Mikroprozessoren und Mikrorechner

Lehrheft 1

ITT Fachlehrgänge, Pforzheim

ITT

2. Auflage, April 1977

Druck: Druckerei Seiter, 7535 Königsbach-Stein

Einleitung

Für zahlreiche Bereiche der Technik wird die Einführung der Mikroprozessoren von enormer Bedeutung sein. Viele Experten prophezeien, daß der Einsatz von Mikroprozessoren technische Umwälzungen mit sich bringen wird, die in ihrem Ausmaß mit der Einführung des Transistors vergleichbar sind. Das mag hochgegriffen erscheinen, aber es ist ganz sicher, daß die Mikroprozessoren, die für wenig Geld und auf kleinem Raum die volle Leistungsfähigkeit und Flexibilität eines Computers zur Verfügung stellen, in zahllosen Geräten eingesetzt und darüber hinaus viele neue Produkte ermöglichen werden. Es dauert sicher nicht mehr lange, bis uns Mikroprozessoren in allen Bereichen unseres täglichen Lebens begegnen. Angefangen bei Haushaltsgeräten wie Herde, Waschmaschinen usw. über Heizungsregelungen, Meßgeräte im Labor, Werkzeugmaschinensteuerungen bis hin zur Automobiltechnik bieten sich Einsatzmöglichkeiten für den Mikroprozessor. Dies bedeutet aber, daß Techniker und Ingenieure aus praktisch allen Sparten der Technik mit diesem Bauelement konfrontiert werden. Die Betroffenen stehen plötzlich vor neuen Problemstellungen, für die sie zum großen Teil nicht ausgebildet worden sind und auch gar nicht ausgebildet werden konnten.

Kennzeichnend für diesen Trend ist weiterhin, daß in Zukunft ein immer größer werdender Anteil der Entwicklung eines Gerätes auf die sog. Softwareentwicklung fällt, d.h. die Entwicklung von Computerprogrammen zur Steuerung des Gerätes. So macht z. B. bei der Steuerung einer Waschmaschine mit Hilfe eines Mikroprozessors die Entwicklung des Programmes für diese spezielle Aufgabe einen wesentlichen Teil der Entwicklungsarbeit aus.

Dieser Lehrgang soll den Technikern und Ingenieuren eine fundierte Einführung in diese neue Technik geben. Um den Lernenden Schritt für Schritt an die neue Technik heranzuführen, wird zunächst die grundsätzliche Funktion sowie der prinzipielle Aufbau eines Computers oder Rechners besprochen. Hierzu sind einige Grundlagen der Digitaltechnik erforderlich, die in den Abschnitten 1. und 2. in knapper Form noch einmal erläutert werden.
Nach diesen grundlegenden Abschnitten wird dann in mehreren Stufen die Struktur eines einfachen funktionsfähigen Mikrocomputers zusammengestellt. Angefangen mit den einfachen Bausteinen der Digitaltechnik wird in mehreren Stufen ein komplettes Rechenwerk, also das Kernstück eines Mikrocomputers, entwickelt. Durch Hinzufügen von Speichern und eines Steuerwerkes entsteht schließlich ein einfacher aber funktionsfähiger Mikrocomputer. Zu jedem dieser Entwicklungsschritte können mit dem Experimentiersystem zahlreiche Experimente durchgeführt werden, so daß jeder Entwicklungsschritt nachvollzogen werden kann.
Nachdem der Teilnehmer mit dem Grundkonzept vertraut ist, wird der Mikrocomputer vom Standpunkt der Software, also der Programmierung, her behandelt. Die Eigenschaften eines Mikroprozessors sind festgelegt in Form eines sog. Instruktionssatzes, also der Summe der Befehle und Adressiermöglichkeiten, die ein Rechner bietet. Es wird nicht mehr erklärt, durch welche schaltungstechnische Maßnahmen jeder einzelne Befehl realisiert wird. Wichtig für den Anwender von Mikroprozessoren sind nicht die Kenntnisse der schaltungstechnischen Details, sondern vielmehr die Kenntnisse, die es ihm ermöglichen, anhand der vom Hersteller mitgelieferten Software Programme für seine Aufgaben zu entwickeln. Es werden also hier die Instruktionen oder Befehle sowie die Adressierarten, die in Mikrorechnern zur Verfügung stehen, behandelt. Über Programmbeispiele werden die Grundlagen der Programmierung und die Verwendung der Rechnerbefehle erläutert.
Für diese grundsätzlichen Betrachtungen und Programmübungen wird kein bestimmter Rechnertyp eines bestimmten Herstellers angesprochen. Vielmehr wollen wir Ihnen einen Überblick geben über die Befehle und Adressiermöglichkeiten, wie man sie heute in Mikroprozessoren findet.
Damit sind Sie befähigt, sich in jeden Rechnertyp einzuarbeiten und dann den für Ihre Problemstellung geeigneten Typ auszusuchen.
Aus diesem Grunde haben wir einen hypothetischen Mikrorechner entwickelt, der sich durch einen möglichst übersichtlichen Befehlssatz auszeichnet. Es werden hier alle die Tricks und Ausnahmen vermieden, die die Hersteller praktischer Mikroprozessoren anwenden, um an dieser und jener „Ecke" noch ein wenig mehr Leistung aus dem Produkt herauszuholen. Das Hauptaugenmerk liegt hier also auf einem klar strukturierten, leicht überschaubaren Instruktionssatz, in den Sie sich leicht einarbeiten können und an dem die wesentlichen Punkte klar herausgearbeitet werden können. Dieser zweite Hauptteil des Lehrganges wird ergänzt durch

Abschnitte über allgemeine Fragen der Programmierung und über die für das praktische Arbeiten unerläßlichen Hilfsprogramme.
Im dritten Hauptteil wird dann ein echter Mikrorechner, und zwar der weitverbreitete Typ 8080, besprochen. Dieser 8080 ist in dem Experimentiergerät enthalten, so daß Sie wieder selbst Programme schreiben und testen können, um sich somit in diesen populären Rechnertyp einzuarbeiten. Zu Ihrer Unterstützung ist in dem Experimentiergerät ein einfaches sog. Monitorprogramm enthalten, das Ihnen einen optischen Zugriff in die Register und Speicher des 8080 gestattet. Dieser Teil wird ergänzt durch einige ausgewählte Kapitel über Ein- und Ausgabemethoden, verfügbare Bauelemente usw.
Zum Schluß noch ein Wort über das Experimentiergerät selbst. Hierin ist ein vollständiges 8080-Mikrorechnersystem mit einem 8080-Mikroprozessor, Speicher, Ein- und Ausgabeschaltungen enthalten. Dieser Rechner wird dazu verwendet, den Addierer-Subtrahierer, den Akkumulator usw. und schließlich den vereinfachten Rechner und den hypothetischen Mikrorechner zu simulieren. Das Experimentiergerät ist also eine praktische Anwendung eines Mikrorechners. In diesem Zusammenhang können Sie sich vielleicht vorstellen, welcher Aufwand erforderlich wäre, wenn all die Funktionen dieses Experimentiersystems mit „normalen" Digitalschaltungen aufgebaut wären. Hier wird alles mit einem entsprechend programmierten Mikrorechner durchgeführt.

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

Dezimal	3-Exzeß-Code
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

Tab. 2.7.2
3-Exzeß-Code

Die Verarbeitung von BCD-Zahlen bedeutet, daß zwar Dezimalzahlen eingegeben werden, intern im Rechner werden jedoch Binärzahlen verarbeitet.
Als Beispiel für die Arithmetik mit BCD-Zahlen soll hier die Addition im 8421-Code behandelt werden, da eine Reihe von Mikroprozessoren spezielle Instruktionen dafür hat.
Die Arbeitsweise soll anhand einiger Beispiele demonstriert werden:

1.

5	0 1 0 1
+ 4	0 1 0 0
9	1 0 0 1

In diesem Beispiel wurden 2 BCD-Zahlen nach den Regeln der binären Addition addiert, und es ergab sich ein richtiges Ergebnis.

2.

5	0 1 0 1
+ 7	0 1 1 1
12	1 1 0 0

Bei diesem Beispiel wurden ebenfalls die Regeln der binären Addition angewandt, dabei ergab sich aber im Ergebnis ein illegaler Code, nämlich 1100.
Dieses Ergebnis kann dadurch korrigiert werden, daß man zu dem Ergebnis der binären Addition die Zahl 6, also 0 1 1 0, addiert, d.h., man „überspringt" bei der Addition die 6 illegalen Codes von 1 0 1 0 bis 1 1 1 1. Die Korrektur

	1 1 0 0
	+ 0 1 1 0
0 0 0 1	0 0 1 0

ergibt das richtige Ergebnis 12 im 8421-Code.

3. Ein weiterer Fall muß noch betrachtet werden:

8	1 0 0 0
+ 9	+ 1 0 0 1
17	0 0 0 1 0 0 0 1

Die binäre Addition ergibt in diesem Fall das Ergebnis 11 im 8421-Code.
Das ist ein legaler BCD-Code, aber ein falsches Ergebnis. Dieser Fehler entsteht dadurch, daß bei der Addition die 6 illegalen Codes mitgezählt wurden, die Korrektur führt man also ebenso wie im vorigen Fall durch Addition von 0 1 1 0 aus.

