 0 0 0 0 0 0 0 0 –	0 0 0 1 1 1 1 1 1 1 1 1 1 1
ROM 1:	0 0 1 0 0 0 0 0 0 0 0 0 0 0 –	0 0 1 1 1 1 1 1 1 1 1 1 1 1
ROM 2:	0 1 0 0 0 0 0 0 0 0 0 0 0 0 –	0 1 0 1 1 1 1 1 1 1 1 1 1 1
ROM 3:	0 1 1 0 0 0 0 0 0 0 0 0 0 0 –	0 1 1 1 1 1 1 1 1 1 1 1 1 1
ROM 4:	1 0 0 0 0 0 0 0 0 0 0 0 0 0 –	1 0 0 1 1 1 1 1 1 1 1 1 1 1
RAM:	1 0 1 0 0 0 0 0 0 0 0 0 0 0 –	1 0 1 1 1 1 1 1 1 1 1 1 1 1

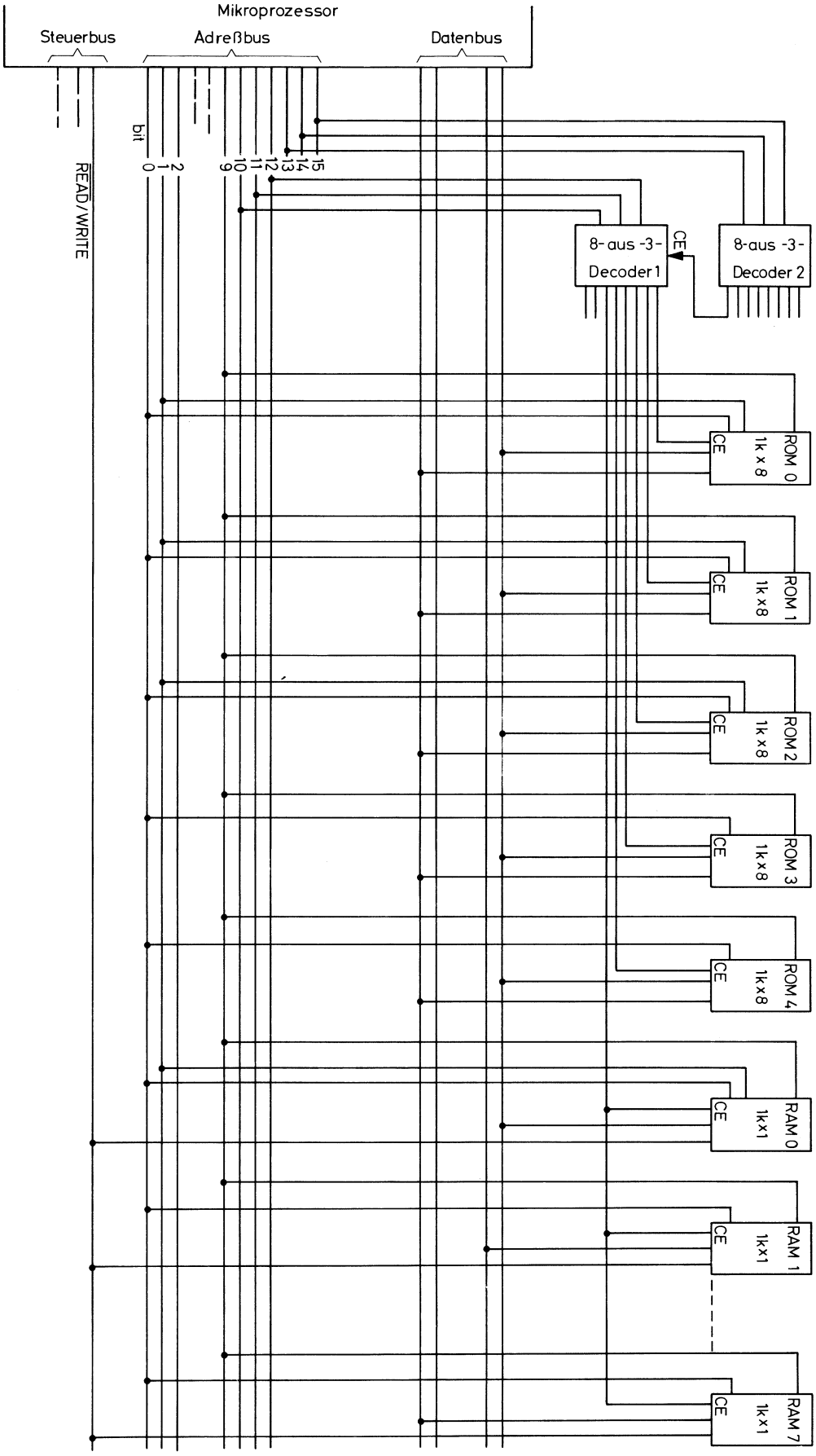


Bild 8.2.3
Vollständige Adreßdecodierung für 1-k-Blöcke ROM und RAM (Chip-Enable-Signale)

In der Praxis sind viele Zwischenlösungen zwischen der einfachen und der vollständigen Decodierung gebräuchlich, je nach Anforderungen des Systems. Häufig werden auch Speicherbausteine und Interface-Schaltungen mit mehr als einem Chip-Enable- oder Chip-Selekt-Eingang verwendet. Dies ermöglicht eine zusätzliche Adreßdecodierung ohne Verwendung weiterer Bausteine.
Der Datentransfer zwischen Mikroprozessor und Speicher kann synchron oder asynchron durchgeführt werden. Beim synchronen Transfer wird der Zeitablauf ausschließlich vom Taktgenerator des Systems bestimmt. Zu einem bestimmten Zeitpunkt innerhalb eines Befehlszyklus werden die Adressen über den Adreßbus ausgegeben. Dieser Zeitpunkt wird durch ein Steuersignal markiert. Leider gibt es für diese Steuersignale keine einheitlichen Namen. Ein Name, der teilweise verwendet wird und auch aussagekräftig ist, wäre Valid-Memory-Address (etwa: gültige Speicheradresse). Der Mikroprozessor erwartet dann, daß zu einem zweiten späteren Zeitpunkt im Befehlszyklus die Daten auf dem Datenbus bereitstehen bzw. vom Datenbus in den Speicher übernommen werden. Die Zeitverzögerung zwischen diesen beiden Zuständen, die die Zugriffszeit des Speichers berücksichtigt, ist also nur vom Systemtakt festgelegt. Man kann deshalb in solchen Systemen nur Speicherbauelemente verwenden, deren Zugriffszeit kompatibel mit dieser festgelegten Zeitverzögerung ist.
Beim asynchronen Datentransfer werden zwischen Mikroprozessor und Speicher Synchronisiersignale ausgetauscht. Der Datentransfer wird vom Mikroprozessor mit einem Anforderungsignal (Transfer-Request) eingeleitet. Der Speicher quittiert dieses Signal mit einem Anerkennungssignal (Ready-Signal). Dieses Signal deutet an, daß die Daten vom Speicher auf dem Datenbus bereitstehen bzw. daß der Speicher die Daten vom Datenbus übernommen hat. Der Mikroprozessor geht nach Ausgabe des Valid-Memory-Address-Signales in einen Wartezustand. In diesem Wartezustand prüft er wiederholt die Ready-Leitung. Diese Prüfung wird vom Systemtakt getriggert. Wenn nun das Ready-Signal anliegt, wird der Wartezustand beendet. Auf diese Weise können Speicher mit beliebiger Zugriffszeit mit einem Mikroprozessor synchronisiert werden.
Der asynchrone Datentransfer z. B. vom Speicher zum Mikroprozessor verläuft also in folgenden Schritten:
- Der MP gibt die Adresse auf den Bus aus
- Der MP gibt ein Transfer-Request-Signal aus und wartet auf das Ready-Signal vom Speicher
- Der Speicher gibt ein Ready-Signal, das anzeigt, daß die Daten auf dem Datenbus bereitstehen
- Der MP liest die Daten und setzt den Befehlszyklus fort

Die verschiedenen MP-Typen unterscheiden sich etwas in der genauen Bedeutung dieser Steuersignale und im genauen Zeitablauf. Es handelt sich dabei jedoch immer um Abwandlungen dieser grundsätzlichen Zusammenhänge.
Ein weiteres Steuersignal, das der Mikroprozessor ausgibt, legt fest, ob die Daten vom Speicher gelesen oder in den Speicher geschrieben werden sollen ($\overline{\text{Read}}$/Write-Signal). Dieses Signal steuert den Schreib/Leseeingang des RAMs, für ROMs hat es natürlich keine Bedeutung.

8.3 Datentransfer zwischen Mikroprozessor und Peripheriegeräten

Das Bussystem des Mikroprozessors wird über spezielle Anpaßschaltungen (Interface-Schaltungen, I/O-Ports) mit den Peripheriegeräten verbunden. Es werden verschiedene Datentransferarten verwendet, für die zum Teil unterschiedliche Anpaßschaltungen erforderlich sind. Nachfolgend wird auf diese Punkte näher eingegangen.

8.3.1 Anpaßschaltungen für Datentransfer unter Programmkontrolle

Bei dieser Art des Datentransfers steuert das Programm des Rechners den Datentransfer. Normalerweise wird der Inhalt eines Registers an das Peripheriegerät ausgegeben, bzw. es werden die Daten von einem Peripheriegerät in ein Register übernommen. Zunächst muß also der Rechner ein bestimmtes Peripheriegerät ansprechen oder adressieren. Diese Aufgabe entspricht der Adressierung einer bestimmten Speicherzelle. Dazu gibt der Rechner über

einen Bus (z.B. einen Perpheral-Adreßbus) eine Adresse aus. Jeder Interface-Schaltung, also jedem Peripheriegerät im System, ist eine bestimmte Adresse zugeordnet. Die Interface-Schaltung decodiert die Adresse und spricht nur dann an, wenn die eigene Adresse auf dem Bus erscheint. Der Zeitpunkt, zu dem die Adresse auf dem Adreßbus ansteht, wird durch ein Steuersignal (Valid-Peripheral-Address) markiert. Gleichzeitig stehen bei einem System mit getrenntem Datenbus die Daten bereit. Diese Daten werden in ein Register übernommen. Der Takt für das Register wird vom Valid-Peripheral-Address-Signal, dem Datenrichtungssignal (Input/$\overline{\text{Output}}$) und der decodierten Adresse abgeleitet (Bild 8.3.1.1).

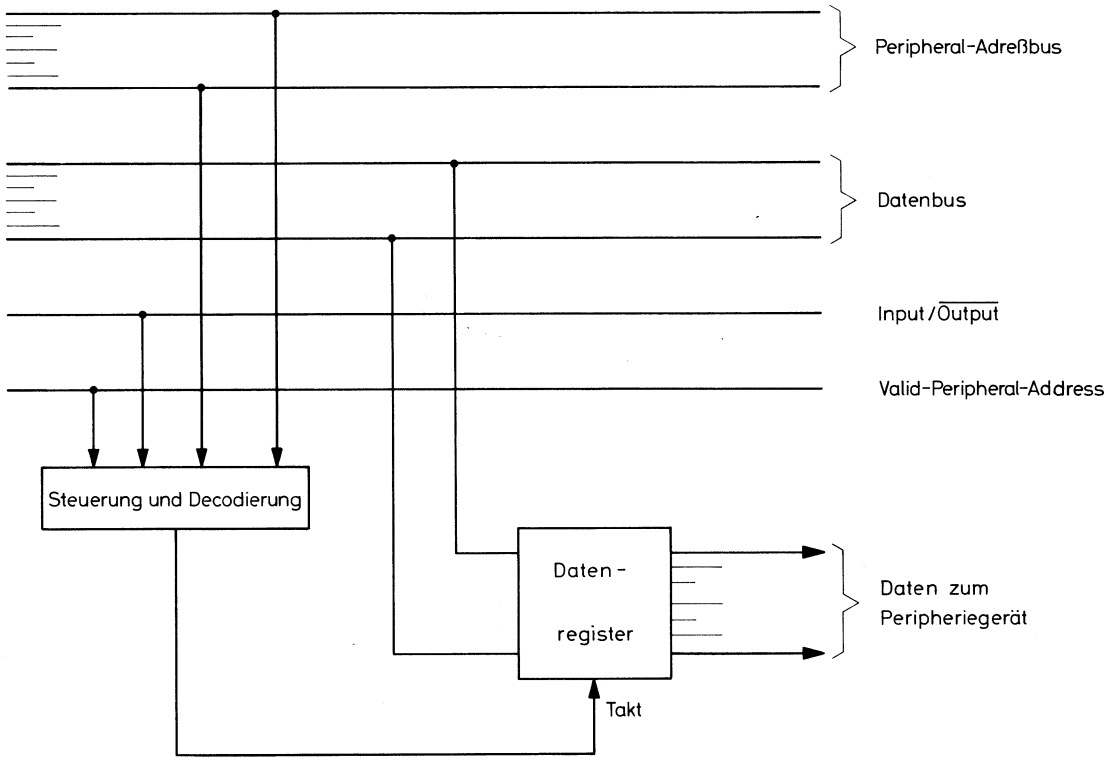


Bild 8.3.1.1
Blockschaltbild einer Ausgangsanpaßschaltung

Das Eingabe-Interface hat die gleiche Struktur wie das Ausgabe-Interface. Die Daten werden dabei aber über Bustreiber auf den Datenbus übertragen. Das Datenrichtungssignal Input/$\overline{\text{Output}}$ markiert den Zeitpunkt, zu dem der bidirektionale Datenbus frei ist, so daß die Bustreiber im Interface den Bus ansteuern können (Bild 8.3.1.2).
Bei den verschiedenen Mikroprozessoren werden unterschiedliche Konzepte verwendet, um Peripherieadressen, Daten und Steuersignale über die zur Verfügung stehenden Busse zu übertragen. Manche Rechner haben einen getrennten Bus für die Peripherieadressen, die Daten werden hierbei z. B. über den Systemdatenbus übertragen. Andere Mikrorechner verwenden den Adreßbus für Speicher und Peripherieadressen. Sie stellen aber dann eine besondere Steuerleitung zur Verfügung, die anzeigt, ob auf dem Adreßbus eine Speicheradresse oder eine Peripherieadresse ausgegeben wird. Die zur Verfügung stehenden Adressen werden also in diesem Falle sowohl für die Speicheradressierung als auch für die Adressierung der Peripheriegeräte verwendet. Es gibt Befehle, die Datentransfer zum Speicher ausführen und zusätzliche Ein/Ausgabebefehle.
Eine dritte Gruppe von Mikrorechnern behandelt die Peripheriegeräte wie Speicher. In diesem Falle werden zur Steuerung der Peripheriegeräte dieselben Busse und Steuersignale verwendet wie für den Speicher. Ein Teil des zur Verfügung stehenden Adreßraumes ist für Speicher reserviert, ein anderer Teil für Peripheriegeräte. Es gibt bei diesen Rechnern keine getrennten Speicher- und Ein/Ausgabebefehle; ob sich ein Befehl auf eine Speicherzelle oder ein Peripheriegerät bezieht, wird durch die Adresse festgelegt.
Ein Beispiel für diese Art der Adressierung von Peripheriegeräten ist im hypothetischen Mikrorechner für die Dateneingabe verwirklicht. In diesem Rechner ist die Adresse F F einem Peripheriegerät, nämlich dem *A*-Schalter-Register zugeordnet. Alle Befehle, die sich auf den

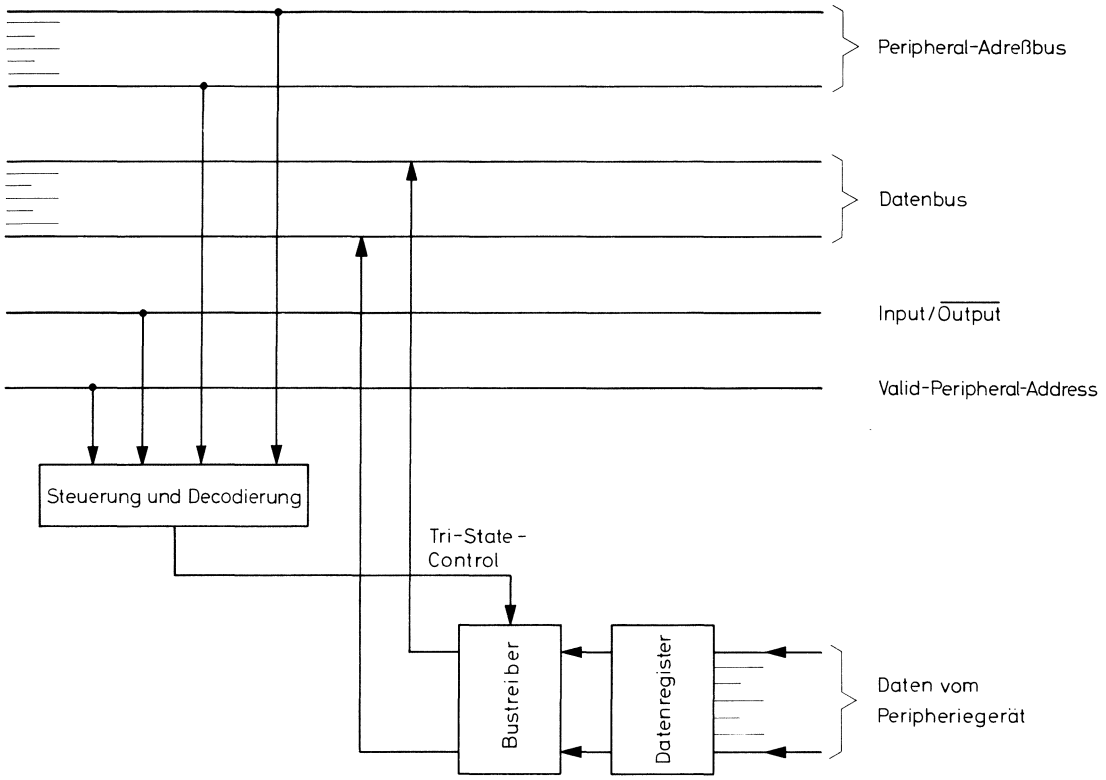


Bild 8.3.1.2
Blockschaltbild einer Eingangsanpaßschaltung

Speicher beziehen, können auch mit dem Peripheriegerät verwendet werden, wenn im Adreßteil des Befehles die Adresse F F angegeben wird. Besondere Eingabebefehle sind nicht vorhanden.

8.3.2 Einleitung und Durchführung des Datentransfers zwischen Mikroprozessor und I/O-Port

Der Datentransfer zwischen Mikroprozessor und Peripheriegerät über I/O-Ports kann durch Rechnerabfrage (Polling) oder durch Programmunterbrechung (Interrupts) eingeleitet werden.

In einem Polling-System werden zusätzlich zu den Daten noch Statusinformationen zwischen Peripheriegerät und dem Rechner ausgetauscht. Über ein besonderes Steuersignal teilt jedes Eingabegerät dem Rechner mit, ob Daten zur Verfügung stehen oder nicht. Jedes Ausgabegerät teilt dem Rechner mit, ob es frei ist oder ob es gerade andere Daten ausgibt. Diese Statusinformation wird genauso zum Rechner übertragen wie die Daten selbst. Dem Peripheriegerät ist ein Statusregister mit einer eigenen Adresse zugeordnet. Dieses zeigt den Zustand des Datenregisters (leer oder voll) an.
Der Rechner adressiert dieses Register und liest die Information über den Zustand des Peripheriegerätes. Ein Datentransfer verläuft also in mehreren Schritten. Bei einer Dateneingabe fragt der Mikroprozessor das Peripheriegerät, also das Statusregister, ob dort Daten zur Verfügung stehen. Wenn ja, so liest er diese ein, wenn nicht, so wartet er auf die Daten in einer Programmschleife, bzw. er setzt sein Programm fort und fragt zu einem späteren Zeitpunkt erneut ab. In Bild 8.3.2.1 ist ein einfacher Programmablaufplan für eine programmierte Dateneingabe mit Warteschleife dargestellt.
Bei der Datenausgabe muß der Rechner prüfen, ob das Peripheriegerät frei ist. Ist es frei, können die Daten ausgegeben werden, ist es belegt, z. B. noch nicht fertig mit der Ausgabe des vorhandenen Datenwortes, muß der Rechner entweder in einer Programmschleife warten (Bild 8.3.2.2) oder sein Programm fortsetzen und später noch einmal abfragen.

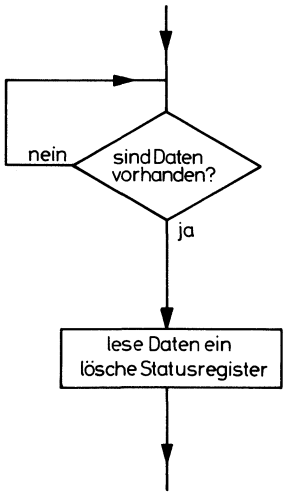


Bild 8.3.2.1
Programmierte Dateneingabe

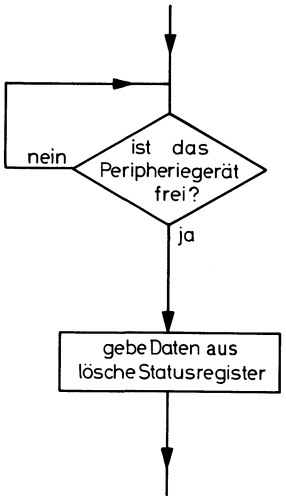


Bild 8.3.2.2
Programmierte Datenausgabe

Bei dieser Art der Datenein/ausgabe hat also der Programmierer die vollständige Kontrolle über den Transfer. Der Datentransfer findet an den dafür vorgesehenen Stellen im Programm statt.
Beim Interruptbetrieb geht die Initiative zu einem Datentransfer vom Peripheriegerät bzw. dem I/O-Port aus. Das Peripheriegerät erzeugt ein Signal, das sog. Unterbrechungsanforderungssignal (Interrupt-Request-Signal), das dem Rechner mitteilt, daß bei einem Dateneingabegerät Daten zur Verfügung stehen bzw. daß ein Ausgabegerät neue Daten übernehmen kann. Auf dieses Interrupt-Request-Signal hin unterbricht der Rechner die Ausführung des gerade laufenden Programms und springt zu einem Programm, das das Peripheriegerät bedient. Ist dieses Programm beendet, so wird das unterbrochene Programm fortgesetzt. Programmtechnisch gesehen entspricht dies einem Sprung zu einem Unterprogramm, der nicht durch einen CALL-Befehl verursacht wird, sondern der hardware-mäßig durch das Interrupt-Request-Signal eingeleitet wird. Die Vorgänge beim Sprung zur Interrupt-Routine, die das Peripheriegerät bedient, sowie der Rücksprung von dieser Routine gleichen somit weitgehend den Vorgängen beim Anruf von Unterprogrammen. Trifft beim Rechner ein Interrupt-Request-Signal ein, so wird zunächst der gerade laufende Rechnerzyklus bzw. der gerade laufende Befehl normal abgearbeitet. Dann wird der Inhalt des Programmzählers auf dem Stack abgespeichert. Nun wird der Programmzähler mit der Adresse der Interrupt-Routine geladen, d. h., der Sprung wird ausgeführt. Diesen Zustand teilt der Rechner dem Peripheriegerät durch ein Unterbrechungsanerkennungssignal (Interrupt-Acknowledge-Signal) mit. Damit weiß das Peripheriegerät, daß seine Unterbrechungsanforderung vom Rechner akzeptiert wurde. In der nun folgenden Interrupt-Routine werden die Daten ein- bzw. ausgelesen. Der entsprechende I/O-Port wird adressiert und die Daten werden wie beschrieben übertragen. Am Schluß der Interrupt-Routine erfolgt der Rücksprung zum Hauptprogramm mit dem normalen RETURN-

Befehl. Die Verwendung des Stacks zur Abspeicherung der Rücksprungadressen hat zur Folge, daß ein Interrupt an einer beliebigen Stelle im Programm erfolgen darf, auch mitten in einer Serie verschachtelter Unterprogramme. Es wird so auch möglich, Interrupts zu verschachteln, d. h. eine gerade laufende Interrupt-Routine durch eine weitere Interruptanforderung zu unterbrechen. In diesem Falle, d.h., wenn mehrere Peripheriegeräte Interruptbetrieb durchführen wollen, stellt sich natürlich die Frage nach der Priorität: Wer darf wen unterbrechen, und wer wird zuerst bedient, wenn 2 Anforderungen gleichzeitig eintreffen? Diese Priorität muß im System festgelegt sein. Normalerweise bekommt das Gerät mit höherer Geschwindigkeit auch die höhere Priorität zugewiesen. So wird z. B. ein Bandgerät normalerweise eine höhere Priorität haben als die Eingabetastatur.
In der Art der Festlegung und in der Zahl dieser Prioritätsebenen unterscheiden sich die verschiedenen auf dem Markt befindlichen Mikrorechner. Prinzipiell sind 2 unterschiedliche Methoden möglich:
- Eine software-mäßige Festlegung und
- Eine hardware-mäßige Festlegung

Bei der Software-Lösung, dem sog. Interrupt-Polling, sind alle Peripheriegeräte an eine gemeinsame Interrupt-Request-Leitung angeschlossen. Bei einem Interrupt-Request fragt nun der Rechner über die Statusregister alle Peripheriegeräte ab, welches den Interrupt-Request erzeugt hat.
Nun kann der Rechner per Programm entscheiden, ob der Interrupt zugelassen wird oder nicht. Er kann so gegebenenfalls den Sprung zur Interrupt-Routine ausführen, die dem unterbrechenden Peripheriegerät zugeordnet ist. Diese Methode ist wegen der vielen Abfragen recht langsam, d.h., es vergeht eine recht große Zeit zwischen dem Interrupt-Request und der Bedienung des Gerätes.
Dieser Nachteil wird bei der hardware-mäßigen Lösung vermieden. Hier wird durch eine interne oder externe Schaltung die Priorität der einzelnen Geräte festgelegt. Außerdem werden die Startadressen der verschiedenen Interruptprogramme definiert, so daß bei einem Interrupt-Request ohne programmierte Abfrage das richtige Unterprogramm ausgeführt wird.

2 programmtechnische Punkte sollen noch erwähnt werden. In vielen Fällen wird man in der Interrupt-Routine die Register des Rechners zum Arbeiten benötigen und auch die Flags verändern. Aus diesem Grunde muß generell am Anfang der Interrupt-Routine der Rechnerstatus abgespeichert werden, wenn er in der Routine verändert wird. Die bequemste Art dieser Abspeicherung erfolgt auch hier wieder mit Hilfe des Stacks, vorausgesetzt, daß der Stack-Bereich groß genug ist. Am Ende der Interrupt-Routine, also unmittelbar vor dem Rücksprung, werden dann die Register und Flags wieder in ihre Ausgangszustände zurückgesetzt. Für das Hauptprogramm ist somit keine Veränderung des Prozessorstatus sichtbar.
Manchmal ist es erforderlich, daß an bestimmten Stellen im Hauptprogramm kein Interrupt erfolgen darf (z. B. bei der Durchführung von zeitkritischen Operationen). In einem solchen Fall kann man durch Abschalten der Interrupt-Request-Leitung den Interruptbetrieb verhindern. Dazu sind sog. Interruptmasken vorhanden. Das sind Flipflops bzw. Register, die anzeigen, ob ein Interrupt im Augenblick zulässig ist, bzw. bei einem Prioritätsschema, ab welcher Prioritätsebene ein Interrupt zugelassen wird. Diese Flipflops bzw. Register können per Programm geladen oder gelöscht werden, d.h., Interrupts können per Programm erlaubt oder verboten werden.
Wie schon erwähnt, unterscheiden sich die Mikroprozessoren in ihren Interruptsystemen beträchtlich, so daß hier nur die allgemeinen Prinzipien aufgezeigt werden konnten.
Wenn wir Datentransfer im Abfragebetrieb (Polling) und im Unterbrechungsbetrieb (Interrupt) vergleichen, so ergibt sich für das Abfrageverfahren folgender Vorteil: Der Datentransfer kann nur an genau spezifizierten Programmstellen erfolgen, so daß der Programmierer die vollständige Kontrolle über den Programmablauf hat. Beim Interruptbetrieb ist die Kontrolle über den Programmablauf nicht so weitgehend, d.h., die Programmstruktur wird möglicherweise komplizierter und muß sorgfältiger geplant werden. Demgegenüber hat der Interruptbetrieb den Vorteil, daß die Peripheriegeräte schneller bedient werden und daß keine Rechenzeit in Warteschleifen verschwendet wird.
Je nach Interruptsystem des Rechners kann ein Interruptprioritätsschema mit mehreren Ebenen einen beträchtlichen Logikaufwand erfordern. Es wurden deshalb dafür hochintegrierte Bausteine entwickelt, die diese Aufgabe erheblich erleichtern. In Abschnitt 9. werden wir auf nähere Einzelheiten für das 8080-System noch eingehen.

8.3.3 Direct-Memory-Access (DMA)

Mit den vorher beschriebenen Methoden werden durch die Ein/Ausgabebefehle die Daten normalerweise zwischen einem Register des Rechners und dem Peripheriegerät transferiert. Bei verschiedenen Peripheriegeräten wie z. B. Plattenspeichern oder Bandgeräten fallen die Daten blockweise und mit hoher Geschwindigkeit an. Das führt dazu, daß der Rechner zwischen den einzelnen Datenwörtern keine Zeit hat, die Daten zu verarbeiten. Er muß vielmehr ein Datenwort nach dem Einlesen zwischenspeichern und dann sofort das nächste Wort einlesen. Dafür sind bei dem Datentransfer mit Programmsteuerung mindestens 2 Befehle erforderlich, der Einlesebefehl und der Befehl, der die Daten abspeichert und dabei automatisch die Speicheradresse modifiziert (z. B. Auto-Increment-Mode). Wenn das Peripheriegerät die Daten direkt in den Speicher übertragen würde, ohne dabei den Rechner zu benutzen, so wäre es möglich, mit wesentlich höherer Datenrate zu übertragen. Die Zugriffszeit des Speichers betrüge maximal etwa die Hälfte eines Befehles. Dies bedeutet, in der Zeit, die der Rechner zum Einlesen und Abspeichern eines Datenwortes benötigt, könnten unter diesen Bedingungen 4 Datenwörter direkt in den Speicher übertragen werden.

Man nennt diese Methode des direkten Datentransfers zwischen einem Peripheriegerät und dem Speicher **Direct-Memory-Access (DMA).** Der Rechner ist bei dieser Transferart nicht beteiligt, er wird für die Dauer des Transfers in einen Wartezustand geschaltet. Dabei müssen die Busleitungen frei sein, d.h., DMA-Betrieb ist nur möglich bei einem Prozessor mit Tri-State-Bustreibern, die während dieser Zeit in Hochimpedanzzustand geschaltet sein müssen. Das DMA-Interface muß also den gesamten Datentransfer selbst steuern. d.h. es muß die Adresse und die Daten über die entsprechenden Busse zum Speicher übertragen, und es muß den Zeitablauf des Vorganges steuern. Das DMA-Interface muß außerdem selbsttätig entscheiden, wann alle Daten übertragen sind. Es muß dann die Busse wieder freigeben und diesen Zustand dem Rechner mitteilen, so daß dieser sein Programm fortsetzen kann.

Im DMA-Betrieb hat das Interface wesentlich mehr Aufgaben zu erfüllen und ist dementsprechend komplizierter als ein Interface für Datentransfer unter Programmkontrolle. Der Vorteil dieses Verfahrens ist jedoch die Übertragungsrate beim Transfer von Datenblöcken. Diese Übertragungsrate ist nur begrenzt durch die Zugriffszeit des Speichers und die Auslegung der Busse.

Aber auch wenn keine so hohe Übertragungsrate beim Blocktransfer erforderlich ist, kann die DMA-Methode von Vorteil sein. Es ist dann möglich, den Rechner jeweils nur für die Dauer der Übertragung eines Datenwortes anzuhalten und ihn in der Zeit bis zum Eintreffen des nächsten Wortes wieder freizugeben. Dabei wird die Ausführung des Rechnerprogramms durch den Transfer nur etwas verlangsamt, da der Rechner während der Ausführung seines Programms nur immer wieder für einen Zyklus angehalten wird. Man nennt diese Methode deshalb Cycle-Stealing.

In Bild 8.3.3.1 ist das Blockschaltbild eines DMA-Interfaces gezeigt.

Die wichtigsten Register in diesem Interface sind der Adreßzähler und der Wortzähler. Der Adreßzähler enthält jeweils die Adresse der Speicherzelle, auf die zugegriffen werden soll. Der Inhalt des Adreßzählers wird über den Adreßbus zum Speicher übertragen. Nachdem ein Wort transferiert ist, wird der Adreßzähler von der Interface-Steuerung um 1 erhöht, so daß das nächste Datenwort in die nächste Speicherzelle geschrieben bzw. aus dieser ausgelesen wird. Vor Beginn des Datentransfers wird die Anfangsadresse des Speicherbereiches, der an dem DMA-Transfer beteiligt sein soll, in den Adreßzähler geladen. Dieses Laden erfolgt unter Programmkontrolle, die Schaltungsanordnung entspricht der eines I/O-Ports. Der Adreßzähler hat eine Peripherieadresse und wird mit einem Ausgabebefehl geladen.

Der Wortzähler zählt die übertragenen Wörter. Bei Erreichen der gewünschten Blocklänge, d.h. nach der gewünschten Anzahl von Datenwörtern, erzeugt er ein Überlaufsignal und leitet so das Transferende ein. Das Interface gibt die Busse frei, löscht das DMA-Request-Signal und zeigt so dem Rechner an, daß er sein Programm fortsetzen kann. Zusätzlich wird in vielen Fällen ein Interrupt-Request-Signal erzeugt. In der Interrupt-Routine kann der Rechner z.B. Statusinformationen vom Peripheriegerät abfragen, wie z.B. Informationen über Lesefehler bei einem Platten- oder Bandspeicher usw., oder er kann einen neuen Transfer einleiten.

Der Datentransfer geht normalerweise über ein Datenregister vom Datenbus zum Peripheriegerät. Die Ablaufsteuerung im DMA-Interface muß das erforderliche Taktsignal erzeugen. Dieses Taktsignal muß beim Lesen die Zugriffszeit des Speichers berücksichtigen. Beim synchronen Transfer wird dieses Taktsignal vom Systemtakt, beim asynchronen Transfer vom Ready-Signal abgeleitet.

Nachfolgend noch einmal die Schritte eines DMA-Transfers:

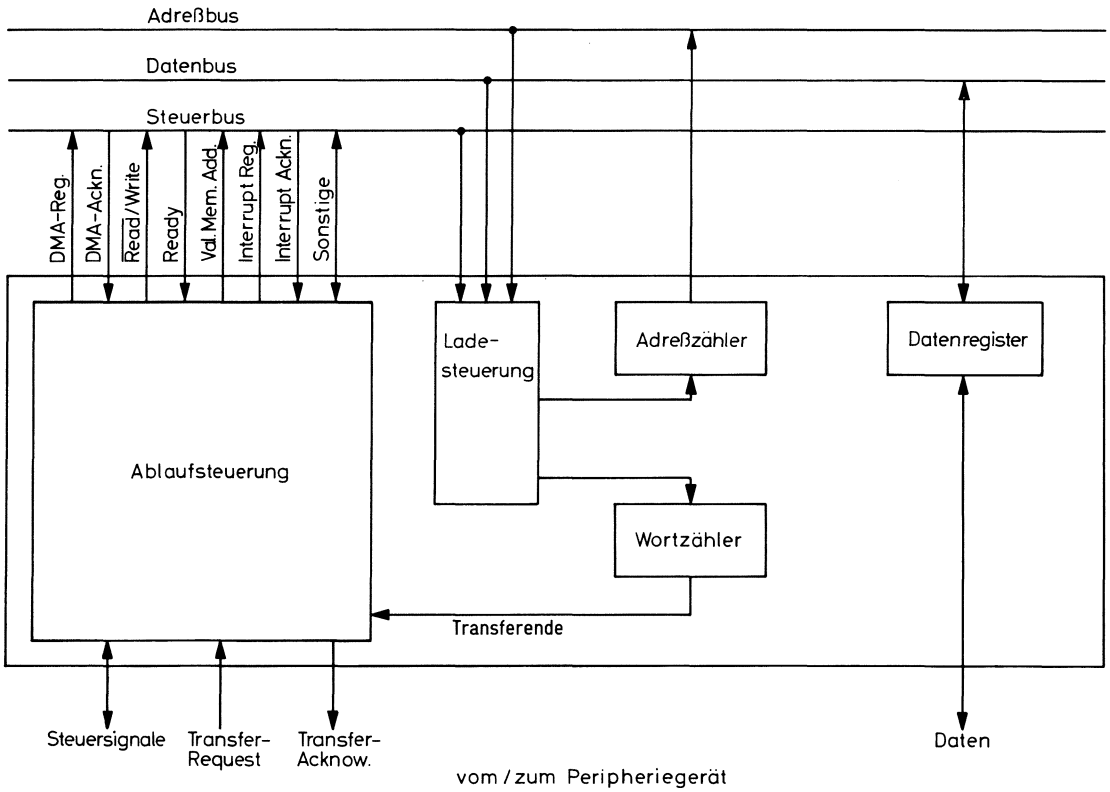


Bild 8.3.3.1
Blockschaltbild eines DMA-Interfaces

1. Laden des Adreßzählers mit der Anfangsadresse des Speicherbereiches

2. Laden des Wortzählers mit der Blocklänge

3. Transferanforderung durch das Peripheriegerät (Transfer-Request-Signal)

4. Das Interface gibt diese Anforderung an den Rechner weiter (DMA-Request). Wenn mehrere DMA-Kanäle vorhanden sind, muß durch eine Prioritätslogik entschieden werden, ob dieses Peripheriegerät im Moment zugelassen wird

5. Der Rechner beendet den laufenden Rechnerzyklus bzw. den laufenden Befehl normal und geht dann in den Wartezustand. Dabei werden die Bustreiber in den Hochimpedanzzustand geschaltet. Das Erreichen dieses Zustandes signalisiert er dem Peripheriegerät über die DMA-Anerkennungsleitung (DMA-Acknowledge)

6. Das Interface gibt diese Bestätigung an das entsprechende Peripheriegerät weiter, worauf der eigentliche Transfer beginnt

Dateneingabe	**Datenausgabe**
7. Das Peripheriegerät lädt das Datenwort ins Datenregister	7. Der Inhalt des Adreßzählers wird über den Adreßbus zum Speicher übertragen
8. Der Inhalt des Adreßzählers wird über den Adreßbus, der Inhalt des Datenregisters wird über den Datenbus zum Speicher übertragen	8. Die Daten vom Speicher werden über den Datenbus in das Datenregister transferiert
9. Das Interface erzeugt das Schreibsignal. Damit ist ein Wort geschrieben	9. Das Peripheriegerät übernimmt den Inhalt des Datenregisters. Damit ist ein Datenwort gelesen

10. Der Adreßzähler und der Wortzähler werden erhöht. Wenn der Wortzähler nicht „überläuft", wird das nächste Datenwort übertragen. Bei hoher Übertragungsrate (Burst Mode) wird dabei der Rechner nicht freigegeben, d.h., der DMA-Request bleibt bestehen. Die Schritte 7 bis 10 werden wiederholt.
Bei niedriger Übertragungsrate (Cycle-Stealing) werden nun der DMA-Request gelöscht und die Busse freigegeben. Der Rechner setzt sein Programm fort. Zum Transfer des nächsten Datenwortes werden die Schritte 3 bis 10 wiederholt.
Wenn der Wortzähler überläuft, sind alle Daten übertragen. Der DMA-Request wird gelöscht, die Busse werden freigegeben, d.h., die Tri-State-Ausgänge des Interfaces werden in den Hochimpedanzzustand geschaltet. Ein Interrupt-Request-Signal wird erzeugt

11. Der Rechner setzt sein Programm fort und akzeptiert je nach Priorität die Interruptanforderung des DMA-Interfaces

Zur Steuerung dieses recht komplexen Ablaufes ist ein beträchtlicher Logikaufwand erforderlich. Die für diese Aufgabe speziell entwickelten hochintegrierten Bausteine vereinfachen diese Aufgabe beträchtlich.