0 0 0 1	0 0 0 1
+ 0 0 0 0	0 1 1 0
0 0 0 1	0 1 1 1

Man erhält also 17 in BCD-Darstellung. Dieser Fall ist daran erkennbar, daß das Ergebnis eine legale BCD-Zahl war, daß aber ein Übertrag in die nächsthöhere BCD-Stelle erfolgte.

Eine Addition im 8421-Code erfolgt somit in mehreren Schritten. Dies ist in einem einfachen Flußdiagramm (Bild 2.7.1) grafisch dargestellt.

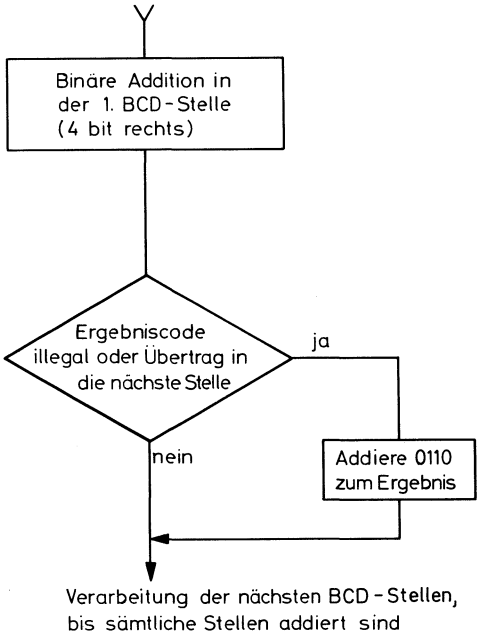


Bild 2.7.1
Flußdiagramm zur Erläuterung einer Addition im 8421-Code

Bei der Subtraktion müssen die illegalen Codes auf ähnliche Weise berücksichtigt werden, ebenso bei Multiplikation und Division, die ja eine wiederholte Anwendung der Addition bzw. Subtraktion sind.
Zusammenfassend kann man also über BCD-Codes folgendes sagen:
Die Tatsache, daß illegale Codewörter vorkommen, bedeutet, daß für einen gegebenen Zahlenvorrat in BCD-Codierung eine größere Wortlänge erforderlich ist, als in der binären Darstellung. Eine größere Wortlänge bedeutet größeren Speicherbedarf und mehr Schaltungsaufwand.
Die arithmetischen Operationen in BCD-Codes sind generell komplizierter und somit in der Praxis auch langsamer und aufwendiger als im Binärcode. Dagegen stehen die Vorteile der dezimalen Arbeitsweise, das Wegfallen der Binär-Dezimalumwandlungen bei Ein- und Ausgabe, so daß in jedem Einzelfall geprüft werden muß, ob BCD-Codes zweckmäßig sind.

Fragen zu Abschnitt 2.7

1. Addieren Sie folgende BCD-Zahlen

a)
```
  0 0 0 1   0 1 0 1   1 0 0 1
+ 0 0 1 1   0 1 1 1   0 0 1 0
```

b)
```
  0 0 0 1   1 0 0 0   0 1 1 1
+ 1 0 0 1   1 0 0 1   0 1 1 0
```

3. Arbeitsweise eines Rechners bzw. Computers

Grundsätzlich läßt sich die Arbeitsweise eines Rechners dadurch beschreiben, daß er in der Lage ist, Daten so zu verarbeiten, wie es durch ein Programm vorgeschrieben ist. Durch verschiedene Programme kann ein Rechner zur Lösung verschiedener Aufgaben eingesetzt werden. Bis vor einigen Jahren waren Rechner nur als sehr voluminöse und teure Einrichtungen zu haben. Dadurch war ihr Einsatz auf komplexe Aufgaben begrenzt, wie z.B. die Verarbeitung von Massendaten im kommerziellen Bereich. Erst die rapide Weiterentwicklung der Halbleitertechnologie ermöglichte es, auch kleinere und nicht so teure Rechner herzustellen,

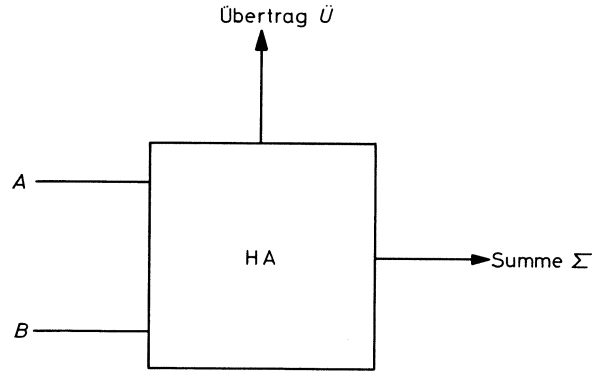
die allerdings auch weniger leistungsfähig und weniger komfortabel vom ganzen Bedienungsablauf her gesehen waren. Diese kleineren Computer wurden und werden allgemein als **Minicomputer** bezeichnet.
Der nächste Schritt in dieser Entwicklungsrichtung war der, mittels der sog. **L**arge **S**cale **I**ntegration (LSI), was mit hohem Integrationsgrad übersetzt werden kann, durch mehrere tausend Bauelemente die **Zentraleinheit CPU** (CPU = **C**entral **P**rocessing **U**nit) eines Mikrocomputers auf einem Chip herzustellen. Eine solche Zentraleinheit eines Mikrocomputers wird als **Mikroprozessor** bezeichnet. Interessant in diesem Zusammenhang dürfte für Sie sein, daß der in unserem Experimentiersystem verwendete Mikroprozessor vom Typ 8080 auf einem Chip mit 23 mm^2 (rund 4,8 mm x 4,8 mm) mehr als 4 500 MOS-Transistoren enthält. Mit diesen über 4 500 Transistoren ist es nun möglich, ein komplettes Steuer- und Rechenwerk für einen Mikrocomputer zu realisieren. Im Rechenwerk werden dabei die arithmetischen und logischen Operationen ausgeführt, während das Steuerwerk den internen Ablauf im Rechner steuert. Damit aus einem Mikroprozessor ein Mikrocomputer wird, sind weitere zusätzliche Einrichtungen wie Programmspeicher, Datenspeicher, Ein- und Ausgabebausteine, zusätzliche Logikschaltungen usw. erforderlich. Wie umfangreich diese zusätzlichen Einrichtungen werden, hängt dabei vom jeweiligen Anwendungsfall ab. Wichtig ist jedoch die Erkenntnis, daß man in der Praxis mit einem Mikroprozessorbaustein alleine noch keine Aufgabenstellungen lösen kann, dazu ist immer ein Mikrorechner erforderlich. In den nachfolgenden Abschnitten soll die prinzipielle Arbeitsweise eines Mikrorechners erläutert werden. Da sich prinzipiell die Arbeitsweise eines Mikrorechners von einem anderen Rechner nicht unterscheidet, werden Schritt für Schritt die einzelnen Baustufen behandelt, die für einen Rechner allgemein notwendig sind. Jeder Schritt wird dabei durch entsprechende Experimente verdeutlicht.

3.1 Addierwerk

Aus Abschnitt 2. ist bekannt, wie man Binärzahlen addiert. Wenn im einfachsten Falle 2 1-bit-Binärzahlen addiert werden sollen, so gibt es grundsätzlich folgende Möglichkeiten:

$A + B$	Summe	Übertrag
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

Eine Schaltung, die in der Lage ist, diese Rechenoperationen durchzuführen, wird als Halbaddierer (HA) bezeichnet. In Bild 3.1.1 ist ein HA in Blockschaltbildform dargestellt.



a)

A	B	Σ	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

b)

Bild 3.1.1
Halbaddierer
a) Blockschaltbild
b) Funktionstabelle

Aus der Funktionstabelle können die Funktionsgleichungen abgeleitet werden, die für die Summen- und Übertragsbildung erfüllt sein müssen:

Summe $\Sigma = (A \wedge \overline{B}) \vee (\overline{A} \wedge B) = A \veebar B$ (EXCLUSIV-ODER)
Übertrag $\ddot{U} = A \wedge B$ (UND)

Aus diesen Gleichungen kann jetzt die logische Schaltung eines Halbaddierers abgeleitet werden (Bild 3.1.2).