Fragen zu Abschnitt 8.

1. Was verstehen Sie unter einem Bussystem bei einem Mikrorechner?

2. Welche Voraussetzung muß erfüllt sein, damit auf einem Bus Daten in beide Richtungen übertragen werden können?

3. Welche Aufgabe hat der Chip-Enable-Eingang eines Speicherbausteines?

4. Auf was ist hinsichtlich der zu verwendenden Speicherbauelemente besonders zu achten, wenn ein synchroner Datentransfer zwischen Mikroprozessor und Speicher durchgeführt werden soll?

5. Was bedeutet „Datentransfer unter Programmkontrolle" hinsichtlich des Datentransfers zwischen Mikroprozessor und Peripheriegerät?

6. Was bedeutet der Begriff Polling?

7. Welche Aufgabe hat das Interrupt-Request-Signal?

8. Welchen Vorteil hat Datentransfer im Interruptbetrieb gegenüber Datentransfer im Polling-Betrieb?

9. Was heißt DMA-Betrieb?

9. Mikrorechnersysteme

Wie bereits erwähnt, besteht ein vollständiges Mikrorechnersystem aus der Zentraleinheit (CPU), dem Taktgenerator, Festwertspeicher, Schreib/Lesespeicher und den verschiedenen Ein/Ausgabe-Interfaces für die Peripheriegeräte. Solche Systeme werden in vielen unterschiedlichen Versionen auf dem Markt angeboten. Man kann dieses große Angebot nach verschiedenen Kriterien klassifizieren, um sich so den Überblick zu erleichtern. Besonders nützlich für den Anwender wäre eine Klassifizierung der Mikrorechner nach der Leistungsfähigkeit bzw. dem Preis/Leistungsverhältnis. Dazu müßte man jedem Mikrorechner eine allgemein gültige Leistungskennziffer zuordnen. Es ist aber praktisch unmöglich, eine solche Leistungskennziffer zu finden. Man kann zwar sehr leicht Parameter angeben, die in eine solche Leistungskennziffer eingehen, wie z.B. die Wortlänge, die Größe des adressierbaren Speicherbereiches, die Arbeits-

geschwindigkeit usw. Man kann jedoch kaum die Leistungsfähigkeit eines Instruktionssatzes (Befehlssatzes) allgemein gültig erfassen. Die häufig angegebene Zahl der Instruktionen ist als Kriterium völlig ungeeignet, da praktisch nie angegeben ist, wie diese Zahl ermittelt wurde, was als eine Instruktion und was als verschiedene Instruktionen gezählt wurde. Außerdem kommen die Struktur und die Leistungsfähigkeit des Instruktionssatzes in dieser Zahl überhaupt nicht zum Ausdruck. Im Abschnitt über Systemplanung werden wir auf diesen Punkt noch zurückkommen.

Häufig werden deshalb die Mikrorechner nach ihrer Wortlänge eingeteilt. Hiernach gibt es die Gruppen der 4-, 8-, 12- und 16-bit-Rechner sowie die sog. Bit-Slice-Mikroprozessoren, bei denen die Wortlänge beliebig vom Anwender festgelegt werden kann.

Diese Einteilung stellt in einer sehr groben Näherung auch eine Einteilung in verschiedene Leistungsgruppen dar. Wir wollen in diesem Abschnitt jedoch keine Marktübersicht angeben, da diese viel zu schnell veraltet wäre. Es soll vielmehr dargestellt werden, wie typische Mikrorechnersysteme aussehen, wie die zu Beginn des Abschnittes erwähnten Funktionsblöcke realisiert und wie sie in Bauelementen miteinander kombiniert werden. Man könnte dies als Klassifizierung nach der Anzahl der für ein Mikrorechnersystem erforderlichen Bauelemente bezeichnen.

Die kompaktesten Systeme auf dem Markt sind die 1-Chip-Mikrorechner. Bei diesen Bauelementen ist ein kompletter Rechner, also alle genannten Funktionen, auf einem Chip (Kristallplättchen) integriert. Man benötigt außer der Stromversorgung lediglich einen Quarz bzw. ein RC-Glied zur Taktfrequenzbestimmung. Eine Anzahl von Anschlüssen für Datenein/ausgabe zu den Peripheriegeräten steht zur Verfügung. Die Anzahl dieser Leitungen ist natürlich begrenzt durch die Anzahl der verfügbaren Anschlüsse am Gehäuse. Die Wortlänge dieser Rechner ist 4 bis 8 bit. Der auf dem Chip integrierte Festwertspeicher wird bei der Herstellung maskenprogrammiert, d.h., das Programm eines solchen Rechners wird einmal bei der Herstellung festgelegt und ist dann nicht mehr zu ändern. Es gibt jedoch auch Rechner dieser Art, die mit RePROMs ausgestattet sind, so daß z.B. in der Entwicklungsphase gelöscht und neu eingeschrieben werden kann. Auch der auf dem Chip integrierte Speicherraum ist natürlich begrenzt. Gängige Werte sind 1-k-Byte-Programmspeicher und einige-Dutzend-Byte-Schreib/Lesespeicher. Reicht dieser Speicher nicht aus, sind viele dieser Systeme erweiterungsfähig. Das gleiche gilt für die Interface-Schaltungen, bei denen auch durch zusätzliche Bausteine eine Erweiterung möglich ist.

Solche Rechner sind hauptsächlich geeignet für die große Zahl der einfacheren Anwendungen wie z.B. für fest programmierte Steuerungen oder Kalkulatoren, die in größeren Stückzahlen gefertigt werden, um die Kosten für die Maskenprogrammierung zu rechtfertigen. Diese Kosten werden bei einem solchen Rechner im allgemeinen höher liegen als bei der Maskenprogrammierung eines ROMs allein, da eine wesentlich kompliziertere Maske benötigt wird als bei einem ROM allein.

Andere Systeme benötigen 2 integrierte Schaltungen in ihrer Minimalkonfiguration. Hierbei ist normalerweise die Programmspeichereinheit kombiniert mit I/O-Ports (Ein/Ausgabebausteine) sowie eine gewisse Steuerlogik in einem IC, die die einzelnen Funktionsblöcke in dem zweiten System enthalten. Auch diese Systeme, deren Anwendungsbereich in etwa dem der zuerst angesprochenen Gruppe entspricht, sind normalerweise erweiterbar.

Der MP 8080 ist ein Vertreter aus einer Gruppe von Mikroprozessoren, bei der die CPU in einer Schaltung integriert ist. Die anderen Systemkomponenten werden als weitere integrierte Schaltungen geliefert. Im Falle des MP 8080 sind das ein Taktgeneratorbaustein, verschiedene Festwert- und Schreib/Lesespeicher, eine ganze Reihe verschiedener Interface-Schaltungen sowie Hilfsbausteine, wie der Systemcontroller usw. Mit den verschiedenen integrierten Schaltungen können hier sehr leistungsfähige und flexible Systeme zusammengestellt werden. In dieser Gruppe ist eine Wortlänge von 8 bit am verbreitetsten. Viele dieser Rechner haben leistungsfähige Instruktionssätze und können 64-k-Byte-Speicher adressieren. Sie sind somit bereits für recht komplexe Anwendungen, z.B. in der Prozeßsteuerung, geeignet. Daneben sind Rechnersysteme mit 12 und 16 bit auf dem Markt. Die größere Wortlänge erhöht die Leistung, da einmal mit einem Befehl mehr Informationen parallel verarbeitet werden können und da zum anderen ein längeres Befehlswort einen leistungsfähigeren Instruktionssatz ermöglicht.

Ganz anders strukturiert sind die sog. Bit-Slice-Mikroprozessoren. Bei den bisher erwähnten Systemen hat der Anwender meistens eine fertige CPU mit einem festen Instruktionssatz und einem gegebenen Bussystem zur Verbindung mit der Außenwelt zur Verfügung. Bei den Bit-Slice-Mikroprozessoren muß die CPU aus mehren hochintegrierten ICs zusammengestellt werden. Diese Rechner sind mikroprogrammgesteuert. Der Anwender kann auf diese Weise die

Größe und Leistungsfähigkeit selbst festlegen. So kann die Wortlänge bestimmt werden, und über ein entsprechendes Mikroprogramm kann ein für die gewünschte Aufgabe spezieller Befehlssatz erstellt werden.
Ein Mikroprogramm darf nicht mit dem Anwenderprogramm verwechselt werden. Es handelt sich um ein Programm, das direkten Einfluß auf die logischen Funktionen des Rechners hat. Die Wirkung eines Maschinenbefehles kann somit durch die Folge von Mikrobefehlen oder Mikroinstruktionen vom Anwender bestimmt werden. Der Anwender ist also nicht auf den vom Hersteller vorgegebenen Befehlssatz angewiesen, sondern kann sich seine Befehle für eine bestimmte Anwendung selbst aus Mikroinstruktionen erstellen.
Da die benötigten ICs für die Bit-Slice-Mikroprozessoren in verschiedenen bipolaren Technologien angeboten werden, können Rechnersysteme mit großer Arbeitsgeschwindigkeit erstellt werden. Die zuerst erwähnten Mikroprozessoren und Mikrorechner werden überwiegend in verschiedenen MOS-Technologien hergestellt und sind deshalb langsamer.
Nach diesem kurzen Überblick über die verschiedenen Mikrorechnerarten sollen nun am Beispiel des MP 8080 einige wesentliche Bausteine besprochen werden, die dem Entwickler von Mikrorechnersystemen heute zur Verfügung stehen. Diese Bausteine realisieren eine Reihe von Grundfunktionen in Mikrorechnersystemen, wie z.B. Speicherung oder Anpassung an verschiedene Peripheriegeräte. Solche Bausteine werden in ähnlicher Form von verschiedenen Herstellern angeboten. Die verschiedenen Funktionen werden heute immer mehr in den ICs kombiniert, wie z.B. Speicher und Ein/Ausgabeschaltungen in einem IC. Hier sollen die Bausteine in ihrer grundsätzlichen Funktion und Struktur besprochen werden. Damit ist es für den Anwender nicht mehr schwer, anhand von Datenblättern und Applikationsberichten der Hersteller die verschiedenen Varianten zu verstehen und praktisch einzusehen. Wir verzichten auf detaillierte Anwendungsinformationen, für die die Datenblätter der Hersteller zuständig sind.

9.1 Speicherbauelemente für Mikrorechner

In Mikrorechnern werden fast ausschließlich Halbleiterspeicher verwendet. Die grundsätzlichen Eigenschaften von Speichern wurden bereits in Lehrheft 1 angesprochen. Wir wollen uns daher mit speziellen Fragen zu den im Mikrorechnereinsatz verwendeten Speichern beschäftigen. In Bild 9.1.1 sind die verschiedenen Halbleiterspeicherbauelemente dargestellt.
Ein wichtiges Leistungsmerkmal aller dieser Speichertypen ist die Zugriffszeit. Jeder Mikroprozessor erlaubt bei maximaler Arbeitsgeschwindigkeit eine bestimmte Speicherzugriffszeit.

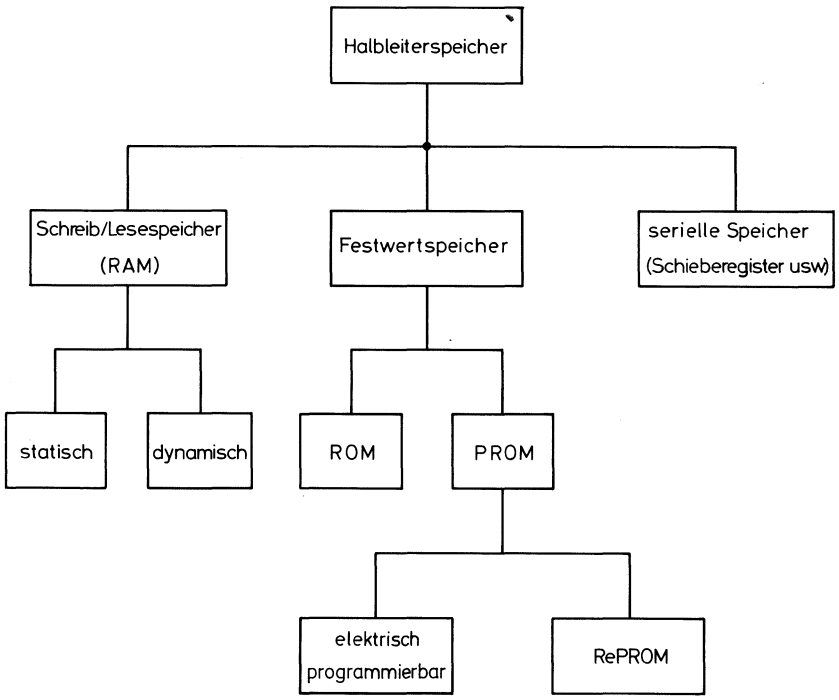


Bild 9.1.1
Halbleiterspeicherbauelemente für Mikrorechner

Ist der Speicher langsamer, wird die Arbeitsgeschwindigkeit des Mikrorechners reduziert. Bei synchronem Datentransfer muß in diesem Fall normalerweise die Systemtaktfrequenz reduziert werden, bei asynchronem Datentransfer ist eine entsprechende Zeitsteuerung erforderlich.

Ein weiteres wichtiges Merkmal eines Speichers ist seine Organisation. Bei Speichern für Mikrorechneranwendungen setzt sich mehr und mehr eine Wortorganisation durch, d.h., unter einer Adresse wird ein Wort von 4 oder 8 bit Länge abgespeichert. Dadurch wird es möglich, bei Systemen mit geringerem Speicherbedarf die Anzahl der integrierten Schaltungen zu reduzieren.

Schreib/Lesespeicher werden in vielen Ausführungsformen angeboten. 2 wichtige Speicherarten sind die statischen und die dynamischen Speicher. Bei statischen Speichern sind die Speicherzellen bistabile Schaltungen, sie bieten beim Einsatz in Mikrorechnern keine besonderen Probleme. Bei dynamischen Speichern wird die Information in Form von Ladungen auf Kondensatoren gespeichert. Da sich diese Kondensatoren im Laufe von einigen Millisekunden entladen, müssen die Speicher periodisch aufgefrischt werden (refresh). Dazu wird ein zusätzlicher Takt verwendet, der in den erforderlichen Zeitabständen einen Auffrischzyklus einleitet. Während dieses Auffrischzyklus kann der Rechner nicht auf den Speicher zurückgreifen, er muß also warten. Diese Wartezeit beträgt typisch etwa 1% bis 2% der gesamten Betriebszeit. Es kann jedoch z.B. bei einer Interruptanforderung zusätzliche Verzögerungen um die Dauer des Auffrischungszyklus geben, die bei der Systemkonzeption berücksichtigt werden müssen. Zur Steuerung des Auffrischvorganges werden spezielle ICs angeboten, so daß auch dynamische Speicher in Mikrorechnersystemen einfach zu verwenden sind. Dynamische Speicher bieten den Vorteil eines sehr geringen Stromverbrauches im Stand-Bye-Betrieb, d.h. in der Zeit, in der das Speicherbauelement nicht selektiert ist. Außerdem kann eine um etwa den Faktor 4 höhere Packungsdichte erreicht werden, also eine um den Faktor 4 höhere Speicherkapazität bei gleicher Fläche des Silizium-Chips.
Beide Schreib/Lesespeicherarten werden in bipolarer und in MOS-Technologie geliefert. Die schnellsten Speicher sind die bipolaren statischen Typen. Industriestandard sind heute statische bipolare Typen mit Zugriffszeiten von 50 ns und weniger.
MOS-Schreib/Lesespeicher sind langsamer, haben aber geringeren Stromverbrauch. MOS-Speicher haben Zugriffszeiten bis etwa 200 ns. Industriestandard sind statische MOS-Speicher mit einer Kapazität von 1 k und dynamische Typen mit 4 k Kapazität. Der nächste technologische Schritt werden statische Speicher mit einer Kapazität von 4 k und dynamische Speicher mit 16 k sein.

Die **Festwertspeicher** lassen sich in 2 Gruppen einteilen: Die beim Herstellungsprozeß mit einer Maske programmierten Typen (ROM) und die vom Benutzer elektrisch zu programmierenden Speicher (PROM). Bei diesen Speichern gibt es dann wieder Typen, die vom Benutzer durch UV-Licht oder elektrisch gelöscht werden können und dann elektrisch neu programmierbar sind.
Bei diesen Speichern ist die Information in Form von Ladungen auf normalerweise hochisolierten Inseln im Siliziumplättchen gespeichert. Beim Löschen und Neuprogrammieren werden diese Inseln umgeladen.
Die nicht löschbaren PROMs werden elektrisch programmiert, entweder durch definiertes Erzeugen interner Kurzschlüsse oder durch Durchbrennen von internen elektrischen Verbindungen. Diese Vorgänge sind irreversibel, so daß eine Neuprogrammierung nicht möglich ist.

Festwertspeicher stehen mit größerer Kapazität zur Verfügung als Schreib/Lesespeicher. Bei den bipolaren ROMs sind heute 2 k Industriestandard, bei den MOS-ROMs 8 und 16 k. Diese Speicher werden vorwiegend wortorganisiert angeboten. Bipolare Festwertspeicher sind schneller als MOS-Speicher, die Zugriffszeiten sind etwa wie bei Schreib/Lesespeichern.

Unter den maskenprogrammierten Festwertspeichern sind verschiedene kompatibel in den Anschlüssen und den elektrischen Daten mit reprogrammierbaren Festwertspeichern. Dies ist für den Einsatz dieser Elemente in Mikrorechnern von großer praktischer Bedeutung. In der Entwicklungsphase des Gerätes werden die RePROMs verwendet, so daß das Programm im Festwertspeicher noch geändert werden kann. Wenn das Programm dann fertig ist und das Gerät soll in Serie gehen, so wird das RePROM durch das entsprechende maskenprogrammierte ROM ersetzt, das dann bei größeren Stückzahlen erheblich billiger ist als das RePROM. Dieses Verfahren wurde beispielsweise bei der Entwicklung des Experimentiersystems verwendet.

Eine besondere Gruppe sind die **seriellen Speicher,** wie z.B. die Schieberegister. Sie sind die preiswertesten Halbleiterspeichermedien. Sie eignen sich zur Speicherung größerer Datenmengen, wenn die Zugriffszeit nicht kritisch ist. Diese Zugriffszeit kann recht lange sein, da ein gewünschtes bit eventuell zuerst durch das ganze Schieberegister geschoben werden muß, bis es am Ausgang erreichbar ist. Solche zirkularen Schieberegisterspeicher werden deshalb nur in besonderen Fällen in Mikrorechnersystemen eingesetzt.

9.2 Ein/Ausgabebausteine für Mikroprozessoren

Im Abschnitt 8. sind verschiedene Möglichkeiten des Datentransfers zwischen dem Mikrorechner und den peripheren Schaltungen sowie die grundsätzlichen Eigenschaften der dafür erforderlichen Schnittstellen (Interfaces) zwischen dem Bussystem des Rechners und dem Datenkanal zum Peripheriegerät besprochen worden. Diese Ein/Ausgabeaufgaben sind von großer Bedeutung für den praktischen Einsatz von Mikroprozessoren. Die Hersteller der Mikroprozessoren bieten deshalb eine Vielzahl verschiedener Bausteine an, die heute zum großen Teil sehr leistungsfähig sind. Diese Bausteine sind teilweise programmierbar vom zentralen Prozessor, und sie arbeiten dann weitgehend selbständig und entlasten so den zentralen Prozessor von der Routinearbeit der Ein/Ausgabe. In diesem Abschnitt sollen für die verschiedenen im Abschnitt 8. behandelten Arten des Datentransfers einige Ein/Ausgabebausteine besprochen werden. Die von den verschiedenen Herstellern gelieferten Bausteine sind zwar primär zur Verwendung mit dem jeweiligen Mikroprozessor gedacht, in vielen Fällen ist es aber durchaus möglich, die Ein/Ausgabebausteine und Prozessoren verschiedener Hersteller zu kombinieren.

9.2.1 8-bit-Ein/Ausgabebaustein (8212)

Dieser Baustein besteht aus einem 8-bit-Register mit Tri-State-Ausgangstreibern und der erforderlichen Steuer- sowie Select-Logik (Bild 9.2.1.1).

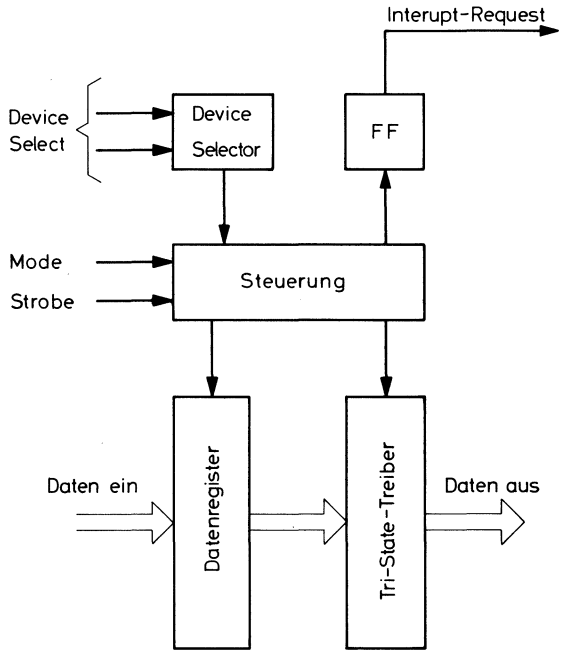


Bild 9.2.1.1
Blockschaltbild

Das zusätzliche FF erzeugt ein Interruptanforderungssignal. Der Baustein ist im wesentlichen für die parallele Eingabe oder Ausgabe von 8-bit-Datenwörtern gedacht. Aufgrund seiner Tri-State-Ausgänge kann er direkt mit dem Datenbus des Rechners verbunden werden. Der Baustein kann

auch als Speicherregister, Bustreiber oder Multiplexer verwendet werden, so daß er für eine Vielzahl von Aufgaben in MP-Systemen einsetzbar ist.
Die Steuersignale haben folgende Bedeutung:

Device/Select: Diese beiden Signale werden zur Adressierung des Bausteines (Device Selection) verwendet.

Mode: Mit diesem Eingang wird die Betriebsart des Bausteines gesteuert, d.h., ob er zur Ein- oder Ausgabe verwendet wird.

Strobe: Mit diesem Eingang wird das Interrupt-Request-FF gesteuert und zusätzlich in der Betriebsart Eingabe das Datenregister getaktet.

In Bild 9.2.1.2 ist die grundsätzliche Schaltung des Bausteines als Ein- und als Ausgabe-Interface gezeigt.

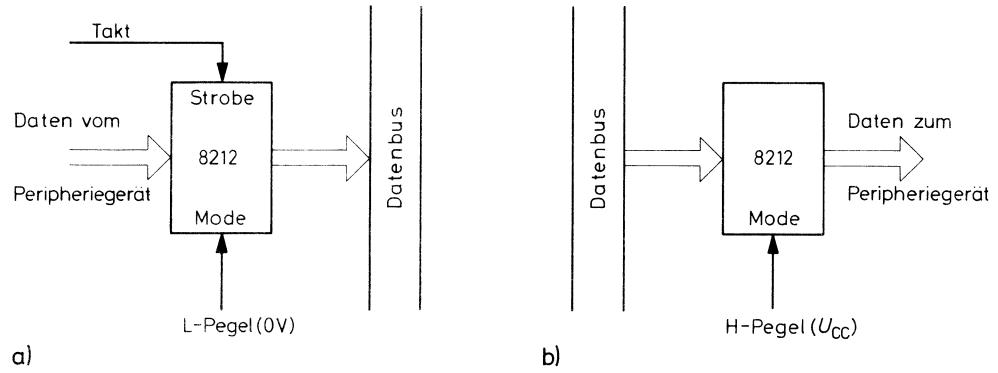


Bild 9.2.1.2
Grundsätzliche Schaltungen des Bausteines 8212
a) Prinzipschaltung eines Eingabe-Interfaces
b) Prinzipschaltung eines Ausgabe-Interfaces

Weitere Anwendungsmöglichkeiten können dem Datenblatt entnommen werden. Ein praktisches Beispiel für den Einsatz dieses Bausteines ist der MP-Experimenter. Alle Schalter und Lampen sind über 8212-Interface-Schaltungen angeschlossen.

9.2.2 Programmierbarer Ein/Ausgabebaustein (8255)

Bausteine dieser Art gehören zu den universellsten Komponenten eines Mikrorechnersystems. Diese Bauelemente haben Eigenschaften, die sie erheblich von anderen Bausteinen der Digitaltechnik unterscheiden. Bild 9.2.1.1 zeigt ein Blockschaltbild dieses Bausteines.
Die Funktion des Bausteines und seiner Anschlüsse wird durch Programmierung vom zentralen Prozessor aus festgelegt. So kann z.B. ein Anschluß unter Kontrolle des Programms als Eingangs- oder als Ausgangsleitung verwendet werden. Der 8255 hat 24 Ein/Ausgabeleitungen, die in 2 Gruppen von je 12 Leitungen programmiert werden können und die in 3 verschiedenen Betriebsarten zu verwenden sind.
Die Programmierung erfolgt durch ein 8-bit-Befehlswort, das die jeweilige Betriebsart festlegt. Dieses Befehlswort wird vom zentralen Prozessor in ein Befehlsregister in dem Baustein geladen.
Die Betriebsart 0 (Mode 0) wird für normale Ein/Ausgabeoperationen verwendet. Hierbei sind die 24 Ein/Ausgabeleitungen in Gruppen von 4 bzw. 8 Leitungen für Eingabe oder Ausgabe programmierbar. Die Betriebsart 1 (Mode 1) ist für Datentransfer mit Synchronsignalen vorgesehen. In jeder Zwölfergruppe sind 8 Datenleitungen für parallele Eingabe oder Ausgabe von 8-bit-Datenwörtern und 4 Leitungen für Steuersignale vorgesehen. Beide Gruppen können für sich für Eingabe oder Ausgabe programmiert werden.

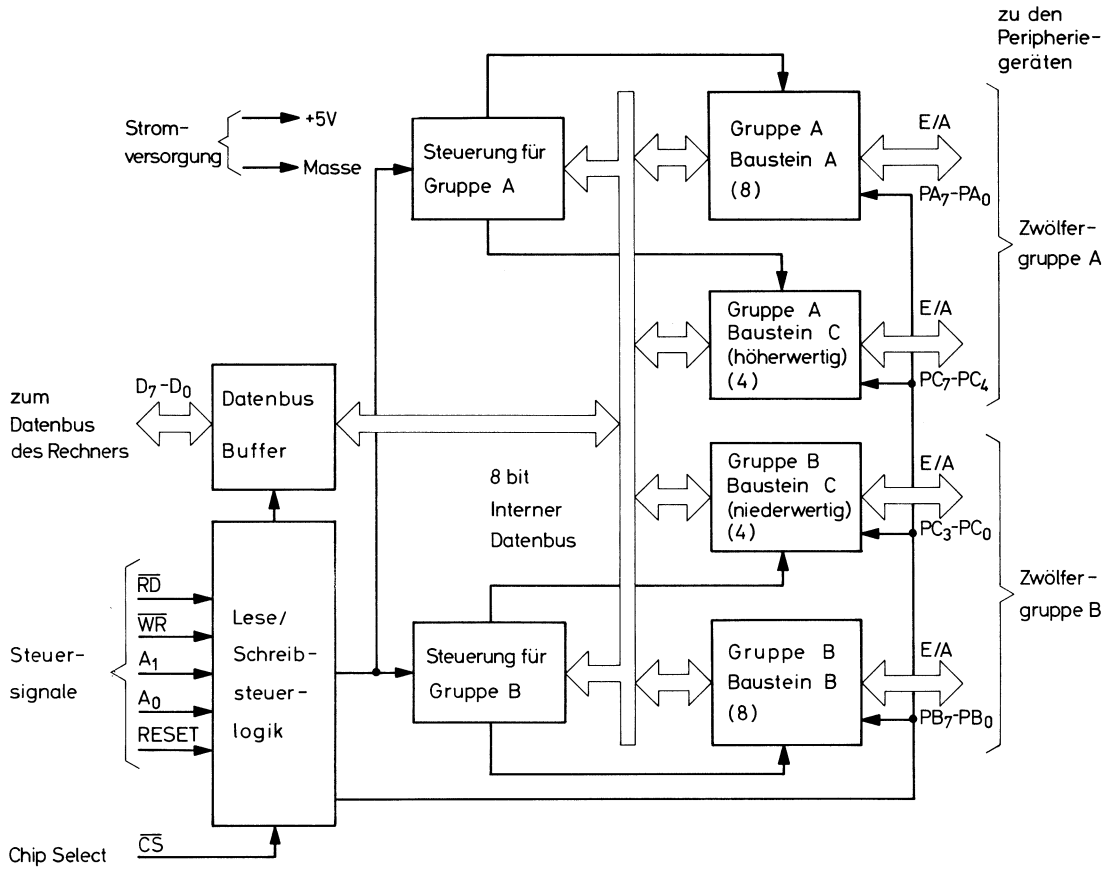


Bild 9.2.2.1
Blockschaltbild des programmierbaren Ein/Ausgabebausteines 8255

Bei Betriebsart 2 (Mode 2) werden 8 Leitungen aus einer der beiden Zwölfergruppen für einen bidirektionalen Bus verwendet, also für Datentransfer in beiden Richtungen. Dazu kommen noch 5 Steuersignale, nämlich die restlichen 4 aus der Zwölfergruppe und eines aus der anderen Gruppe. In dieser Betriebsart eignet sich der 8255 z.B. zum Verbinden zweier Rechner über einen bidirektionalen 8-bit-Datenbus.
Insgesamt sind drei 8-bit-Ein/Ausgabeeinheiten vorhanden, wobei die Einheit C (Gruppe C) in 2 Hälften mit 4 bit zerfällt. Die Verbindung zum Datenbus des Rechners erfolgt über den Datenbuffer. Dieser ist bidirektional und 8 bit breit. Die Lese/Schreibsteuerlogik steuert den internen und externen Datenverkehr. Sie bekommt und gibt Steuerbefehle über den Adreß- und Steuerbus des Rechners. Die Steuerung der A- und B-Gruppen wird abhängig vom Befehlswort in diesen Steuereinheiten durchgeführt.
Die Steueranschlüsse haben folgende Bedeutung:

$\overline{CS}$: Bausteinauswahl (Chip Select). Wird dieser Eingang auf LOW-Pegel gebracht, kann ein Informationsaustausch zwischen dem MP 8080 und dem Baustein 8255 erfolgen.

$\overline{RD}$: Lesen (Read). LOW-Pegel an diesem Eingang bewirkt, daß Daten über den Datenbus zum MP 8080 gelangen können.

$\overline{WR}$: Schreiben (Write). Bei LOW-Pegel kann der MP 8080 Daten in den 8255 einschreiben.

RESET: HIGH-Pegel setzt alle Register zurück und bringt die Kanäle (A bis C) in die Betriebsart Eingabe.

A_0 und A_1: Kanalauswahl. In Verbindung mit $\overline{RD}$ und $\overline{WR}$ können die 3 Kanäle oder das Steuerwortregister selektiert werden.

Folgende Betriebsarten sind grundsätzlich möglich:

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	**Eingabeoperationen (Lesen)**
0	0	0	1	0	Kanal A→Datenbus
0	1	0	1	0	Kanal B→Datenbus
1	0	0	1	0	Kanal C→Datenbus
					Ausgabeoperationen (Schreiben)
0	0	1	0	0	Datenbus→Kanal A
0	1	1	0	0	Datenbus→Kanal B
1	0	1	0	0	Datenbus→Kanal C
1	1	1	0	0	Datenbus→Steuerlogik
					Funktionen nicht ausgewählt
x	*x*	*x*	*x*	1	Datenbus→hochohmiger Zustand
1	1	0	1	0	ungültige Bedingung

Weitere Einzelheiten können aus dem Datenbuch entnommen werden.

9.2.3 Programmierbarer Datensender/empfänger (8251)

Diese Bausteine werden zur Abwicklung serieller Datentransfers zwischen Mikroprozessoren und Peripheriegeräten wie z.B. Fernschreibern verwendet. Eine Vielzahl solcher Bausteine, für die sich die Bezeichnung UART (**U**niversal Synchronous-**A**synchronous **R**eceiver/**T**ransmitter) eingebürgert hat, sind heute auf dem Markt. Die Verwendung dieser Bausteine soll am Beispiel des Typs 8251 dargestellt werden (Bild 9.2.3.1).