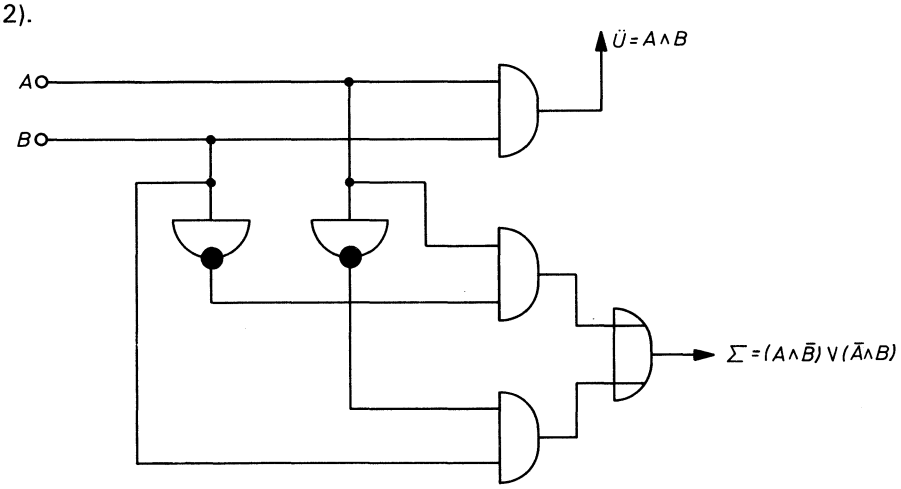
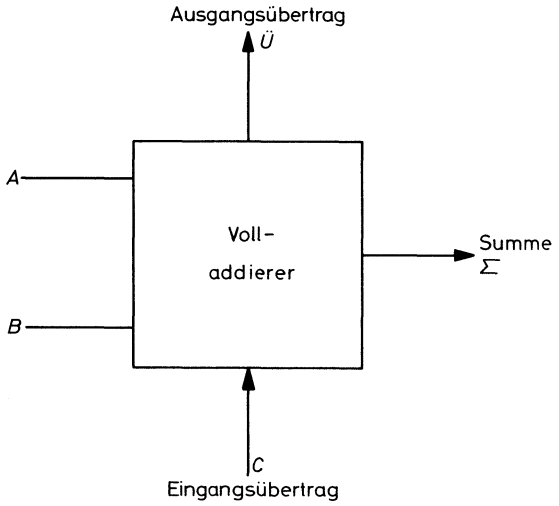


Bild 3.1.2
Schaltung eines Halbaddierers

Wenn nun mehrstellige bit-Kombinationen (sog. Wörter) addiert werden sollen, so reicht ein Halbaddierer hierfür nicht aus. In diesem Fall muß nämlich bei einem Übertrag dieser in der nächsthöherwertigen Stelle berücksichtigt werden. Dies kann nur mit einem Volladdierer gelöst werden, der außer den Eingängen A und B noch einen Übertragseingang C hat (Bild 3.1.3).



a)

A	B	C	Σ	Ü
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

b)

Bild 3.1.3
Volladdierer
a) Blockschaltbild
b) Funktionstabelle

Für die Summe gilt jetzt die Funktionsgleichung:

$$\Sigma = (\overline{A} \wedge B \wedge \overline{C}) \vee (A \wedge \overline{B} \vee \overline{C}) \vee (\overline{A} \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C)$$

Diese Gleichung läßt sich nach den Regeln der Schaltalgebra umformen in:

$$\Sigma = (A \veebar B) \veebar C = A \veebar B \veebar C$$

Diese Gleichung entspricht der Kaskadierung von 2 EXCLUSIV-ODER-Verknüpfungen, wie schon in Abschnitt 1.3, Bild 1.3.5 gezeigt.
Für den Ausgangsübertrag gilt die Beziehung:

$$\ddot{U} = (A \wedge B \wedge \overline{C}) \vee (\overline{A} \wedge B \wedge C) \vee (A \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C)$$

Diese Gleichung läßt sich vereinfachen zu:

$$\ddot{U} = (A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$$

Damit ergibt sich für den Volladierer eine Schaltung entsprechend Bild 3.1.4.

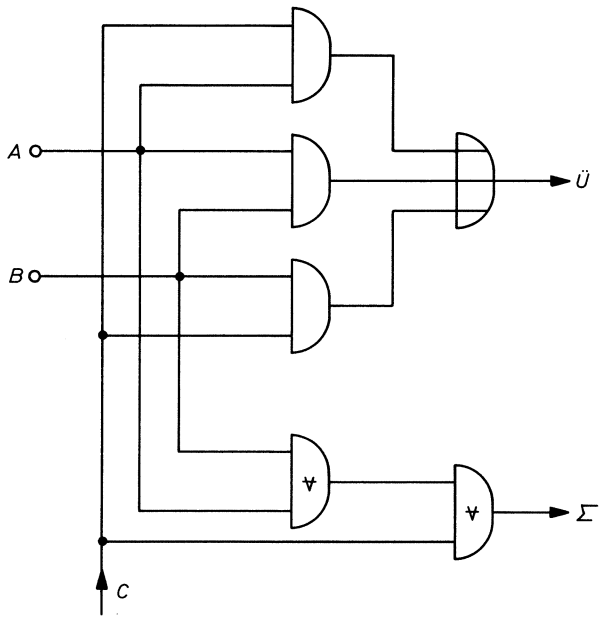


Bild 3.1.4
Schaltung eines Volladdierers

Sind nun 2 *n*-bit-Wörter zu addieren, müssen mehrere Volladdierer zusammengeschaltet werden. Hierzu ein einfaches Beispiel mit 2 3-bit-Wörtern:

	2^2	2^1	2^0
A:	1	0	1
B:	1	1	1
Ü:	1	1	
1	1	0	0

Genau genommen werden für dieses Beispiel ein HA und 2 VA benötigt, da in der Stelle 2^0 noch kein Übertrag vorhanden sein kann. In der Praxis wird man jedoch auch für diese Stelle einen VA benutzen, und den Eingang C_0 für ein solches Beispiel mit 0 belegen. Damit ergibt sich für die Lösung dieser Aufgabe eine Schaltung entsprechend Bild 3.1.5.

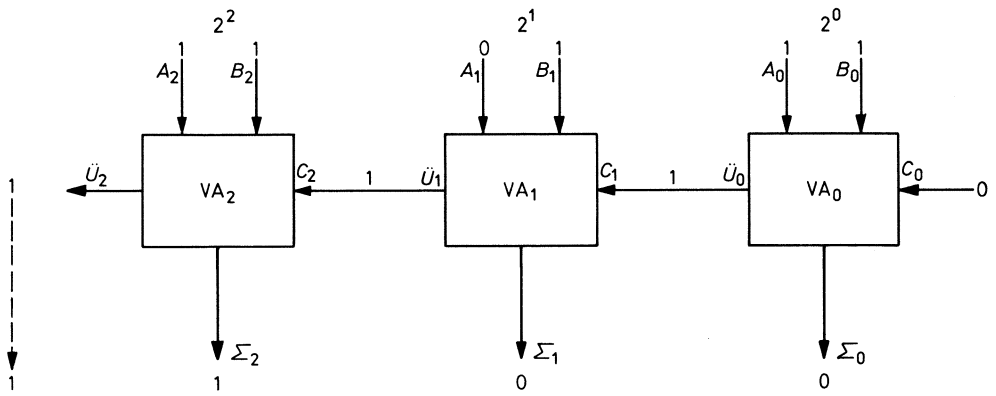


Bild 3.1.5
Addierwerk für 2 3-bit-Binärzahlen

Würde man bei diesem Addierwerk den Eingang C_0 statt mit 0 mit 1 beschalten, so würde das Ergebnis um 1 erhöht. Dies ist in der Praxis für bestimmte Anwendungsfälle erforderlich. Aus diesem Grunde erhält dieser Übertragseingang die Bezeichnung **Inkrementiereingang** INC (von Increment = Zuwachs). Ein Addierer mit einem Inkrementiereingang wird dann als **Ripple-Carry-Addierer** bezeichnet. In Bild 3.1.6 ist ein *n*-bit-Ripple-Carry-Addierer dargestellt.