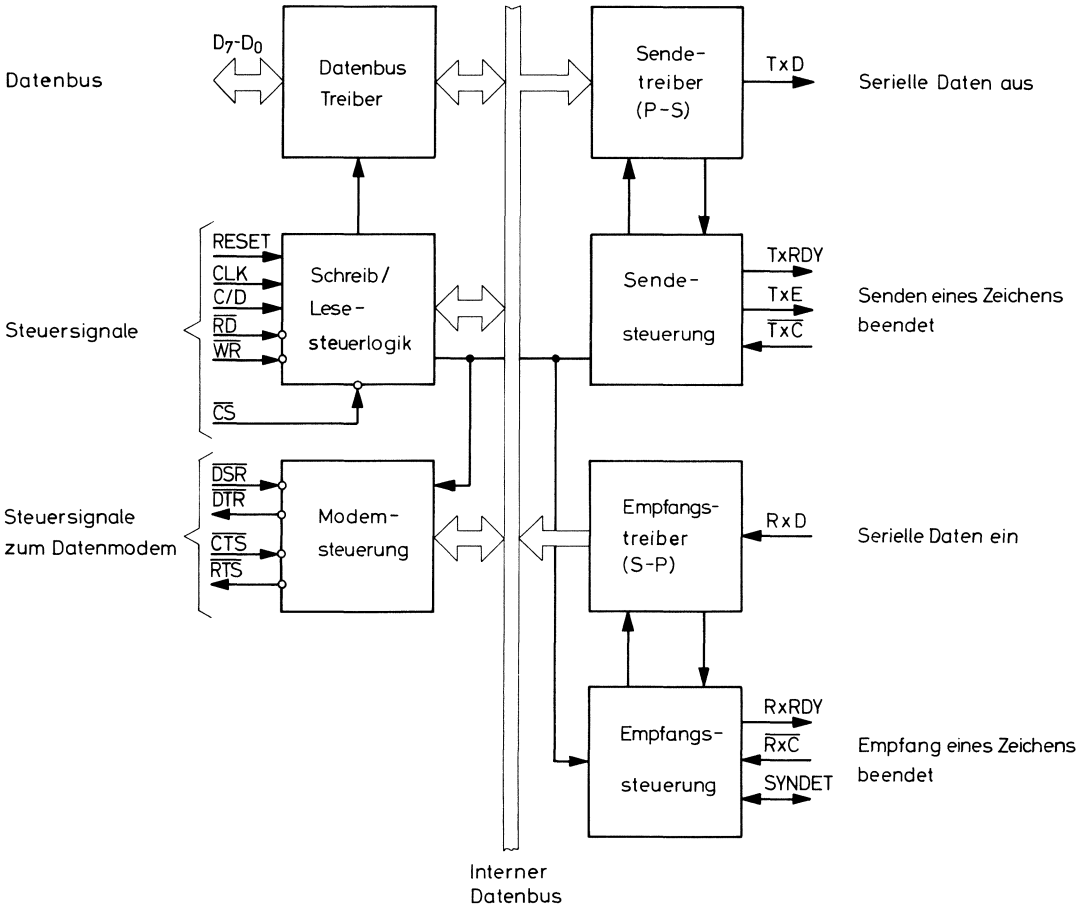
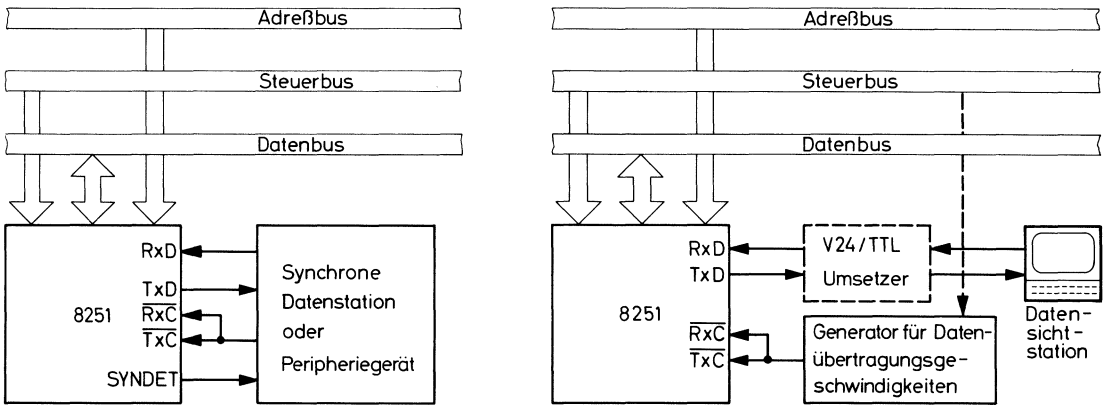


Bild 9.2.3.1
Blockschaltbild des UART 8251

Der Baustein kann parallele Daten von der CPU übernehmen und sie in einen seriellen Datenstrom (T × D) für die Übertragung umsetzen (Senderteil). Gleichzeitig kann ein serieller Datenstrom (R × D) empfangen und parallel zur CPU übertragen werden (Empfängerteil). Die Länge der übertragenen Zeichen ist programmierbar zwischen 5 und 8 bit, die Zahl der Stopp-bits im Datenstrom kann spezifiziert werden. Der Baustein führt ebenfalls eine Paritätsprüfung durch und gibt gegebenenfalls eine Fehlermeldung an die CPU.
Die Daten können entweder synchron übertragen werden,d.h., das periphere Gerät bestimmt die Übertragungsrate, oder die Daten können asynchron mit vorgegebener Taktfrequenz übertragen werden.
Die Programmierung erfolgt durch Laden eines Befehlsregisters von der CPU aus. Der Baustein kann auf diese Weise für eine Vielzahl von Datenformaten eingesetzt werden.
Im Blockschaltbild sind rechts vom internen Datenbus der Sende- und Empfangsteil mit ihren Ablaufsteuerungen und den seriellen Datenein- und Ausgängen dargestellt. Der Baustein teilt der CPU über die Steuerleitungen T × RDY und R × RDY mit, wenn ein Zeichen gesendet oder empfangen wurde. Die CPU kann den kompletten Zustand des Bausteines zu jeder Zeit abfragen. Dazu gehören z.B. die Fehlermeldung und das Signal SYNDET (Synchronisation Detect), das beim synchronen Empfangsbetrieb die ordnungsgemäße Detektion der Synchronisierzeichen meldet.
Der Baustein hat Tri-State-Bustreiber und kann somit direkt an den Datenbus des Rechners angeschlossen werden. Die Schreib/Lesesteuerung wertet die Steuersignale von der CPU aus. Sie enthält auch die Chip-Select-Logik.
Die Modemsteuerung erleichtert den Anschluß von Datenmodems aller Art durch eine Anzahl von entsprechenden Steuerein/ausgängen. Unter einem Modem versteht man in der Nachrichtentechnik einen **Mo**dulator + **Dem**odulator als Wandler für Datenübertragung.
In Bild 9.2.3.2 sind 2 typische Anwendungsfälle für den Baustein 8251 dargestellt.



a) b)

Bild 9.2.3.2
Typische Anwendungen für den Baustein 8251
a) Synchrone Schnittstelle zu einer Datenstation oder einem Peripheriegerät
b) Asynchrone serielle Schnittstelle zu einer Datensichtstation

9.2.4 Interrupt-Controller (8214)

Dieser Baustein erleichtert den Entwurf von Mikrorechnersystemen mit Interruptverarbeitung in mehreren Prioritätsebenen. Dieser Interrupt-Controller kann 8 Prioritätsebenen verarbeiten. Bei mehreren gleichzeitig einlaufenden Interruptanforderungen bestimmt er die Anforderung mit der höchsten Priorität, vergleicht diese Priorität mit einem programmgesteuerten Statusregister, das die Interruptmaske enthält, und gibt eine Interruptanforderung zum Rechner aus, wenn die Priorität der Anforderung der Peripheriegeräte höher ist als die in der Interruptmaske festgelegte Priorität. Mit der Interruptanforderung an den Rechner gibt der Baustein noch eine Adresse aus, den sog. Interruptvektor. Dieser Interruptvektor spezifiziert die Startadresse der Service-Routine, die das unterbrechende Peripheriegerät bedient. Bei der Reaktion auf die

Interruptanforderung wird der Rechner die durch den Interruptvektor festgelegte Interrupt-Routine ausführen. Ein Blockschaltbild dieses Bausteines ist in Bild 9.2.4.1 dargestellt.

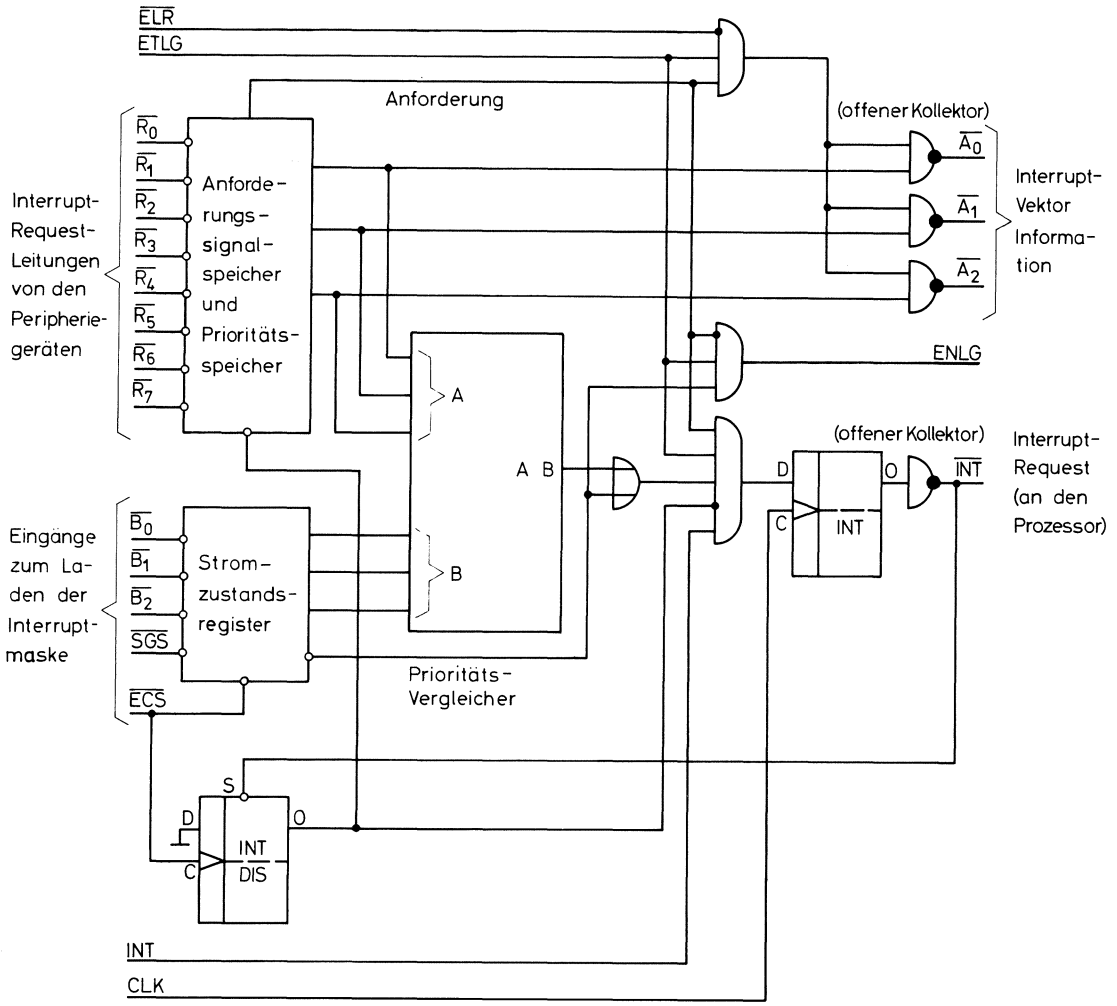


Bild 9.2.4.1
Blockschaltbild des Interrupt-Controllers 8214

Die 8 Interrupt-Request-Leitungen von den Peripheriegeräten kommen zu einem Prioritätskodierer, der die Anforderung mit der höchsten Priorität auswählt und die Nummer dieser Prioritätsebene in eine 3-bit-Zahl codiert. Die Leitung $\overline{R_7}$ hat die höchste, $\overline{R_0}$ die niedrigste Priorität. Wenn also z.B. gleichzeitig auf den Leitungen $\overline{R_1}$ und $\overline{R_4}$ Interruptanforderungen kommen, wird am Ausgang des Prioritätsencoders die Zahl $1\,0\,0_2$ anstehen. Diese Zahl wird über die Ausgänge $\overline{A_0}$ bis $\overline{A_2}$ als Interruptvektorinformation nach außen gegeben, um dem Rechner die Nummer des unterbrechenden Peripheriegerätes mitzuteilen. Außerdem wird diese Zahl in dem Prioritätskomparator mit dem Inhalt des Current-Status-Registers (Stromzustandsregister), also der Interruptmaske, verglichen. Durch diesen Vergleich wird sichergestellt, daß nur Interrupt-Requests mit höherer Priorität durchkommen, als im Current-Status-Register spezifiziert ist. Der Name Current-Status-Register rührt daher, daß normalerweise bei der Programmierung von Interruptsystemen mit mehreren Prioritätsebenen in diesem Register jeweils die Priorität der momentan laufenden Interrupt-Routine enthalten ist. Kommt dann eine neue Interruptanforderung von einem Peripheriegerät, so kann diese die laufende Interrupt-Routine nur dann unterbrechen, wenn die Priorität der neuen Anforderung höher ist als die Priorität der laufenden Routine. Das Current-Status-Register wird unter Programmkontrolle gesetzt, d.h., in jeder Service-Routine wird die betreffende Priorität in dieses Register geladen. Die Programmierung ist völlig freizügig, es kann je nach Erfordernissen jeder Zahlenwert in dieses Register geladen werden. Jeder Interrupt-Request, der von dem Interrupt-Controller an den Prozessor gegeben wird, setzt das Interrupt-Disable-FF und verhindert damit, daß weitere Interrupt-Requests von den Peripheriegeräten durchkommen. Erst wenn das Current-Status-Register neu geladen wird, wird dieses FF wieder gelöscht, so daß Interruptanforderungen wieder bearbeitet werden.

Das Interrupt-FF dient dazu, die Interruptanforderung zum Prozessor mit einem externen Takt, z.B. dem Systemtakt, zu synchronisieren.
Bild 9.2.4.2 zeigt die Verwendung dieses Bausteins in einem MP 8080-Interruptsystem.

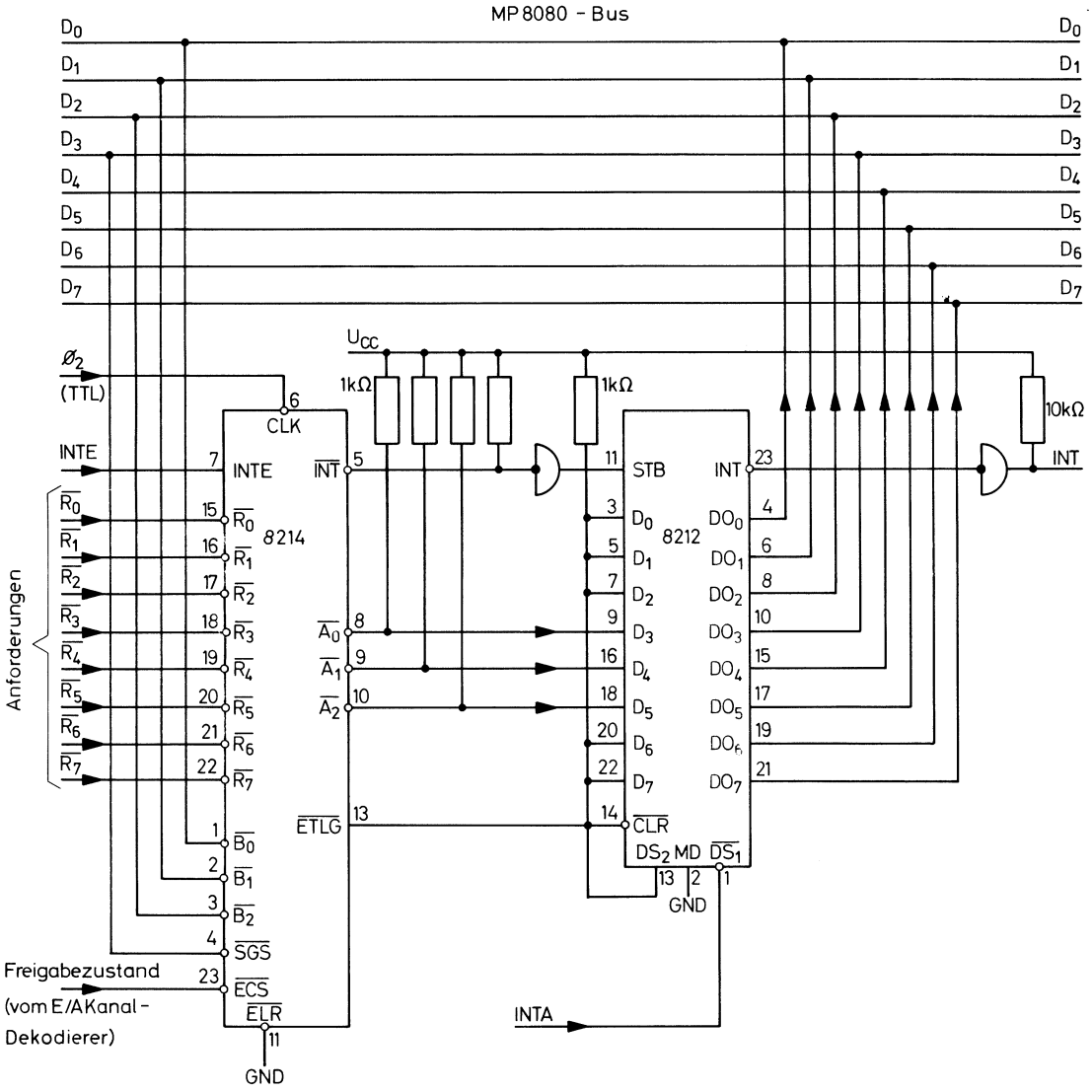


Bild 9.2.4.2
Interrupt-Controller für 8 Prioritätsebenen beim MP 8080

Beim 8080 wird ein zusätzlicher 8212-Ein/Ausgabebaustein verwendet, der in dieser Anwendung als Bustreiber eingesetzt wird. Dieser 8212 bringt im Falle eines Interrupts den entsprechenden RST-Befehl auf den Datenbus. Der RST-Befehl hat den Code 1 1 n n n 1 1 1. Die Zahl n n n wird über die Ausgänge $\overline{A_0}$ bis $\overline{A_2}$ bestimmt, die restlichen Eingänge des 8212 liegen aus L-Pegel. Die Interrupt-Routine, die dem Peripheriegerät 0 zugeordnet ist, beginnt somit bei Adresse 0, die Routine zum Peripheriegerät 1 bei 8 usw.

9.2.5 DMA-Controller 8257

Dieser Baustein ist ein Direct-Memory-Access-Controller mit 4 Kanälen. Nach einem DMA-Request von einem Peripheriegerät erzeugt er das Steuersignal (HRQ), um die CPU in den Wartezustand zu versetzen, und gibt dann das DMA-Acknowledge-Signal an das Peripheriegerät. Der Baustein enthält eine Prioritätslogik, um mehrfache DMA-Requests aufzulösen. Jeder Kanal enthält einen 16-bit-Zähler für die Speicheradressen. Die eine Hälfte dieses Zählers wird über die Adreßleitungen, die andere Hälfte im Zeitmultiplex über die Datenleitungen ausgegeben.

Um also die vollständige 16-bit-Speicheradresse zu bekommen, ist ein zusätzliches 8-bit-Register erforderlich, um Daten und Adressen zu „demultiplexen". Der Baustein erzeugt dann die erforderlichen Steuersignale für den Speicher und das Peripheriegerät, so daß das Peripheriegerät die Daten direkt in den Speicher schreiben bzw. vom Speicher lesen kann. Jedem Kanal ist außerdem ein 14-bit-Wortzähler zugeordnet, so daß Datenblöcke von maximal 16 k Byte Länge übertragen werden können. Alle Zähler werden per Programm vor Beginn eines Tranfers geladen. Bild 9.2.5.1 zeigt das Blockschaltbild dieses Bausteines.

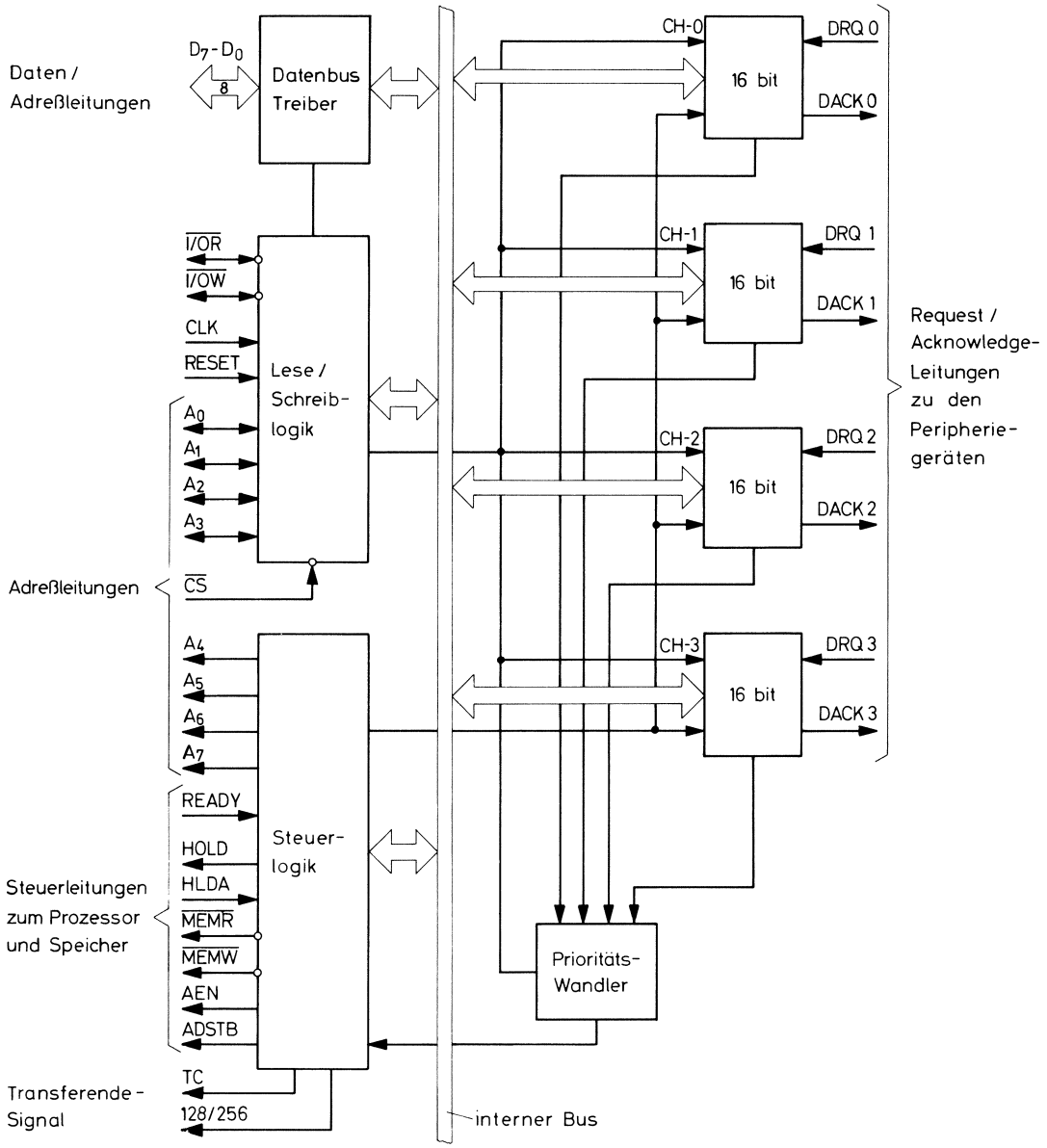


Bild 9.2.5.1.
Blockschaltbild des DMA-Controllers 8257

Die 4 DMA-Kanäle CH-0 bis CH-3 enthalten jeweils einen Adreßzähler und einen Wortzähler. Jedem dieser Register ist auf den Adreßleitungen A_0 bis A_3 eine bestimmte Adresse zugeordnet, so daß die Register selektiert und per Programm geladen werden können. Dabei wird auch die Betriebsart jedes Kanales (Eingabe-Ausgabe-Prüfen) spezifiziert. Von jedem Kanal gehen die beiden Steuerleitungen DMA-Request (DRQ 0 bis DRQ 3) bzw. DMA-Acknowledge (DACK 0 bis DACK 3) zu den Peripheriegeräten.
Der Datentreiber ist bidirektional und in Tri-State-Logik ausgeführt. Die Register in den Kanälen und das Mode-Set-Register in der Steuerlogik können über diese Datenleitungen von der CPU geladen und gelesen werden, so daß die CPU auch den augenblicklichen Zustand des DMA-Controllers abfragen kann.

Die Lese/Schreiblogik steuert das Laden bzw. Lesen der internen Register, sie dekodiert also die Adreßleitungen A_0 bis A_3 und die entsprechenden I/O-Steuerleitungen $\overline{I/OR}$ und $\overline{I/OW}$. Bei einem DMA-Transfer werden hier die Steuersignale für das Peripheriegerät ($\overline{I/OR}$ und $\overline{I/OW}$) sowie die entsprechenden Steuersignale für den Speicher ($\overline{MEMR}$ und $\overline{MEMW}$) erzeugt.

Die Steuerlogik steuert den internen Ablauf im Baustein. Diese Baugruppe enthält das Mode-Set-Register, mit dem verschiedene Betriebsarten, wie feste oder rotierende Prioritätszuweisung u.a.m., per Programm festgelegt werden können. Außerdem ist ein Statusregister vorhanden. Über dieses Register kann der Rechner u.a. abfragen, welcher der Kanäle seinen Transfer beendet hat, d.h., welcher Wortzähler übergelaufen ist.
Bild 9.2.5.2 zeigt die Verbindung des DMA-Controllers zu dem Bussystem des Rechners. Ein zusätzlicher 8212-Baustein wird verwendet, um den Teil der Adreßinformation, der über die Datenleitungen ausgegeben wird, zwischenzuspeichern. Der 8212 wird in diesem Fall also als Latch verwendet.

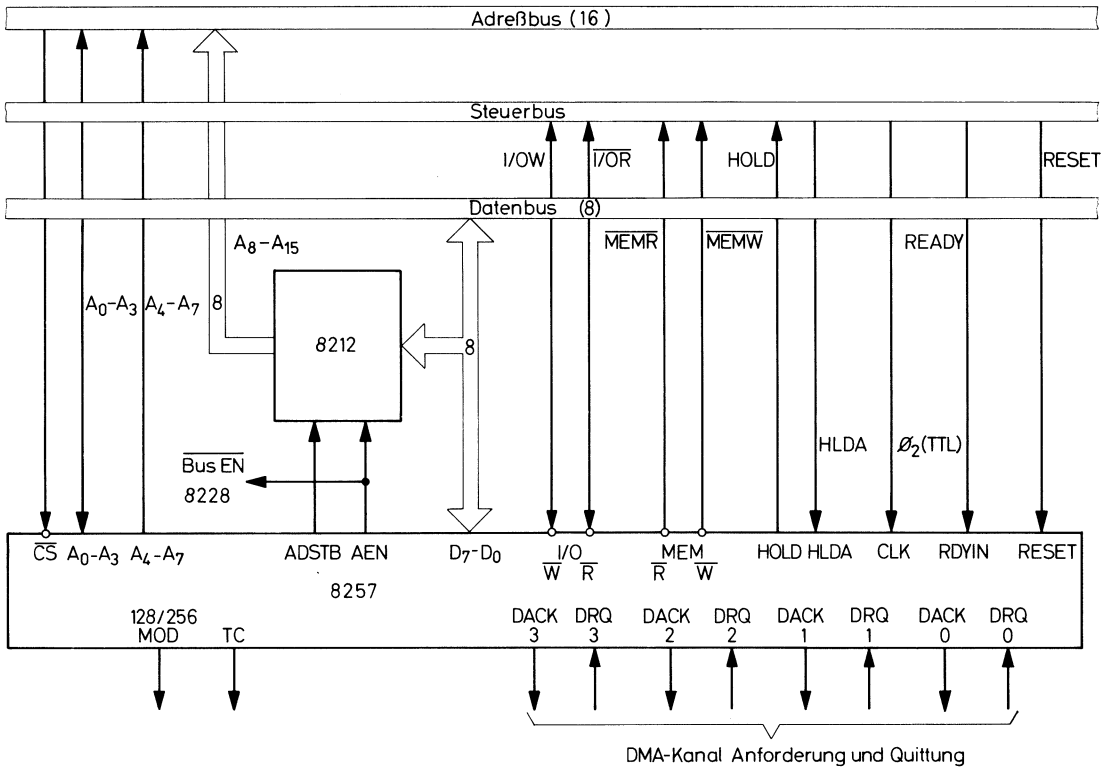


Bild 9.2.5.2
Die Verbindung des DMA-Controllers 8257 zum Bussystem des Mikroprozessors 8080

9.2.6 Programmable Floppy-Disk-Controller (8271)

Die bisher besprochenen Bausteine waren allgemein anwendbar für verschiedene Arten paralleler oder serieller Datentransfers zwischen dem Mikrorechner und beliebigen Peripheriegeräten. Die Floppy-Disk-Controller sind nun Beispiele von Bausteinen, die den Bau von Interfaces zwischen einer bestimmten Art von Peripheriegeräten, nämlich den Floppy-Disks und dem Mikrorechner, ermöglichen. Ein Floppy-Disk ist ein einfacher preiswerter Wechselplattenspieler, dessen Wechselplatten aus dünnen, biegsamen Kunststoffolien bestehen.
Diese Bausteine sind so entwickelt, daß mit einer kleinen Anzahl von integrierten Schaltungen (10 bis 20) ein vollständiges Interface zwischen einer Vielzahl von verschiedenen Floppy-Disk-Laufwerken und einem Mikrorechner hergestellt werden kann. Ein in konventioneller Technik aufgebauter Floppy-Disk-Controller ist erheblich umfangreicher, er würde in der Größenordnung von 150 bis 200 integrierte Schaltungen erfordern.
Der Floppy-Disk-Controller 8271 kann zur Verbindung von maximal 4 Laufwerken mit einem 8-bit-Mikrorechner verwendet werden. Er übernimmt eine Vielzahl von Aufgaben bei der Steuerung dieser Laufwerke und entlastet damit die CPU.

Dieser Baustein stellt eine ganze Reihe leistungsfähiger Datentransfer- und Datenkontrollbefehle zur Verfügung, wie READ DATA, WRITE DATA, VERIFY DATA usw. Ein Transfer läuft in 3 Phasen ab. Die 1. Phase ist die Befehlsphase. Hier lädt der Mikrorechner die Befehle und Transferparameter in die Befehls- und Parameterregister des Floppy-Disk-Controllers. In der 2. Phase führt der Controller diese Befehle selbständig aus. Dabei ist keine Aktion von der CPU erforderlich. Die 3. Phase ist die Ergebnisphase. Der Controller teilt der CPU mit, daß der Befehl ausgeführt ist. Die CPU wird nun die Statusregister des Controllers lesen. Bild 9.2.6.1 zeigt ein Blockdiagramm dieses Bausteines.

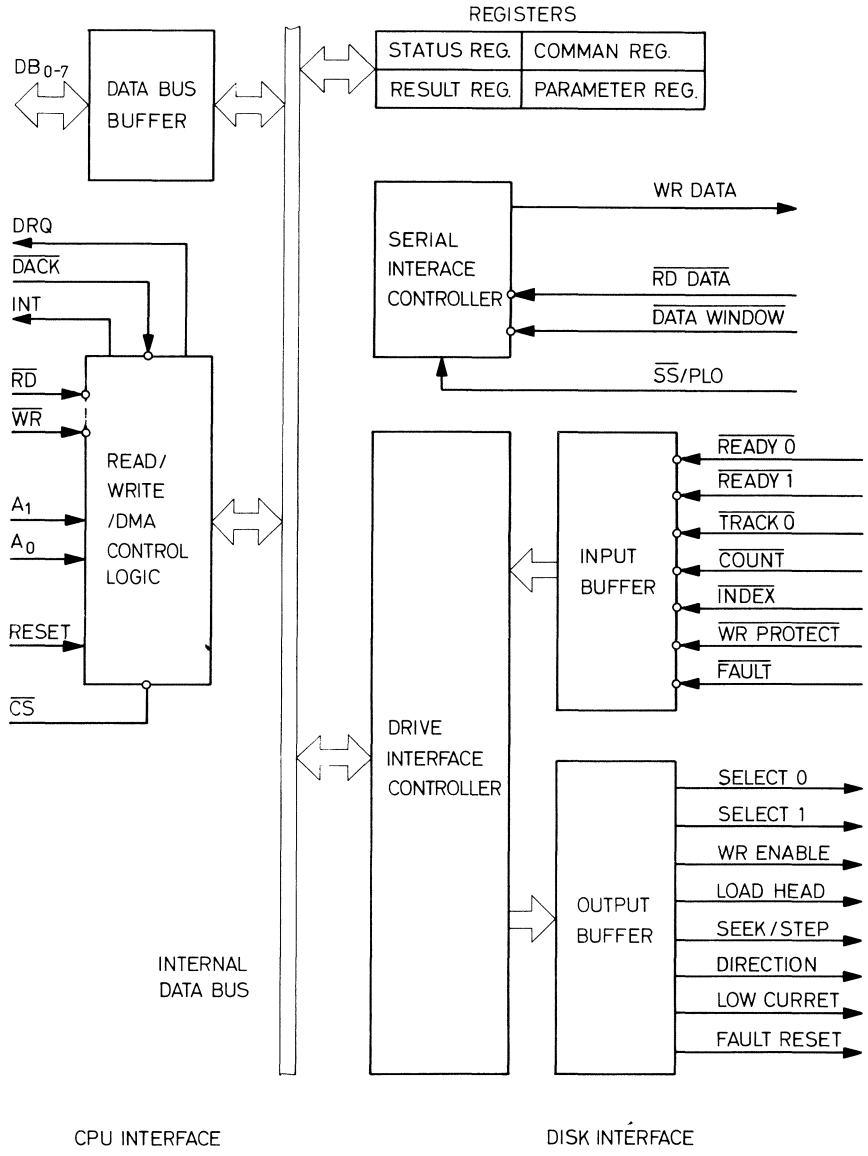


Bild 9.2.6.1
Blockschaltbild des Floppy-Disk-Controllers

Die Daten werden vom Serial-Interface-Controller seriell zur Platte übertragen bzw. von dort gelesen. Der Drive-Interface-Controller steuert das Laufwerk. Der Datentransfer zum Rechner kann im Interrupt-Betrieb oder mit DMA erfolgen. Dazu ist dann ein zusätzlicher DMA-Controller-Baustein erforderlich.

Fragen zu Abschnitt 9.

1. Was verstehen Sie unter einem 1-Chip-Mikrorechner?

2. Welche Funktionsblöcke sind bei einem MP 8080 auf einem Chip integriert?

3. Nennen Sie 2 wichtige Kenngrößen von Halbleiterspeichern im Hinblick auf den Einsatz in Mikrorechnersystemen!

4. Was ist beim Einsatz von dynamischen Schreib/Lesespeichern zu beachten?

5. Wie erfolgt in Bild 9.2.1.2 grundsätzlich die Umschaltung des Ein/Ausgabebausteins 8212 von Ein- in Ausgabebetrieb und umgekehrt?

6. Was verstehen Sie unter dem Begriff UART?

7. Erklären Sie den Begriff Interruptvektor!

8. Was versteht man in der Nachrichtentechnik unter dem Begriff Modem?

10. Systemplanung

In diesem Abschnitt sollen einige Fragen angeschnitten werden, die bei der Planung und beim Entwurf von Systemen und Geräten bedacht werden müssen.

10.1 Auswahl des technischen Konzeptes

Eine sehr wichtige Frage ist zunächst, was ist die günstigste Lösung, ist das der Einsatz eines Mikrorechners oder ist eine andere Lösung günstiger? Bei der Diskussion dieser Frage müssen wir zunächst überlegen, welche Alternativen es gibt. Eine bestimmte Funktion können wir mit den bekannten Standardschaltungsfamilien realisieren. Hierzu gehören die SSI-Schaltungen (Small-Scale-Integration) mit etwa 4 Gattern oder 1 bis 2 Flipflops pro Baustein und die MSI-Schaltungen (Medium-Scale-Integration), bei denen größere Komplexe, wie Zähler, Multiplexer usw., in einem Baustein untergebracht sind.
Eine andere Möglichkeit sind die kundenspezifizierten LSI-Schaltungen (Large-Scale-Integration), bei denen entsprechend den Anforderungen des Anwenders eine Vielzahl von SSI- und MSI-Funktionen in einem Baustein integriert sind. Der Anwender bekommt hier also einen hochspezialisierten Baustein, der für sein Problem maßgeschneidert ist. Daneben gibt es die standardisierten LSI-Schaltungen, bei denen die gewünschten Funktionen durch eine Programmierung erzeugt werden. Der wichtigste Vertreter dieser Gruppe ist der Mikroprozessor. Eine andere Art der standardisierten LSI-Schaltungen sind die PLAs (Programmable Logic Arrays). Daneben sollte bei der Planung eines Gerätes aber auch geprüft werden, ob nicht die Verwendung eines fertigen Rechners die günstigste Lösung ist. Solche Rechner werden heute in vielen Versionen angeboten, als komplette Minicomputer einer Karte (Stripped-Down-Mini), als fertige Mikrorechner auf einer Karte, als Bausatzsystem.
Im folgenden sollen einige wichtige technische Parameter, wie Funktionsgeschwindigkeit, Leistungsbedarf, Raumbedarf, Zuverlässigkeit, Änderbarkeit, Ersatzteilbeschaffung, sowie einige Kostenfaktoren, wie Bauelementekosten, Entwicklungskosten, Implementierungskosten, für die verschiedenen Konzeptionen angegeben und verglichen werden.

Integrierte Standardschaltungen (SSI und MSI)

Bei diesem Konzept kann der Entwickler unter verschiedenen Schaltungsfamilien auswählen. Diese Familien unterscheiden sich in ihrer Funktionsgeschwindigkeit, im Leistungsbedarf, in ihrer Störempfindlichkeit usw. Der Entwickler kann außerdem die Organisation seines Systems frei wählen (z.B. parallele oder serielle Verarbeitung), so daß er seine Lösung in Geschwindigkeit und Leistungsbedarf an die Anforderungen flexibel anpassen kann. Die Realisierung erfordert jedoch eine große Zahl von Bausteinen und Konstruktionselementen wie Leiterplatten, Stecker usw. Daraus ergibt sich ein hoher Raumbedarf. Die große Zahl externer und interner Verbindungen (Lötstellen, Steckkontakte, Bondstellen im IC) begrenzt die Zuverlässigkeit dieser Geräte.

Bei einer vernünftigen Schaltungskonzeption (klar definierte Baugruppen) ist die Entwicklungszeit recht kurz und die Änderbarkeit gut, da nur einzelne Baugruppen geändert werden müssen. Die Ersatzteilbeschaffung bietet keine Probleme, die verwendeten Bausteine werden von vielen unabhängigen Herstellern gefertigt und haben einen sehr breiten Markt.
Berechnet man bei diesem Konzept die gesamten Kosten pro eingebauter IC-Funktion, so zeigt sich, daß abgesehen von sehr kleinen Stückzahlen, wo die Entwicklungskosten entscheidend sind, die Implementierungskosten, also die Kosten für die Leiterplatten, die Verdrahtung usw. und die Kosten der Bauelemente dominieren. Beide Kostenfaktoren hängen nicht sehr stark von der Stückzahl der Geräte und der Zahl der eingebauten Funktionen ab. Ab einer Schaltungsgröße von 100 IC-Bausteinen und Gerätestückzahlen von etwa 500 pro Jahr erhält man praktisch konstante Kosten pro eingebauter IC-Funktion.