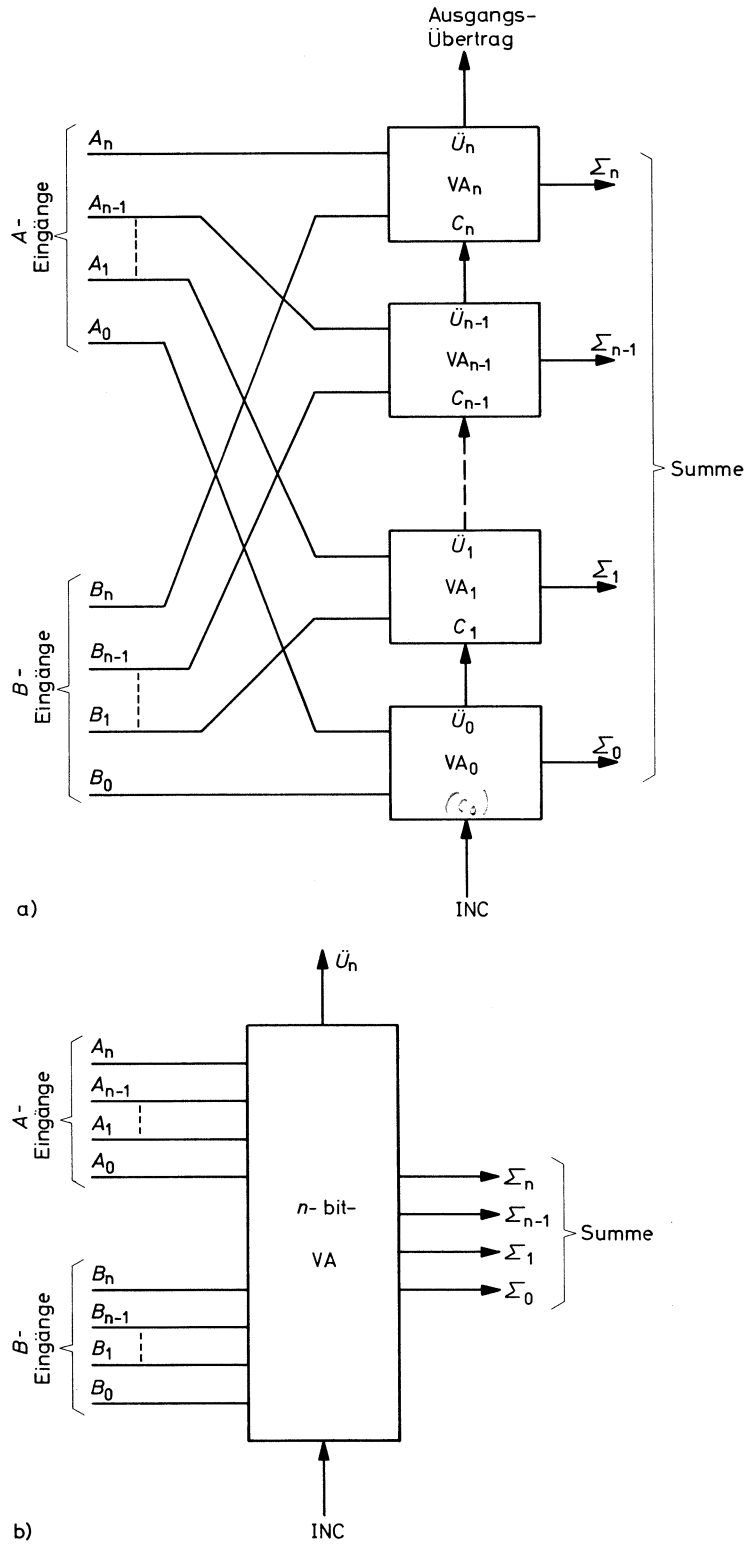


Bild 3.1.6
Ripple-Carry-Addierer
a) Funktionsschaltbild
b) Blockschaltbild

3.2 Addier-Subtrahierwerk

Ein Addierwerk nach Bild 3.1.6 läßt sich durch Vorschalten von Gattern so erweitern, daß viele weitere Funktionen damit verwirklicht werden können. Die in Bild 3.2.1 dargestellte Schaltung ermöglicht u. a. auch die Subtraktion von Binärzahlen und wird deshalb auch als Addier-Subtrahierwerk bezeichnet.

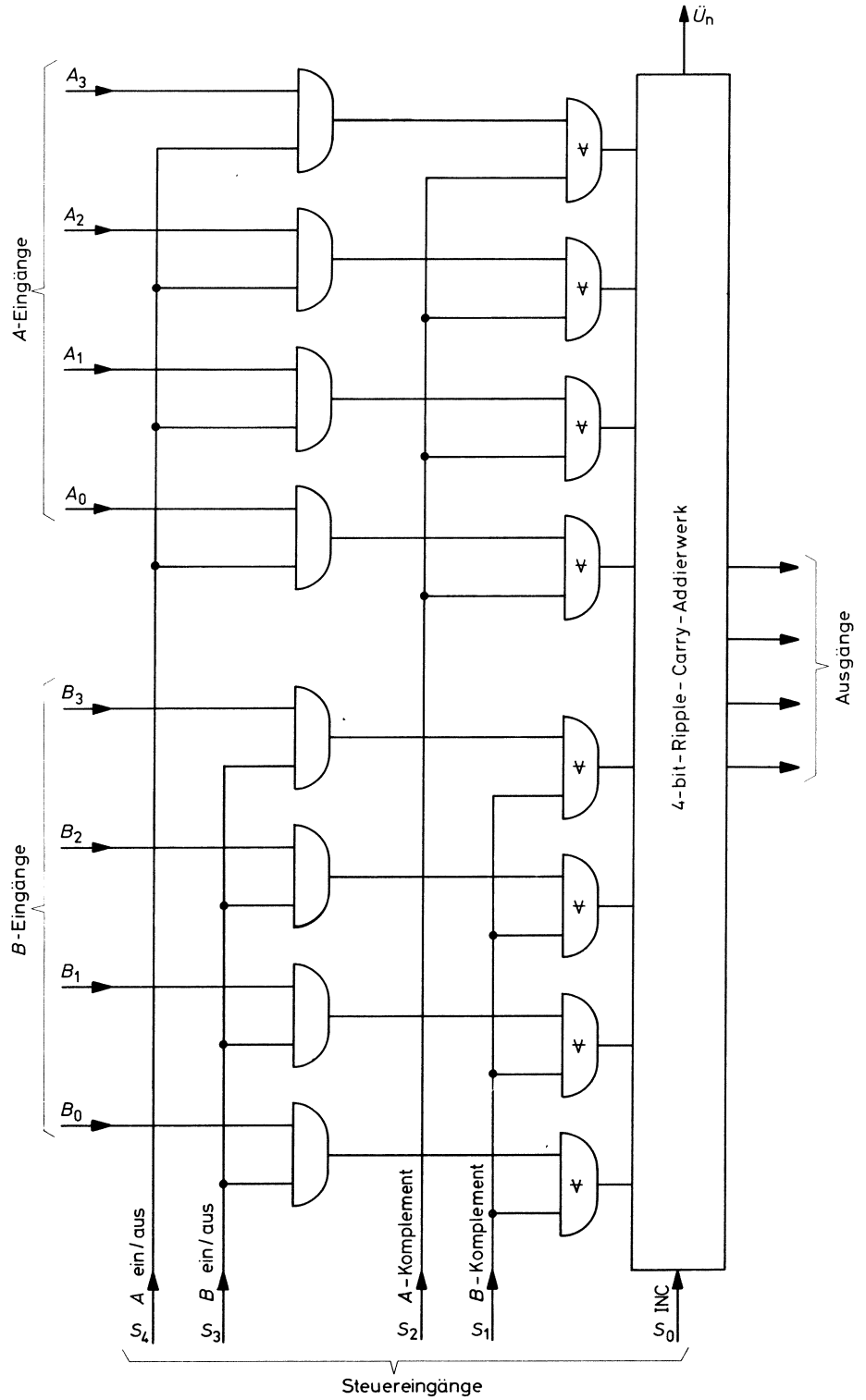


Bild 3.2.1
4-bit-Addier-Subtrahierwerk

Die so entstandene Schaltung enthält 5 Steuereingänge S_4 bis S_0, die es ermöglichen, die *A*- und *B*-Eingänge auf die unterschiedlichste Art miteinander zu verknüpfen. In Tab. 3.2.1 sind alle möglichen Eingangskombinationen mit den dazugehörigen Ausgangsfunktionen dargestellt.

S_4	S_3	S_2	S_1	S_0	Ausgangs-funktion
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	−1
0	0	0	1	1	0
0	0	1	0	0	−1
0	0	1	0	1	0
0	0	1	1	0	−2
0	0	1	1	1	−1
0	1	0	0	0	B
0	1	0	0	1	$B + 1$
0	1	0	1	0	$-B - 1 = \overline{B}$
0	1	0	1	1	$-B$
0	1	1	0	0	$B - 1$
0	1	1	0	1	B
0	1	1	1	0	$-B - 2$
0	1	1	1	1	$-B - 1 = \overline{B}$
1	0	0	0	0	A
1	0	0	0	1	$A + 1$
1	0	0	1	0	$A - 1$
1	0	0	1	1	A
1	0	1	0	0	$-A - 1 = \overline{A}$
1	0	1	0	1	$-A$
1	0	1	1	0	$-A - 2$
1	0	1	1	1	$-A - 1 = \overline{A}$
1	1	0	0	0	$A + B$
1	1	0	0	1	$A + B + 1$
1	1	0	1	0	$A - B - 1$
1	1	0	1	1	$A - B$
1	1	1	0	0	$B - A - 1$
1	1	1	0	1	$B - A$
1	1	1	1	0	$-A - B - 2$
1	1	1	1	1	$-A - B - 1$

Tab. 3.2.1
Steuerfunktionen für das Addier-Subtrahier-Werk nach Bild 3.2.1

Es würde zu weit führen, wollten wir alle 32 möglichen Eingangskombinationen ausführlich diskutieren, wir beschränken uns deshalb auf einige Beispiele. So liefert z. B. die Steuerfunktion S_4 bis S_0 = 1 1 0 0 0 die Ausgangsfunktion $A + B$, d. h., die Eingangssignale werden addiert. Wird S_3 oder S_4 auf 0 gebracht, werden die A- bzw. B-Eingänge abgeschaltet. Am Ausgang erscheint dann nur B oder A. Die oberen EXCLUSIV-ODER-Gatter verknüpfen die A-Eingänge mit S_2. Bei S_2 = 0, werden die A-Eingänge unverändert durchgeschaltet ($A \forall 0 = A$). Bei S_2 = 1, werden die A-Eingänge komplementiert ($A \forall 1 = \overline{A}$). Die Steuereingänge S_1 und S_2 dienen also dazu, die A- bzw. B-Eingänge zu komplementieren. Bei der Steuerfunktion S_4 bis S_0 = 0 0 0 1 0 z. B. ist S_1 = 1 während alle anderen Steuereingänge 0 sind. Dies bedeutet, daß alle Ausgänge der 4 oberen EXCLUSIV-ODER-Gatter 0 sind während die 4 unteren eine 1 liefern. Im Addierwerk wird also folgende Rechenoperation ausgeführt:

$$\begin{array}{rr} A: & 0\,0\,0\,0 \\ B: & +\,1\,1\,1\,1 \\ \hline \text{Ausgang:} & 1\,1\,1\,1 \end{array}$$

Dieses Ergebnis entspricht in der hier gewählten Zweierkomplementarithmetik −1.

Merke:
Bei allen Ausgangsfunktionen in Tab. 3.2.1 liegt die Zweierkomplementarithmetik zugrunde.

Die Steuerfunktion S_4 bis S_0 = 1 1 0 1 1 zeigt die Bedingungen für die Subtraktion $A - B$. Bei der Zweierkomplementarithmetik muß hierzu die Zahl B komplementiert werden (Einer-

komplement $\overline{B}$), hierzu wird dann eine 1 addiert (Zweierkomplement $\overline{B}$ + 1), und das so gewonnene Zwischenergebnis muß zur Zahl A addiert werden. Die Einerkomplementbildung von B wird durch $S_1 = 1$ bewirkt. Durch INC = 1 wird zu $\overline{B}$ eine 1 addiert ($\overline{B}$ + 1). Durch $S_2 = 0$ und $S_3 = S_4 = 1$ gelangt A in der anliegenden Form an das Addierwerk, so daß als Ergebnis die Funktion $A + \overline{B} + 1 = A - B$ erscheint. Durch Umpolen von S_1 und S_2 wird am Ausgang die Differenz $B - A$ gebildet. Wir empfehlen Ihnen, selbst nachzuvollziehen, wie die weiteren Ausgangsfunktionen erklärt werden können.

Exp. 1

3.3 Arithmetische logische Einheit (Arithmetic-Logic-Unit ALU)

Da alle Mikroprozessoren nicht nur arithmetische Verknüpfungen sondern auch logische Verknüpfungen ausführen können, muß die Schaltung nach Bild 3.2.1 erweitert werden. Es gibt hierzu mehrere Möglichkeiten. Im Rahmen dieses Lehrganges werden wir eine Variante behandeln, die einfach zu verstehen ist, obwohl die Lösung zu einer unökonomischen Schaltung führt. Dabei gehen wir davon aus, daß als zusätzliche Funktionen die 3 logischen Verknüpfungen

$A \wedge B$
$A \vee B$ und
$A \barwedge B$

zu bilden sind (jedes bit von A wird mit dem entsprechenden bit von B verknüpft). In Bild 3.3.1 ist eine Lösung für diese Aufgabenstellung gezeigt.

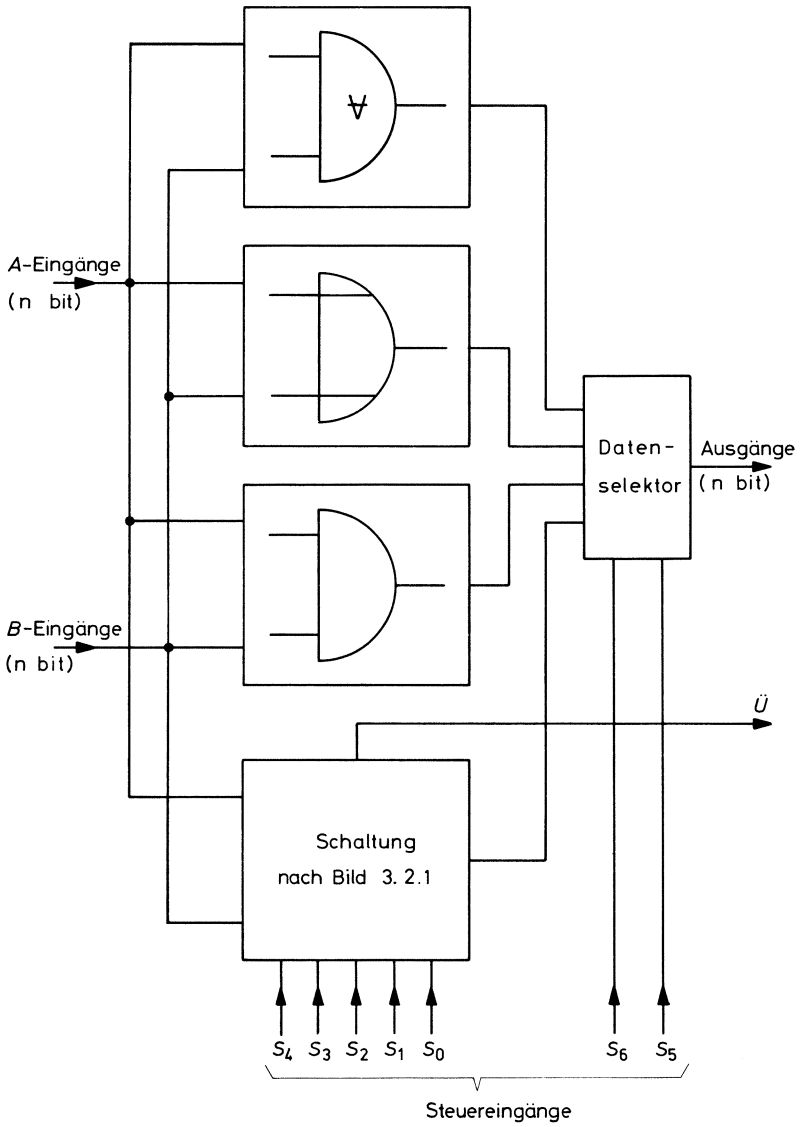


Bild 3.3.1
Arithmetic-Logic-Unit

Wenn die Steuereingänge $S_5 = S_6 = 0$ sind, wird das Addier-Subtrahierwerk durchgeschaltet, und die Funktion entspricht der nach Tab. 3.2.1. Bei einer anderen Beschaltung von S_5 und S_6 wird eine der angegebenen Verknüpfungen durchgeschaltet. In diesem Falle beeinflussen dann die Steuereingänge S_0 bis S_4 das Ergebnis nicht.
Mit 7 Steuereingängen könnte man vom Prinzip $2^7 = 128$ verschiedene Funktionen bilden, die allerdings von der Schaltung gar nicht alle geliefert werden können. Schon die Schaltung nach Bild 3.2.1 enthält Redundanzen, d. h., bestimmte Funktionen wiederholen sich. Tab. 3.2.1 zeigt, daß z. B. die Funktion 0 allein 3 mal vorkommt. Insgesamt enthält die Tabelle nur 24 verschiedene Funktionen. Mit den 3 neuen Funktionen gibt es insgesamt 27 Funktionen, von denen allerdings mehrere keine praktische Bedeutung haben (z. B. $-A - B - 2$). Wir werden uns deshalb auf 13 Funktionen beschränken und kommen dadurch nach einer Umcodierung mit 4 Steuereingängen aus. Die Umcodierung geschieht entsprechend Bild 3.3.2 mit einem ROM.

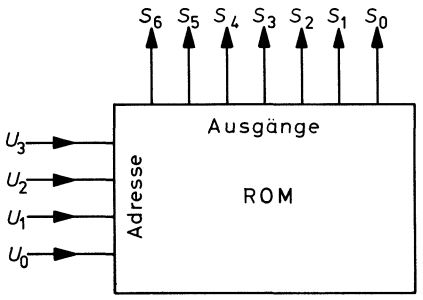


Bild 3.3.2
ROM zur Umcodierung der Steuereingänge

In Tab. 3.3.1 sind die 13 gewünschten Funktionen aufgeschlüsselt.

U_3	U_2	U_1	U_0	Funktion
0	0	0	0	A
0	0	0	1	1
0	0	1	0	$\overline{A}$
0	0	1	1	B
0	1	0	0	0
0	1	0	1	$A + 1$
0	1	1	0	$A - 1$
0	1	1	1	$A + B$
1	0	0	0	$A - B$
1	0	0	1	$A \wedge B$
1	0	1	0	$A \vee B$
1	0	1	1	$A \forall B$
1	1	0	0	-1
1	1	0	1	} für späteren Ausbau
1	1	1	0	
1	1	1	1	

Tab. 3.3.1
Umcodierte Steuerfunktionen

Da mit 4 bit 16 Funktionen möglich sind, bleiben bei der getroffenen Auswahl 3 Funktionen für den späteren Ausbau des Systems übrig. Das für die Umcodierung verwendete ROM benötigt 4 Adreßeingänge, d. h. $2^4 = 16$ Wörter zu je 7 bit. Der Inhalt jedes Wortes ist leicht zu bestimmen. Als Beispiel betrachten wir die Steuerfunktion U_3 bis $U_0 = 0\ 1\ 0\ 1$ mit $A + 1$. Die entsprechende Adresse ist 0 1 0 1. Aus Tab. 3.2.1 ist zu entnehmen, daß auf die Steuereingänge S_4 bis S_0 das bit-Muster 1 0 0 0 1 gelegt werden muß, um die Funktion $A + 1$ zu erhalten. Die Werte für S_5 und S_6 hängen direkt von der Zuordnung des Datenselektors ab. Da, wie bereits erwähnt, für die arithmetischen Funktionen $S_5 = S_6 = 0$ sein müssen, ergibt sich ein Gesamt-bit-Muster S_6 bis S_0 von 0 0 1 0 0 0 1. Dieses Muster muß in dem ROM unter der Adresse 0 1 0 1 gespeichert sein. Nach ähnlichen Überlegungen können auch die restlichen 15 Adresseninhalte bestimmt werden.
Steht ein ROM mit mehr als 7 bit Wortlänge zur Verfügung, können auch noch bestimmte Nebenfunktionen ausgeführt werden. So könnte man z. B. ein zusätzliches bit S_7 dazu

benutzen, den Übertrag der letzten Stelle dann abzuschalten, wenn es das Systemkonzept erfordert. Dies wäre z. B. angebracht im Falle der logischen Verknüpfungen, da hier ein arithmetischer Übertrag keine vernünftige Bedeutung hat. In Bild 3.3.3 ist die gesamte Schaltung der ALU dargestellt.