Kundenspezifizierte LSI-Schaltungen

Die bei diesem Konzept eingesetzten Bausteine sind Spezialanfertigungen für den einzelnen Anwendungsfall. Bei der Bearbeitung einer hochintegrierten Kundenschaltung sind die Beziehungen zwischen dem Halbleiter- und dem Gerätehersteller, der die integrierten Schaltungen in seinen Geräten und Anlagen einsetzt, wesentlich anders als bei Standardbauelementen. Bei diesen basiert der Liefervertrag auf dem Datenblatt oder auf vereinbarten Sonderspezifikationen. Dagegen erfolgt bei Kundenschaltungen zunächst eine Entwicklung speziell nach den Anforderungen des Kunden. Erst nach Abschluß der Entwicklungsphase kann mit Beginn der Serienlieferung das übliche Vertragsverhältnis zwischen Hersteller und Kunden einsetzen. Bei der großen Komplexität der Kundenschaltungen hat sich die frühere klare Abgrenzung zwischen den Aufgaben des Geräte- und denen des Halbleiterherstellers verwischt. Kundenschaltungen sind heute derart umfangreich, daß gute Systemerfahrung beim Halbleiterhersteller erforderlich ist, um eine optimalen Schaltungsentwurf zu liefern. Andererseits verlangen die grundsätzlichen Entscheidungen über den Einsatz von Kundenschaltungen, ihre Zuverlässigkeit und ihre Prüfung weitgehende Kenntnisse über Technologie und Fertigungsverfahren der Bauelemente auf der Seite des Geräteherstellers. Demgemäß kann auch die Aufgabenteilung zwischen beiden Partnern je nach ihren Fähigkeiten, ihrer Auslastung und in Abhängigkeit von anderen Einflüssen von Projekt zu Projekt verschieden sein.
Analysiert man die technischen Gegebenheiten der Kundenschaltungen, so kommt man zu folgenden Ergebnissen: Der Entwicklungsingenieur kann die Schaltung optimal an die Problemstellung anpassen, er hat aber nur die Wahl zwischen wenigen MOS-Technologien, die sich für die Hochintegration eignen. Diese Technologien arbeiten durchweg mit geringem Leistungsverbrauch, erlauben aber keine so hohen Funktionsgeschwindigkeiten wie die bipolaren Schaltungsfamilien. Der hohe Integrationsgrad und die Möglichkeit, die Schaltungen optimal auszulegen, führen zu sehr niedrigen Bauelementezahlen und daher geringem Raumbedarf. Der geringe Leistungsverbrauch erlaubt zudem sehr kompakte Bauweisen. Die Zahl der externen und internen Verbindungen ist durch die hohe Integration besonders gering, so daß eine hohe Zuverlässigkeit zu erwarten ist.
Die Änderbarkeit kundenspezifizierter LSI-Schaltungen ist sehr schlecht, jede Änderung der Schaltung bedingt eine Änderung der Herstellungsmasken und somit einen neuen langwierigen und kostspieligen Entwicklungszyklus. Die Entwicklungszeit für solche Schaltungen ist heute in der Regel noch sehr lange, da mehrere Entwicklungsschritte beim Geräte- und beim Halbleiterhersteller erforderlich sind, die koordiniert werden müssen. Die Ersatzteilbeschaffung kann bei diesen hochspezialisierten Bauelementen, die nur für einen Kunden gefertigt werden, Schwierigkeiten bereiten. Im allgemeinen wird man der Gefahr, daß der Hersteller die Produktion einstellt, durch einen vertraglich gesicherten Zweithersteller begegnen müssen.
Betrachtet man die Kosten bei dieser Technologie, so zeigt sich, daß die Kosten der Bauelemente sehr stark stückzahlabhängig sind. Diese Bauelementekosten setzen sich aus den Fertigungskosten der Halbleiterbauelemente und den Kosten für die Maskenentwicklung zusammen. Wenn man diese hohen Maskenkosten auf geringe Bauelementestückzahlen umlegen muß, so ergeben sich sehr hohe Bauelementekosten. Zudem verlangen die meisten Halbleiterhersteller garantierte Mindestabnahmen, so daß kundenspezifizierte Schaltungen normalerweise nur für Geräte, die in großen Stückzahlen gefertigt werden, einsetzbar sind. Bei sehr einfachen Schaltungen, d.h. bei Schaltungen mit unter 100 IC-Funktionen, lohnt sich im allgemeinen die Herstellung einer kundenspezifizierten Schaltung noch nicht. Bei ausreichender Schaltungskomplexität und großen Stückzahlen kommt man aber wegen der optimalen Auslegung der Schaltung, d.h. der geringstmöglichen Siliziumfläche in den Bauelementen, auf sehr niedrige

Bauelementepreise, geringe Implementierungskosten und somit sehr geringe Gesamtkosten pro eingebauter IC- oder Gatterfunktion.

Mikrorechner

Mikrorechner sind standardisierte LSI-Schaltungen, eine gewünschte Funktion wird durch entsprechende Programmierung erreicht, d.h. durch eine große Zahl aufeinander folgender Einzelschritte. Die mögliche Funktionsgeschwindigkeit ist deshalb wesentlich geringer als bei den beiden vorher besprochenen Konzepten.
Die Mikrorechner sind als standardisierte Schaltungen nicht optimal abgestimmt auf ein bestimmtes Problem. Der Schaltungsaufwand, gerechnet als die Zahl der Gatter in den Bausteinen, wird deshalb im allgemeinen höher sein als bei einer maßgeschneiderten Lösung. Daraus resultiert eine etwas höhere Leistungsaufnahme als bei entsprechenden kundenspezifizierten Schaltungen. Bei den bipolaren Bit-Slice-Mikrorechnern ist die Leistungsaufnahme hoch. Der Raumbedarf ist gering, bedingt durch den hohen Integrationsgrad dieser Schaltungen, die Zuverlässigkeit ist hoch. Die Änderbarkeit ist sehr gut. Eine Funktionsänderung erfordert im wesentlichen eine Programmänderung, die normalerweise keine besonderen Schwierigkeiten mit sich bringen wird. Der Hauptaufwand bei der Entwicklung liegt in der Programmierung. Die Entwicklungszeit wird im allgemeinen vergleichbar sein mit der Entwicklung entsprechender Schaltungen mit SSI- und MSI-Bausteinen. Für viele Mikrorechnertypen sind heute Zweithersteller vorhanden, so daß die Ersatzteilbeschaffung keine besonderen Probleme mehr bereitet. Bei sehr langlebigen Geräten, deren Lebensdauer die erwartete Marktlebensdauer der verwendeten Mikroprozessoren übertrifft, müssen besondere Vorkehrungen für geeignete Austauschmöglichkeiten getroffen werden.
Die Gesamtkosten pro IC-Funktion sind nicht sehr stark stückzahlabhängig (etwa vergleichbar mit den SSI- und MSI-Standardschaltungen). Die Kosten liegen schon bei einer geringen Zahl von IC-Funktionen recht günstig.

Programmable Logic Arrays (PLAs)

Ein PLA ist im Prinzip eine Matrix logischer AND-, NAND-, OR-, NOR-Gatter in einem integrierten Schaltkreis. Die Gatter können miteinander verbunden werden, um eine beliebige kombinatorische Logikfunktion zu erzeugen, d.h., bei einem bestimmten Eingangs-bit-Muster wird ein bestimmtes Ausgangs-bit-Muster erzeugt. Führt man Rückkopplungen von den Ausgängen über ein Register zu den Eingängen, so bekommt man Schaltwerke, deren Ausgangs-bit-Muster durch den inneren Zustand und die Eingangsvariablen bestimmt ist. Eine solche Konfiguration ist typisch geeignet als Ablaufsteuerwerk für die verschiedensten Vorgänge (In der Tat sind die Steuerwerke mancher Mikroprozessoren in dieser Form aufgebaut).
PLAs werden zur Zeit in 2 Arten angeboten, maskenprogrammierbar, also bei der Herstellung programmiert, oder elektrisch vom Anwender programmierbar.
Das Anwendungsgebiet der PLAs kann man etwa folgendermaßen umreißen: PLAs sind geeignet für kombinatorische Logik, wenn die Zahl der Ein- und Ausgänge nicht zu hoch ist, und für sequentielle Logik, wenn die Zahl der internen Zustände nicht zu hoch ist. In diesen Fällen ergeben sich kompakte Lösungen, die für hohe Funktionsgeschwindigkeiten geeignet sind. Weniger geeignet sind PLAs für sehr komplexe Vorgänge mit komplizierten Ablaufdiagrammen.

PLAs sind hochintegrierte Standardschaltungen wie Festwertspeicher und Mikroprozessoren. Das über die Ersatzteilversorgung, die Zuverlässigkeit usw. bei den Mikroprozessoren Gesagte gilt auch für PLAs.

Minicomputer und fertige Mikrocomputer

Bei diesem Konzept wird der Rechner als fertige Baugruppe eingekauft. Der Rechner wird im allgemeinen nicht optimal an die Anforderungen angepaßt sein, so daß Leistungsverbrauch, Raumbedarf, Zuverlässigkeit nicht die optimalen Werte erreichen. Probleme können auch die unterschiedlichen Bauweisen von Gerät und Rechner bieten. Die Funktionsgeschwindigkeit dagegen entspricht der der eigenentwickelten Rechner. Die wesentlichen Vorteile ergeben sich beim Entwicklungsaufwand und bei der Entwicklungszeit der Gerätehersteller. Diese werden einmal dadurch verringert, daß die Entwicklung des Rechners ganz entfällt. Dazu kommt, daß vor allem bei Minicomputern die Software-Unterstützung durch den Rechnerhersteller gut ist.

Bei guten Minicomputerherstellern stehen Pakete von Systemprogrammen zur Verfügung, die die Programmentwicklungszeit beim Gerätehersteller erheblich verkürzen können.
Diese Geräte sind also besonders bei sehr kleinen Stückzahlen geeignet, bei denen der höhere Einkaufspreis der fertigen Computer gegenüber den Entwicklungskosten keine Rolle spielt. Sehr häufig werden auch für Labormodelle und Prototypen fertige Mikrorechner auf einer Karte eingesetzt, die dann im endgültigen Gerät durch eine eigene Entwicklung ersetzt werden.

Die technischen Eigenschaften der verschiedenen Schaltungskonzeptionen sind in Tab. 10.1.1 zusammengefaßt. Daraus wurde Tab. 10.1.2 abgeleitet, die die wesentlichen Anwendungsgebiete den einzelnen Schaltungskonzeptionen auflistet. Jede Schaltungskonzeption hat ihre typischen Anwendungsbereiche, am ausgeglichensten sind die Merkmale bei den Mikroprozessoren verteilt, so daß ihnen ein besonders breites Anwendungsgebiet zukommt. Bestimmte extreme Anforderungen lassen sich nur in einer Technologie erfüllen, ist z.B. eine sehr hohe Geschwindigkeit gefordert, so kommen unabhängig von der Komplexität und der Stückzahl bis jetzt nur die schnellen SSI- und MSI-Bausteine in Frage. Wird eine besonders hohe Flexibilität gefordert, soll dasselbe Produkt z.B. auf den verschiedensten Exportmärkten eingesetzt werden, so scheiden kundenspezifizierte Schaltungen aus.

Merkmale	Schaltungen mit SSI- und MSI-ICs	kundenspezifizierte LSI-Schaltungen	standardisierte LSI-Schaltungen		fertige Mini- und Mikrorechner
			Mikrorechner	PLAs	
Funktionsgeschwindigkeit	je nach Technologie hoch bis sehr hoch	hoch	mittel	hoch	mittel
Leistungsaufnahme	niedrig bis hoch	sehr niedrig	niedrig	niedrig	je nach Modell niedrig bis hoch
Raumbedarf	groß	sehr klein	klein	klein	klein bis groß
Zuverlässigkeit	mittel	hoch	hoch	hoch	mittel bis hoch
Änderbarkeit	gut	sehr schlecht	sehr gut	gut	sehr gut
Entwicklungszeit	kurz	sehr lang	kurz	kurz	sehr kurz
Ersatzteilbeschaffung	sehr gut	schlecht	gut	gut	schlecht bis gut
Bemerkungen		Kosten sehr stark stückzahlabhängig		nicht so breit einsetzbar wie die anderen Konzeptionen	gute Software-Unterstützung bei Minicomputern, breite Palette von Peripheriegeräten verfügbar

Tab. 10.1.1
Merkmale von Einrichtungen mit unterschiedlichen Schaltungskonzeptionen

Weitere Informationen zu diesem Fragenkomplex, vor allem detaillierte Kostenangaben, wie man sie heute in der nachrichtentechnischen Industrie einsetzt, sind in einem Artikel von G. Zeidle und F. Ulrich enthalten, der in der Zeitschrift Elektrisches Nachrichtenwesen 51, Nr. 2 1976 erschienen ist und aus dem auch eine Reihe der hier besprochenen Punkte entnommen wurden.

Schaltungen mit SSI- und MSI-Bausteinen	kundenspezifizifizierte LSI-Schaltungen	standardisierte LSI-Schaltungen		fertige Mini- und Mikrorechner
		Mikrorechner	PLAs	
Einrichtungen mit: – niedriger Komplexität – hoher bis sehr hoher Funktionsgeschwindigkeit – kleinen Stückzahlen	Einrichtungen mit: – sehr hoher Stückzahl – höherer Komplexität und kleinem Volumen – hoher Funktionsgeschwindigkeit	Einrichtungen mit: – hoher Flexibilität – hoher Komplexität – mittleren Funktionsgeschwindigkeiten – größeren Stückzahlen	Einrichtungen mit: – einfachem Ablaufdiagramm – hoher Funktionsgeschwindigkeit	Einrichtungen mit: – sehr geringer Stückzahl – hoher Flexibilität – geringer Entwicklungszeit – mittlerer Funktionsgeschwindigkeit – hohen Anforderungen an die Peripheriegeräte – Prozeßsteuerungen – zentraler Meßdatenverarbeitung
Beispiele: – schnelle zentrale Steuerungen/Regelungen – Anpaßschaltungen	Beispiele: – Konsumgeräte – Endgeräte in der Nachrichtentechnik	Beispiele: – dezentrale und kleinere zentrale Steuerungen/Regelungen – Konsumgeräte – Meßgeräte – Datenverarbeitung	Beispiele: – Steuerungswerke aller Art – Codewandler	

Tab. 10.1.2
Anwendungsgebiete der verschiedenen Schaltungskonzeptionen

10.2 Auswahl eines geeigneten Mikroprozessors

Hat man sich bei der Entwicklung eines Gerätes für den Einsatz von Mikrorechnern entschlossen, so steht man nun vor der Frage, aus dem großen Angebot einen geeigneten Rechnertyp auszuwählen. Diese Auswahl wird von vielen Überlegungen beeinflußt: Ist der Mikroprozessor technisch geeignet, sind die Preisangebote und Lieferbedingungen befriedigend, gibt es schon Erfahrungen mit diesem Typ im Hause, sind geeignete Entwicklungssysteme lieferbar, sind diese im Hause verfügbar oder sind hier Investitionen erforderlich, bietet der Hersteller ausreichende Software-Unterstützung in Form von Systemprogrammen, Sammlungen von allgemein verwendbaren Hilfs-Routines usw., gibt es echte Zweithersteller, die die Versorgung sichern und andere Punkte mehr.
Am einfachsten sind die technischen Daten zu beurteilen: Leistungsverbrauch, Zahl und Belastbarkeit der Spannungsversorgung, Raumbedarf bei gegebener Programmgröße und gegebener Zahl der Anschlüsse nach außen, erforderlicher Temperaturbereich. In vielen Fällen wird bereits aufgrund dieser Daten eine Vorauswahl getroffen werden können. Wenn ein bestimmter Rechnertyp nicht mit dem erforderlichen Temperaturbereich angeboten wird oder er mehrere Versorgungsspannungen benötigt, die man sich im Gerät nicht verschaffen kann, so wird dieser Rechner ausscheiden, auch wenn er sonst noch so gut paßt.

Schwieriger zu beurteilen ist die Frage, ist der Rechner software-mäßig geeignet. Hier muß man zunächst entscheiden, soll das Programm in Assemblersprache geschrieben werden oder strebt man die Anwendung einer Hochsprache wie Fortran, PLA, Basic usw. an.
Assemblerprogrammierung ist zumeist zeitaufwendiger und erfordert gute Programmierer, erlaubt es aber, die volle Flexibilität und Leistung des Rechners in der gegebenen Anwendung auszuschöpfen. Eine besonders gute Dokumentation und Programmstruktierung ist erforderlich, um die Lesbarkeit und Änderbarkeit der Programme zu gewährleisten auch dann, wenn der Erfinder der Programme die Firma einmal verlassen haben sollte.
Hochsprachen kann man nur dann verwenden, wenn für den Rechner entsprechende Compiler angeboten werden, es sei denn, das Projekt ist so groß, daß sich die Entwicklung einer eigenen problemorientierten Hochsprache und des entsprechenden Compilers lohnt. Vorteile der Hochsprachen sind der geringere Programmieraufwand, die bessere Lesbarkeit und Änderbarkeit der Programme sowie die Tatsache, daß die Programmierer weniger Erfahrung benötigen. Nachteile der Hochsprachen sind im allgemeinen größere Programmlänge, längere Programmlaufzeiten, geringere Flexibilität auf speziellen Gebieten, wie z.B. der Ein- und Ausgabe usw. Die Nachteile der größeren Programmlänge und der längeren Laufzeiten im Vergleich zu Assemblerprogrammen können jedoch auch verschwinden, dann nämlich, wenn bei einem großen, komplizierten Programm die Assemblerprogrammierer die Übersicht über ihr Werk verlieren und deshalb keinen auch nur annähernd optimalen Code mehr produzieren. In diesem Fall kann das compilierte Programm auch durchaus kompakter sein als die Assemblerversion. Hier wirkt sich dann die größere Übersichtlichkeit und einfachere Strukturiermöglichkeit der problemorientierten Hochsprachen aus.
Nach dem heutigen Stand der Technik wird man deshalb kleinere Programme, d.h. Programme mit maximal einigen Tausend Instruktionen, häufig in Assemblersprache schreiben, größere Programme dagegen in einer problemorientierten Hochsprache.
Hat man sich für die Verwendung einer Assemblersprache entschieden, muß man den Instruktionssatz auf seine Eignung für das anstehende Problem prüfen. Zunächst wird man prüfen, ob der Instruktionssatz einfach struktiert und vollständig ist. Ist die Struktur nicht einfach und regelmäßig, gibt es viele Ausnahmen, unterschiedliche Befehlsformate, unverständliche Nebenwirkungen von Befehlen wie Löschen von Flags und Registern, die mit dem Befehl nichts zu tun haben, so ist dieser Instruktionssatz schwer zu verwenden und wird viele Programmfehler mit sich bringen, die dann mühsam herausgetestet werden müssen. Es gibt auch Instruktionssätze, die nicht vollständig sind, wo es fast unmöglich ist, bestimmte Operationen, wie z.B. ein Computed GOTO, auszuführen. Diese Fragen, die nun schon die Eignung eines Prozessors für ein bestimmtes Problem betreffen, behandelt man durch besondere Testprogramme, sog. Bench-Mark-Tests. Man sucht aus der vorliegenden Problemstellung wichtige und kritische Funktionen aus und programmiert diese Funktionen. Bei einer Steuerungsaufgabe wären das z.B. Abfragen, Verknüpfen und Setzen von Leistungen nach außen. Bei einer Codewandlung müssen vielleicht lange Listen sehr schnell durchsucht werden, in einem Meßgerät sind viele arithmetische Operationen in kurzer Zeit auszuführen. Man darf diese Funktionen nicht zu klein wählen, sonst sind sie nicht problemspezifisch genug; führt man die Bench-Mark-Tests jedoch sorgfältig aus, so erhält man dadurch eine Menge wertvoller Informationen. Man kann die Programmlänge abschätzen und verschiedene Prozessoren bezüglich ihres Speichbedarfes beurteilen. Man bekommt Werte für die Programmlaufzeit und kann so beurteilen, ob die Verarbeitungsgeschwindigkeit ausreichend ist. Die Programmlänge beeinflußt über den Speicherbedarf entscheidend die Systemkosten und außerdem den Programmaufwand. Man kann den Programmieraufwand in dieser Phase grob abschätzen, wenn man für die gesamte Aktivität der Programmplanung, dem Programmieren, Testen, Implementieren und Dokumentieren, im Durchschnitt etwa eine Stunde pro Instruktion für einen Mann ansetzt.
Wenn eine problemorientierte Hochsprache eingesetzt werden soll, hat man mit dem Instruktionssatz des Rechners wenig zu tun. Um Informationen über die zu erwartende Länge des Programms, den Speicherbedarf und die Programmlaufzeiten zu erhalten, wird man wieder Bench-Mark-Tests ausführen. Durch diese Tests wird man nun nicht nur den Rechner, sondern vor allem auch den Compiler beurteilen. Im Falle der problemorientierten Hochsprachen läßt die Länge des Quellenprogramms zwar Schlüsse über den Programmaufwand zu, nicht ohne weiteres aber über den Speicherbedarf. Dieser wird maßgeblich von der Effektivität des Compilers beeinflußt. Um sich davon ein Bild zu machen, kann man für eine bestimmte Funktion selbst ein Assemblerprogramm schreiben und dieses Programm dann mit dem vom Compiler produzierten Maschinencode vergleichen.