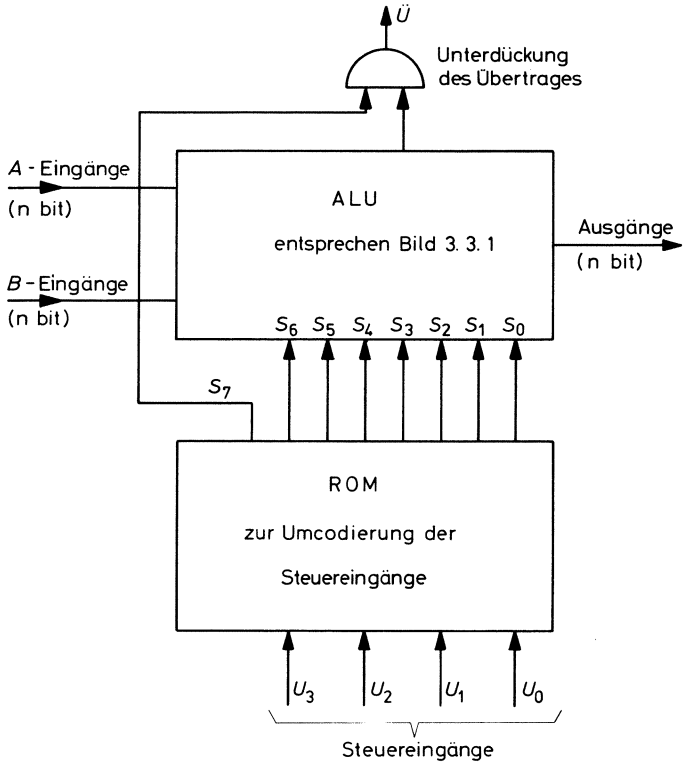


Bild 3.3.3
Komplette ALU mit Umcodierung der Steuereingänge

Exp. 2

3.4 Akkumulator

Der Akkumulator (kurz Akku) ist die nächste Erweiterungsstufe der ALU (Bild 3.4.1).

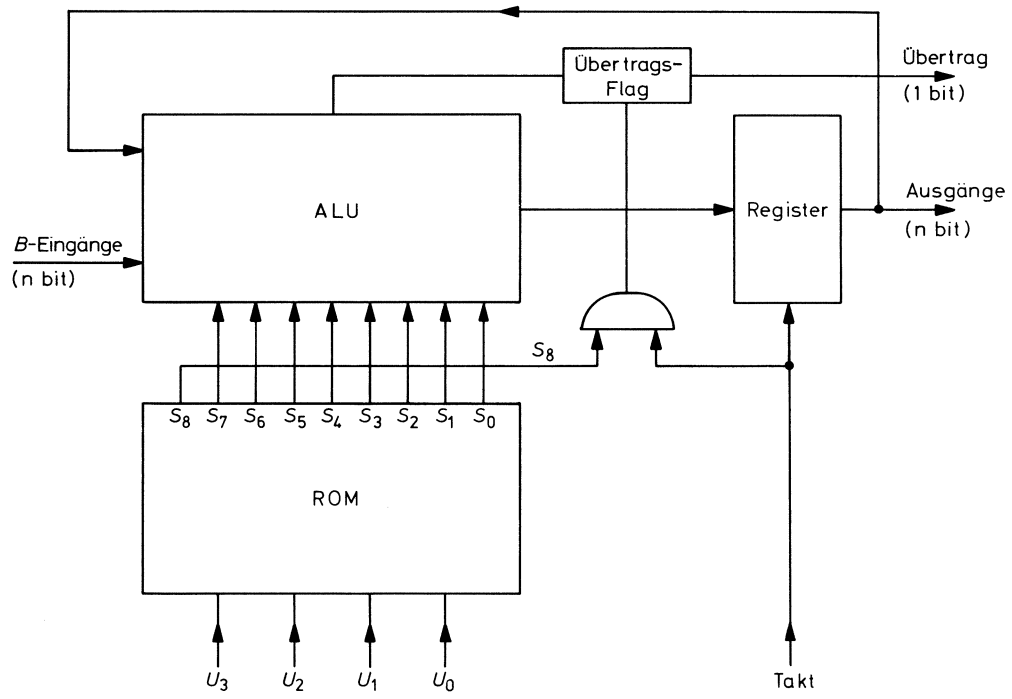


Bild 3.4.1
Akkumulator mit Übertrags-Flag

Außer den Funktionen nach Bild 3.3.3, enthält diese Schaltung als wesentliche Bestandteile noch ein Register sowie ein Übertrags-Flag. Das Register dient zum Zwischenspeichern der Ergebnisse. Hierzu wird eine der Eingangsgruppen (im Beispiel die *A*-Eingänge) mit den Ausgängen des Registers verbunden, so daß eine Art Rückkopplung entsteht. Jetzt werden die Informationen an den *B*-Eingängen mit dem Inhalt des Registers verknüpft. Durch Taktimpulse wird das Verknüpfungsergebnis in das Register geladen, wobei der alte Registerinhalt verloren geht. Damit ein möglicher Übertrag nicht nur kurzzeitig erscheint, wird er in einem Flag ebenfalls zwischengespeichert. Ob dieses Flag getaktet wird oder nicht, wird von einem zusätzlichen bit S_8 im ROM bestimmt (UND-Funktion in Bild 3.4.1). Dies ist erforderlich, da ein Übertrag nur bei sinnvollen ALU-Funktionen gespeichert wird.
Die Funktionen, die mit diesem Akkumulator ausgeführt werden können, lassen sich aus Tab. 3.3.1 ableiten. So führt diese Anordnung z. B. bei einer Steuerkombination U_3 bis U_0 von 0 1 0 1 die Funktion $A + 1$ aus. Da A durch die Rückkopplung dem jeweils vorhandenen Registerinhalt entspricht, wird in diesem Falle mit jedem Taktimpuls der Akkumulatorinhalt um 1 erhöht, d. h., der Akkumulator arbeitet bei dieser Steuerkombination als Zähler. Soll der Zähler rückwärts zählen, so muß über U_3 bis U_0 gleich 0 1 1 0 die Funktion $A - 1$ ausgelöst werden. In Tab. 3.4.1 sind die Funktionen des Akkumulators unter Berücksichtigung der Rückkopplung dargestellt. Die dabei in der Spalte Abkürzung verwendeten Ausdrücke sind allgemein gebräuchlich und von englischen Bezeichnungen abgeleitet.

U_3	U_2	U_1	U_0	Abkürzung	Funktion	Übertrags-Flag
0	0	0	0	NOP	Keine Operation	ja
0	0	0	1	SP1	Setze Akku = 1	ja
0	0	1	0	CMA	Komplementiere Akku	nein
0	0	1	1	LDA	Lade *B* in den Akku	nein
0	1	0	0	CLA	Lösche Akku	nein
0	1	0	1	INC	Incrementiere Akku	ja
0	1	1	0	DEC	Decrementiere Akku	ja
0	1	1	1	ADD	Addiere *B* in den Akku	ja
1	0	0	0	SUB	Subtrahiere *B* von Akku	ja
1	0	0	1	AND	Akku UND *B* in den Akku	ja
1	0	1	0	IOR	Akku ODER *B* in den Akku	ja
1	0	1	1	XOR	Akku EXCLUSIV-ODER in den Akku	ja
1	1	0	0	SM1	Setze Akku = −1	ja
1	1	0	1	–	–	nein
1	1	1	0	–	–	nein
1	1	1	1	–	–	nein

Tab. 3.4.1
Akkumulatorfunktionen der Schaltung nach Bild 3.4.1

Aus der Spalte Übertrags-Flag kann entnommen werden, ob das Flag getaktet wird oder nicht. In vielen Mikroprozessoren wird dieses Flag auch bei logischen Operationen getaktet. Da hierbei aber normalerweise kein Übertrag entsteht, wird das Flag gelöscht.
Sollen mit dem Akkumulator kompliziertere Funktionen ausgeführt werden, müssen diese zunächst in einfachere zerlegt werden. Für die Durchführung einer solchen Operation sind dann mehrere Taktzyklen erforderlich. Als Beispiel soll die Aufgabe $3 \cdot B$ (B = Zahl an den *B*-Eingängen) gerechnet werden. Da es keine Multiplizierfunktion gibt, muß diese durch mehrere einfachere erzeugt werden.
Hierzu sind folgende 3 Steuerkombinationen an U_3 bis U_0 erforderlich:

0 0 1 1
0 1 1 1
0 1 1 1

Die erste Kombination bewirkt, daß beim Takten die Zahl *B* in den Akkumulator geladen wird. Dann wird die Kombination 0 1 1 1 eingestellt und ein zweiter Taktzyklus erzeugt. Jetzt

wird B zum Akku-Inhalt addiert, d. h. $B + B$ gebildet. Mit unveränderter Steuerfunktion wird ein weiterer Taktzyklus benötigt, um zum Zwischenergebnis $B + B$ noch einmal B zu addieren. Als Ergebnis enthält der Akku $3 \cdot B$.
Durch die Auswahl geeigneter Steuerkombinationsfolgen können sehr komplizierte Ausdrücke errechnet werden. Eine solche Steuerkombination wird als **Rechnerbefehl** und eine sinnvolle Folge davon als **Rechnerprogramm** bezeichnet.