In vielen Fällen besteht die Möglichkeit, Assemblerprogramme und Programme in Hochsprachen zu kombinieren. Man hat dann die Möglichkeit, besonders kritische Programmteile, z.B. die innerste Schleife im Programm, deren Laufzeit vielleicht besonders kritisch ist, selbst zu optimieren und dadurch die Vorteile der Assemblerprogrammierung mit den Vorteilen der Programmierung in Hochsprachen zu kombinieren.
Zum Schluß dieses Abschnittes noch folgende allgemeine Bemerkungen: Wählt man für eine bestimmte Aufgabe einen Rechner, dessen Leistungsfähigkeit die Anforderungen erheblich übertrifft, so ist dieser Rechner nur zu einem geringen Teil seiner Kapazität ausgelastet. Man verschwendet also Geld. Wählt man den Rechner und die Speichergröße so, daß die Leistung gerade ausreicht, so kann man dadurch die Programmierkosten beträchtlich in die Höhe treiben.
Es kann enorm viel Zeit kosten, ein Programm so umzuschreiben, daß es ein paar Instruktionen kürzer wird und in einen gegebenen Speicher hineinpaßt.
Wäre der Speicherraum großzügiger bemessen, könnte man dieses Geld einsparen. Das gleiche gilt hinsichtlich der Laufzeit der Programme: Wenn die Bench-Mark-Tests ergeben, daß der Rechner gerade schnell genug ist, so muß man sehr genau überlegen oder besser auf einen etwas schnelleren Typ ausweichen. Irgendeine unvorhergesehene Schwierigkeit, und man muß plötzlich auf dem zeitkritischen Pfad einige Mikrosekunden einsparen, und das kann sehr schwer sein. Man wird also in aller Regel versuchen, bei der Auswahl des Rechners und der Speichergröße einige Sicherheit einzukalkulieren, und das um so mehr, je weniger Aufwand man in die Bench-Mark-Tests investiert, um so ungenauer also die Schätzungen für den Speicherbedarf und die Laufzeiten sind.
Mit Preisen und Lieferbedingungen als Auswahlkriterium für Mikrorechner wollen wir uns in diesem Lehrgang nicht befassen.
Ein anderer wichtiger Punkt, der bei der Auswahl zu bedenken ist, ist die Verfügbarkeit leistungsfähiger Entwicklungssysteme mit entsprechender Software-Unterstützung. Dieser Punkt wurde bereits angesprochen. Es soll aber hier nochmals betont werden, wie wichtig er für ein effektives praktisches Arbeiten ist. In vielen Fällen sind auch Programmbibliotheken, die der Rechnerhersteller evtl. zur Verfügung stellt, von großer Bedeutung, z.B. eine Sammlung von Statistikprogrammen oder Programmen zur Signalanalyse, aber auch einfachere Routines, wie spezielle Ein/Ausgaberoutines für Peripheriegeräte, Routines für Festkomma, Gleitkomma- und BCD-Arithmetik usw. Solche Programme aus Bibliotheken können gegebenenfalls den Programmaufwand beträchtlich reduzieren.
Sehr sorgfältig muß auch vor allem bei langlebigen Produkten die Frage der Ersatzteilversorgung betrachtet werden. Ist der Typ breit eingeführt und hat somit voraussichtlich eine lange Marktlebensdauer, oder handelt es sich um einen „exotischen Eintagstyp"? Gibt es Zweithersteller? Haben diese den Rechner nur kopiert, so daß vielleicht die absolute Kompatibilität fraglich ist, oder gibt es ein Abkommen über Maskenaustausch zwischen Erst- und Zweithersteller?
Die Entscheidung für einen bestimmten Rechner kann im Einzelfall recht schwierig sein. Wenn Sie einen solchen Katalog von Fragen und Einzelpunkten durchgehen, die Punkte entsprechend den Anforderungen Ihres Produktes bewerten, so sollte Ihnen aber eine fundierte Grundlage für Ihre Entscheidung zur Verfügung stehen.

10.3. Überlegungen zum Systementwurf

Es gibt eine ganze Reihe von Entwurfsparametern in einem rechnerorientierten System, die der Ingenieur an äußere Bedingungen anpassen kann. Wenn wir annehmen, daß die Minimalfunktion des Systems festgelegt ist, so können verschiedene Parameter gegeneinander abgewogen werden:
- Zeit zum Schreiben des Programms
- Anteil von Hardware- und Software-Funktionen
- Bedarf an Programm- und Datenspeicher
- Programmlaufzeit
- Leistungsfähigkeit der benutzten CPU
- Flexibilität und Änderbarkeit des Programms
- Lesbarkeit und Verständlichkeit des Programms
- Sicherheit, daß das Programm fehlerfrei ist

- Fähigkeit und Erfahrungen der Programmierer
- Hilfsmittel, die den Programmierern zur Verfügung stehen
- Zahl der Programmierer, die am Projekt arbeiten

Betrachten Sie z.B. ein Produkt, das in großen Stückzahlen verkauft werden soll. Hier müssen vor allem die Herstellkosten so niedrig wie möglich gehalten werden, d.h., das System muß auf minimale Rechnerkosten entwickelt werden mit hoher Sicherheit, daß keine Programmfehler beim Testen unentdeckt bleiben. Das erfordert erhöhten Entwicklungsaufwand. Ist der Zeitplan für ein Projekt besonders knapp, so sollte man einen Rechner wählen, der wesentlich leistungsfähiger ist als unbedingt erforderlich, die besten Programmierer einsetzen und ihnen vor allem gut ausgestattete Entwicklungssysteme (Display, schnelle Drucker zur Ausgabe der Programmlisten, Plattenspeicher) und viel Arbeitszeit an diesen Systemen zur Verfügung stellen.
Die oben aufgeführten Entwurfsparmeter sind selbstverständlich nicht linear abhängig voneinander, man kann nicht etwa die Programmierzeit halbieren, wenn man das Programm auf die doppelte Länge anwachsen läßt. Wie bereits erwähnt, kann es Tage dauern, ein Programm um einige Instruktionen zu kürzen, so daß es in einen gegebenen Speicherbereich paßt. Dies ist allerdings in der Praxis doch recht oft erforderlich, denn Speicher werden nicht wortweise eingebaut, sondern in Blöcken von hunderten oder gar tausenden von Wörtern. Ist nun ein Programm um einige Instruktionen zu lang, so muß ein zusätzlicher Speicherbaustein eingebaut werden, der zum größten Teil leer ist, oder das Programm muß eben gekürzt werden.

Die Lage der Grenze zwischen den Funktionen, die in Hardware aufgebaut werden, und denen, die in Software realisiert werden, muß beim Entwurf ebenfalls bedacht werden. Es gibt Funktionen, die in Hardware sehr einfach zu realisieren sind, aber viele Programmschritte benötigen, also viel Speicher und viel Zeit bei einer Realisierung in Software. Ein einfaches Beispiel dafür wäre z.B. die Umkehrung der Reihenfolge der bit in einem Rechnerwort. Diese Operation ist etwas mühsam zu programmieren und ist dann auch relativ langsam. Muß eine solche Operation sehr häufig ausgeführt werden und ist die Zeit kritisch, so kann es sich vielleicht lohnen, das Wort auszugeben und es wieder einzulesen, wobei die Leitungen zwischen Ausgabe- und Eingabe-Interface entsprechend überkreuzt werden. Dagegen sind Vorgänge mit komplexeren Flußdiagrammen mit viel Bedingungen und Verzweigungen mit Arithmetik im allgemeinen in Hardware schwierig zu realisieren, sie sollten deshalb per Programm ausgeführt werden.
Es ist schwierig, hier allgemein gültige Regeln anzugeben, die Anforderungen sind in der Praxis zu verschieden. Wichtig ist, daß man sich im konkreten Fall diese Punkte überlegt und sie optimiert. Dadurch sind in vielen Fällen erhebliche Kosteneinsparungen und einfachere technische Lösungen möglich.
Wenn man die Programmierungsarbeiten an einem System beschleunigen will, so ist die naheliegende Lösung, die Zahl der Leute zu vergrößern. Dabei ist zunächst zu bedenken, daß vor allem die Assemblerprogrammierung eine beträchtliche Einarbeitungszeit von einigen Monaten erfordert, in der keine vernünftigen Programme produziert werden. Zum anderen erhöhen sich die Reibungsverluste durch Kommunikationsschwierigkeiten usw. sehr stark mit der Zahl der beteiligten Programmierer. Die Aussage „2 Programmierer sind durchaus in der Lage, in 2 Jahren ein Programm zu schreiben, das ein Programmierer in einem Jahr schreibt", ist sicher übertrieben, enthält aber einen wahren Kern. Für Großprojekte wurden deshalb spezielle Organisationsformen entwickelt, um diese Schwierigkeiten zu vermeiden. Bei kleineren Projekten ist es häufig möglich, die Entwicklungszeit dadurch zu verkürzen, daß man besonders gute Hilfsmittel, gut ausgestattete Rechner und viel Rechnerzeit zur Verfügung stellt und die Mitarbeiter von anderen Arbeiten so weit wie möglich entlastet. Man kann dann die Zahl der Programmierer klein halten, die Kommunikations- und Anpaßprobleme bleiben gering.
Diese und weitere problemspezifischen Punkte müssen bei der Planung eines Systems, in dem Mikrorechner eingesetzt werden sollen, möglichst frühzeitig bedacht werden, um so sicherzustellen, daß das Projekt mit größter Wahrscheinlichkeit erfolgreich durchgeführt werden kann.

Auch die beste Planung kann unvorhergesehene Probleme nicht ausschalten, sie kann aber deren Zahl reduzieren und Reserven schaffen, um diese Probleme zu lösen.

Schlußwort

Nach diesen allgemeinen Betrachtungen im Abschnitt 10. sind wir am Ende des Lehrganges angelangt. Sicherlich war es für Sie nicht immer einfach, den komplexen und teilweise schwierigen Lehrstoff sofort zu begreifen und auch gedanklich zu verarbeiten. Wir sind jedoch davon überzeugt, daß sie jetzt über ein fundiertes Basiswissen verfügen, das Ihnen erlaubt, mit Mikrorechnern zu arbeiten. Wir sind uns darüber im klaren, daß wir nicht alle speziellen Punkte, die mit Mikroprozessoren zusammenhängen, ansprechen konnten. Wie Sie das erworbene Wissen weiter ausbauen und in der Praxis anwenden, liegt nun bei Ihnen.

Viel Erfolg in Ihrem weiteren Berufsleben wünschen Ihnen die Autoren und die ITT Fachlehrgänge.

Anhang

Antworten auf die Fragen zu Abschnitt 7.

1. Als Minimalkonfiguration werden benötigt:
- Ein Mikrorechner mit Anschlußmöglichkeiten für Festwertspeicher
- Ein Fernschreiber oder ein äquivalentes Gerät
- Schreib/Lesespeicher
- Ein Bedienungspult, das Zugriff zum Bussystem des Prozessors ermöglicht
- Ein Programmiergerät für RePROMs

2. Es handelt sich um ein Hilfsprogramm, mit dessen Hilfe Textverarbeitung möglich ist. So kann z.B. ein auf dem Papier entwickeltes Programm mit Hilfe eines Fernschreibers unter Verwendung eines bestimmten Codes in den Speicher geladen (normalerweise ASCII) und auch mit sämtlichen Kommentaren ausgedruckt werden.

3. ES GIBT MIKROPROZESSOREN MIT 4, 8 UND 16 BIT WORTLAENGE!

4. Die Grundfunktion eines Assemblerprogramms ist, ein in mnemonischer Schreibweise erstelltes Programm in den Maschinencode zu übersetzen.

5. Durch Sonderzeichen und besondere Befehle wird der Übersetzungsvorgang gesteuert. So besagt zum Beispiel ein Semikolon, daß der eigentliche Befehl abgeschlossen ist und in der gleichen Zeile nachfolgender Text den Kommentar darstellt, der nicht übersetzt werden darf.

6. In dieser Tafel bzw. Liste sind alle mnemonischen Ausdrücke mit der Übersetzung in den Maschinencode enthalten. Damit erhält der Assembler die Möglichkeit, eine Übersetzung durchzuführen.

7. Beim ersten Durchlauf wird das edierte Programm eingelesen und dabei gleichzeitig die benötigte Anzahl der Speicherplätze festgelegt. Die vom Assembler erkannten symbolischen Adressen werden in eine Symboltafel eingetragen. Dann wird ihnen eine absolute Adresse zugeordnet. Im zweiten Durchlauf erfolgt dann die eigentliche Übersetzung.

8. Die symbolische Adresse MAP1 hat den Wert 3 4 und den Inhalt 4 6.

9. Mit dem Ladeprogramm können Programme von externen Speichern in den Schreib/Lesespeicher des Rechners geladen werden.

10. Debugging-Routines (Entwanzungs-Routinen) werden zur Fehlersuche und Fehlerbeseitigung bei der Programmerstellung eingesetzt.

11. Erst durch ein Monitorprogramm besteht die Möglichkeit, mit einem MP-System zu korrespondieren. Ohne Monitorprogramm sind keine Eingriffe in das MP-System möglich.

12. Mit einem Compiler können Befehle einer höheren Programmsprache in den Maschinencode umgesetzt werden.

Antworten auf die Fragen zu Abschnitt 8.

1. Das Bussystem verbindet die Baugruppen eines Rechners miteinander. Es handelt sich also um Datenleitungen, an die mehrere Baugruppen gleichzeitig angeschlossen sind.

2. Die an den Bus angeschalteten Geräte müssen über Tri-State-Ein- bzw. -Ausgänge verfügen. Die Geräte, die im Moment für einen Datentransfer nicht benötigt werden, müssen in den Hochimpedanzzustand geschaltet werden.

3. Durch ein entsprechendes Signal am Chip-Enable-Eingang kann ein Speicherbaustein ein- oder ausgeschaltet werden.

4. Die Zugriffszeit der Speicherbauelemente muß kompatibel mit der vom Systemtakt abhängigen Zeitverzögerung sein.

5. Bei dieser Transferart werden den einzelnen Peripheriegeräten Adressen zugeordnet, die über das Programm angesprochen werden können.

6. Beim Polling-Verfahren fragt der Rechner die einzelnen Ein- und Ausgabegeräte ab, ob Daten transferiert werden müssen. Zu welchem Zeitpunkt diese Abfragen erfolgen, wird durch das Programm bestimmt.

7. Dieses Signal zeigt dem Prozessor an, daß ein Peripheriegerät Daten abgeben bzw. neue Daten aufnehmen möchte.

8. Der Hauptvorteil ist die Bedienungsgeschwindigkeit der Peripheriegeräte, da, wenn nicht ausdrücklich verboten, an jeder Stelle im Programm ein Interrupt erfolgen kann.

9. DMA = **D**irect-**M**emory-**A**ccess = direkter Speicherzugriff besagt, daß periphere Geräte einen direkten Zugriff zum Speicher haben (ohne Umweg über die CPU). Dadurch lassen sich auch größere Datenmengen schnell ein- und auslesen (große Übertragungsrate).

Antworten auf die Fragen zu Abschnitt 9.

1. Bei einem 1-Chip-Mikrorechner sind alle Funktionen eines Rechners (CPU, Speicher, Ein/Ausgabestufen, Taktgenerator usw.) auf einem Kristallplättchen (Chip) integriert.

2. Ein MP 8080 stellt die CPU eines Mikrorechners dar.

3. Die wichtigsten Kenngrößen sind die Zugriffszeit und die Speicherorganisation.

4. Es muß eine periodische Auffrischung (refresh) im Abstand von einigen Millisekunden erfolgen, da ansonsten der Speicherinhalt verlorengeht.

5. Die Umschaltung erfolgt grundsätzlich durch Anlegen von H- oder L-Pegel am Mode-Eingang des 8212-Bausteines.

6. Mit UART werden Bausteine zur seriellen Abwicklung des Datentransfers zwischen Mikroprozessor und Peripheriegeräten bezeichnet (UART = **U**niversal Synchronous-**A**synchronous **R**eceiver/**T**ransmitter).

7. Der Interruptvektor stellt die Startadresse einer Interrupt-Routine eines anfordernden Peripheriegerätes dar.

8. Unter Modem = **Mo**dulator + **Dem**odulator versteht man einen Wandler für Datenübertragung. Ein Modem übernimmt digitale Informationen und wandelt sie in modulierte Wechselstromsignale um. Auf der Empfangsseite werden die emfangenen modulierten NF-Signale in Gleichstromschritte umgewandelt.