3.5 Akkumulator mit Datenspeicher

Um den Akkumulator im Rechner einsetzen zu können, muß die Möglichkeit vorhanden sein, Zwischenergebnisse abzuspeichern und sie später zurückzuholen. Dieses wird mit Hilfe eines Schreib-Lese-Speichers (RAM) erreicht (Bild 3.5.1).

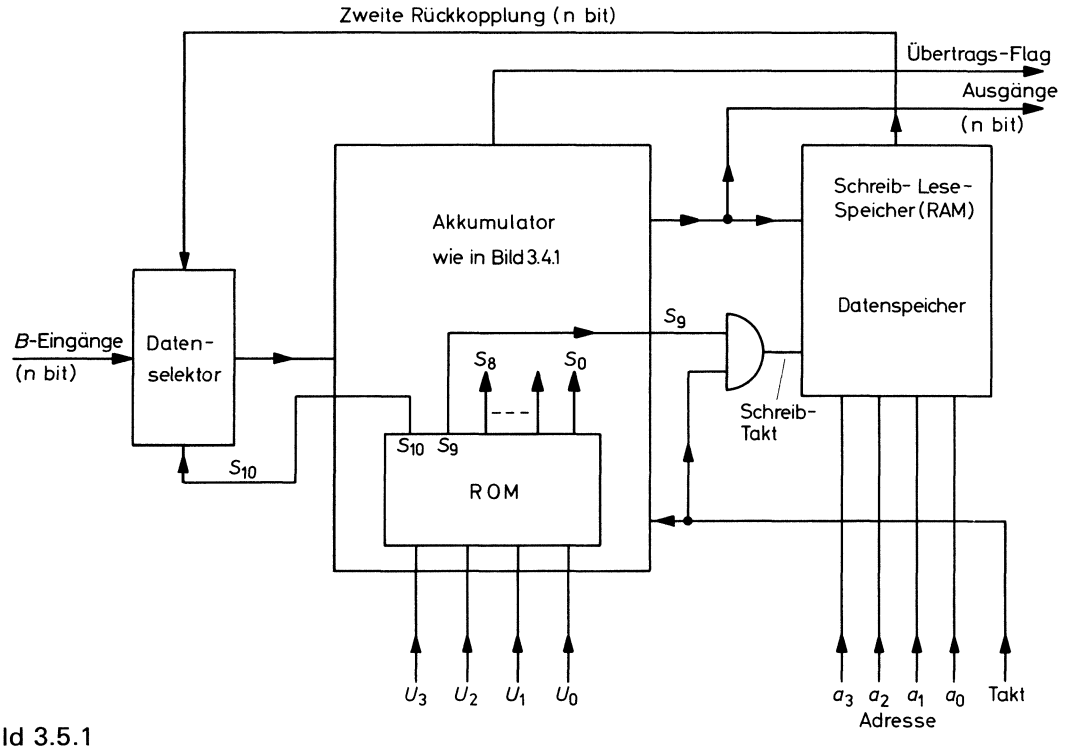


Bild 3.5.1
Akkumulator mit Datenspeicher

Bei dieser Anordnung werden die B-Eingänge über einen Datenselektor gesteuert. Mit dem Datenselektor werden entweder die B-Eingänge oder die Ausgänge des Datenspeichers durchgeschaltet. Die Akkumulatorausgänge gehen nicht nur nach außen, sondern auch zu den Eingängen des Datenspeichers. Dadurch ist es möglich, den Akku-Inhalt in den Datenspeicher oder auch Daten vom Speicher über den Datenselektor in den Akkumulator zu laden. Das Umcodierungs-ROM wird um 2 bit erweitert (S_9 und S_{10}). Die Funktionen dieser Anordnung zeigt Tab. 3.5.1.
Der zusätzliche ROM-Ausgang S_{10} dient zum Umschalten des Datenselektors. Normalerweise ist der Selektor so geschaltet, daß der Datenspeicher mit dem Akkumulator verbunden ist. Bei dem bit-Muster U_3 bis U_0 gleich 1 1 0 1 schaltet der Datenselektor auf die B-Eingänge um, die dann direkt mit dem Akkumulator verbunden sind und deren Daten in dessen Register zwischengespeichert werden. Dabei dürfen die an den Eingängen stehenden Daten in der ALU nicht verändert werden, d. h., die Steuerfunktion der ALU muß so gewählt werden, daß die B-Eingänge unverändert durchgeschaltet werden (S_6 bis S_0 = 0 0 0 1 0 0 0 nach Tab. 3.2.1). Das gleiche gilt für die Steuerfunktion U_3 bis U_0 gleich 0 0 1 1. In diesem Falle werden allerdings die Akku-Eingänge über den Datenselektor mit dem Datenspeicher verbunden.
Das zweite zusätzliche bit im ROM (S_9) dient dazu, dann einen Takt für den Datenspeicher zu erzeugen, wenn das bit-Muster U_3 bis U_0 gleich 1 1 1 0 ist. Bei dieser Steuerfunktion

U_3	U_2	U_1	U_0	a_3	a_2	a_1	a_0	Abkürzung	Funktion	Übertrags-Flag
0	0	0	0	*x*	*x*	*x*	*x*	NOP	Keine Operation	ja
0	0	0	1	*x*	*x*	*x*	*x*	SP1	Setze Akku = 1	ja
0	0	1	0	*x*	*x*	*x*	*x*	CMA	Komplementiere Akku	nein
0	0	1	1	*a*	*a*	*a*	*a*	LDA	Lade Inhalt Adresse *a a a a*	nein
0	1	0	0	*x*	*x*	*x*	*x*	CLA	Lösche Akku	nein
0	1	0	1	*x*	*x*	*x*	*x*	INC	Incrementiere Akku	ja
0	1	1	0	*x*	*x*	*x*	*x*	DEC	Decrementiere Akku	ja
0	1	1	1	*a*	*a*	*a*	*a*	ADD	Addiere Inhalt Adresse *a a a a*	ja
1	0	0	0	*a*	*a*	*a*	*a*	SUB	Subtrahiere Inhalt Adresse *a a a a*	ja
1	0	0	1	*a*	*a*	*a*	*a*	AND	Akku UND Inhalt Adresse *a a a a*	ja
1	0	1	0	*a*	*a*	*a*	*a*	IOR	Akku ODER Inhalt Adresse *a a a a*	ja
1	0	1	1	*a*	*a*	*a*	*a*	XOR	Akku EXCLUSIV-ODER Adresse *a a a a*	ja
1	1	0	0	*x*	*x*	*x*	*x*	SM1	Setze Akku = −1	ja
1	1	0	1	*x*	*x*	*x*	*x*	INP	Lade *B*-Eingänge in den Akku	nein
1	1	1	0	*a*	*a*	*a*	*a*	STA	Speichere Akku in Adresse *a a a a*	nein
1	1	1	1	*x*	*x*	*x*	*x*	–	–	–

a a a a = eine Datenspeicheradresse
x x x x = „don't care"-Zustand, d.h. beliebig

Tab. 3.5.1
Funktionen des Akkumulators mit Datenspeicher

wird nämlich der Akkumulatorinhalt in den Datenspeicher geschrieben. Damit der Akkumulatorinhalt durch diese Operation nicht verändert wird, muß an den Steuerausgängen S_6 bis S_0 des ROMs das bit-Muster 0 0 1 0 0 1 1 erzeugt werden (siehe auch Tab. 3.3.1), da dann der Akkumulatorinhalt wieder in sich zurückgeschrieben wird.

Anmerkung:
Bei Experiment 4 können Sie feststellen, daß die Steuerfunktion U_3 bis U_0 = 1 1 0 1 die *B*-Eingänge durchschaltet, während die Funktionen 1 1 1 0 und 1 1 1 1 den Akku-Inhalt in sich selbst zurückschreiben. Auf das bit-Muster 1 1 1 1 kommen wir noch zu sprechen.

Aus Tab. 3.5.1 ist zu erkennen, daß nicht alle Rechnerbefehle U_3 bis U_0 den Datenspeicher verwenden. In solchen Fällen wie z.B. 0 1 0 0 = CLA = Lösche Akku, haben die Adressen-bit a_3 bis a_0 keine Bedeutung und können deshalb beliebige Werte annehmen. Dies wird durch ein *x* gekennzeichnet.

Exp. 4

3.6 Vereinfachter Rechner

Der nächste Schritt zur Entwicklung eines vollständigen Rechners ist, die Steuermusterfolge in einem **Programmspeicher** zwischenzuspeichern. Damit ist dann letztlich ein automatischer Betrieb möglich (Bild 3.6.1).

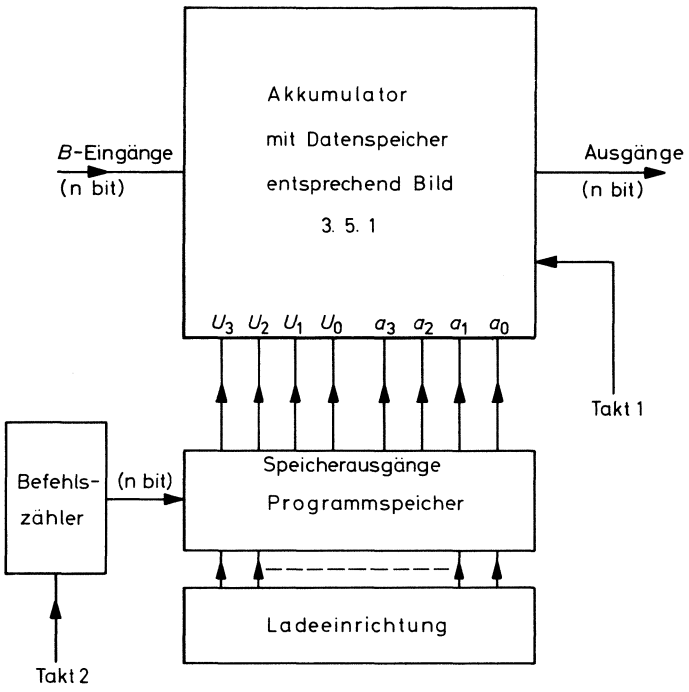


Bild 3.6.1
Anwendung von Befehlszähler und Programmspeicher

Die abzuarbeitende Folge von Steuerwörtern oder Befehlen (das Programm) wird zunächst in den Programmspeicher geladen. Dabei ist natürlich die Reihenfolge der einzelnen Befehle wichtig. Die Befehle werden deshalb im Programmspeicher mit **steigenden aufeinanderfolgenden** Adressen gespeichert. Wenn dann über einen Zähler die Programmspeicheradressen automatisch erzeugt werden, erscheinen die Befehle in der richtigen Reihenfolge und können nacheinander ausgeführt werden. Bevor man allerdings ein solches System benutzen kann, muß das Programm zunächst in den Programmspeicher geladen werden. Dabei sind 2 Möglichkeiten zu unterscheiden:
Wenn das System eine feste Aufgabe hat, z. B. Steuerung eines Aufzuges, wird im allgemeinen während des ganzen Betriebes ein festes Programm benötigt. In solchen Fällen wird ein Festwertspeicher (ROM) benutzt und das Programm beim Herstellungsprozeß des ROMs eingespeichert.
Wenn dagegen das Programm häufig geändert werden muß, z. B. bei der Entwicklung und beim Testen des Programms oder in den Experimenten dieses Lehrganges, wird als Programmspeicher ein Schreib-Lese-Speicher (RAM) benutzt. In diesem Falle muß über eine zusätzliche Logik das Programm in den Programmspeicher geladen werden. Außerdem muß eine Kontrolle des Programms möglich sein.
Obwohl einige Mikroprozessoren getrennte Daten- und Programmspeicher enthalten, wird normalerweise nur ein gemeinsamer Speicher für beide Zwecke benutzt, d. h., der Mikroprozessor hat einen gemeinsamen Adreß- und Datenraum. Der Speicher selbst kann intern als gemischtes ROM und RAM verwirklicht werden. Ein gemeinsamer Speicher hat mehrere Vorteile:
– Größere Flexibilität. Je nach Aufgabenstellung kann die Grenze zwischen Programm- und Datenkapazität vom Anwender festgelegt werden, da manche Aufgaben viel Programm und wenig Daten oder umgekehrt benötigen.
– Es werden weniger Anschlüsse benötigt (bei Mikroprozessoren besonders wichtig).
– Es können auch die Programmschritte als Daten verarbeitet werden (dieser Punkt wird in einem späteren Abschnitt näher behandelt).
Als Nachteil der Einspeicherversion kann der größere Schaltungsaufwand im Mikroprozessor genannt werden. Die Adresse muß hierbei ja entweder vom Befehlszähler oder aber vom Adressenteil des Befehles kommen können. Dafür wird ein Datenselektor benötigt. Auch die Speichereingänge benötigen einen Datenselektor. Zusätzlich wird noch ein Zwischenspeicher (Befehlsregister) für das Befehlswort benötigt, damit der Befehl so lange festgehalten wird, wie der Speicher die Daten aus- oder einliest.
Eine mögliche Realisierung zeigt Bild 3.6.2.
Damit der Rechner anhält, wenn das Programm abgearbeitet worden ist, muß am Ende eines Programms ein HALT-Befehl den Ablauf stoppen. Ohne diesen Befehl hätte das Programm

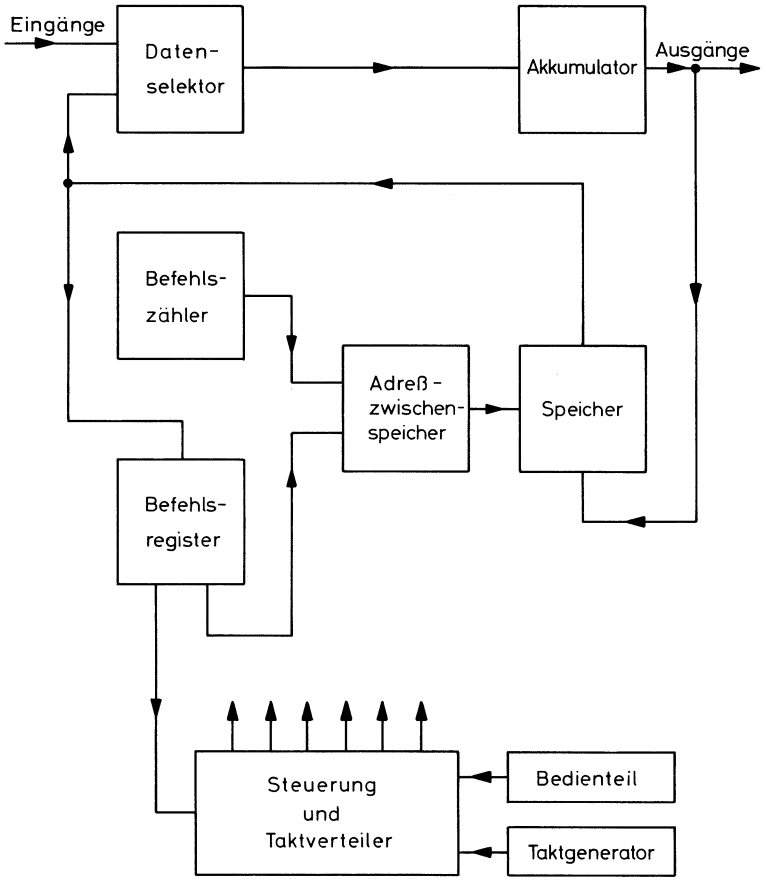


Bild 3.6.2
Einfacher Rechner mit gemeinsamem Daten- und Programmspeicher

kein Ende. Der Rechner würde auch die Daten ausführen und am Ende des Speichers wieder von vorne beginnen. Dem HALT-Befehl ist das Steuermuster U_3 bis U_0 gleich 1 1 1 1 zugeordnet.

Zur Steuerung des Rechenablaufes wird ein **Steuerwerk** benötigt. Ein solches Steuerwerk ist recht kompliziert, und wir werden uns deshalb im Rahmen dieses Lehrganges auf eine kurze Beschreibung beschränken. Die Aufgabe des Steuerwerkes ist es, die verschiedenen Taktimpulse und Steuerwörter für die einzelnen Stufen des Rechners zu erzeugen. Die zu erzeugenden Steuerimpulse hängen jeweils vom gerade auszuführenden Befehl ab. Anhand eines vereinfachten Ablaufdiagramms eines Rechnerbefehles soll das Ganze näher erläutert werden (Bild 3.6.3).

Der Befehlszähler zeigt an, welcher Befehl des Programms (z. B. Nr. 17 des Programms) ausgeführt werden soll. Der Befehlszählerinhalt wird also zuerst auf die Adreßeingänge des Speichers übertragen. Der auszuführende Befehl wird jetzt aus dem Speicher geholt und im Befehlsregister zwischengespeichert. Da der Befehl im allgemeinen aus dem Operationsteil (U_3 bis U_0) und dem Adreßteil (a_3 bis a_0) besteht, muß der Inhalt des Befehlsregisters in Operations- und Adreßteil aufgespalten werden. Aus dem Operationsteil (Op-Code) des Befehles erkennt das Steuerwerk durch eine entsprechende Logik, ob dieser Befehl eine Adresse benötigt oder nicht. Wenn nicht, veranlaßt das Steuerwerk direkt die entsprechende Operation (z. B. U_3 bis U_0 = 0 0 0 1 in Tab. 3.5.1). Wenn ja, wird der Adreßteil über den Adreßzwischenspeicher auf die Adreßeingänge des Speichers gegeben). Das unter der angesprochenen Adresse liegende Datenwort gelangt aus dem Speicher zur Ausführung der Operation in den Akkumulator. Damit ist der Befehl ausgeführt, und der Rechner kann nach Erhöhen des Befehlszählers den nächsten Befehl der Programmliste durchführen. War dieser Befehl ein HALT-Befehl (letzter Befehl jedes Programms), stoppt das Steuerwerk den Rechenablauf.

Die Durchführung eines Befehles erfordert eine bestimmte Anzahl von Taktimpulsen bzw. Taktzyklen. Die Anzahl der Taktzyklen für die verschiedenen Befehle kann verschieden groß sein.

Um bei Mikroprozessoren die vielen Befehle zu ermöglichen, ist eine große Anzahl von Verbindungen zwischen den einzelnen Baustufen erforderlich. Dieses erfordert eine große

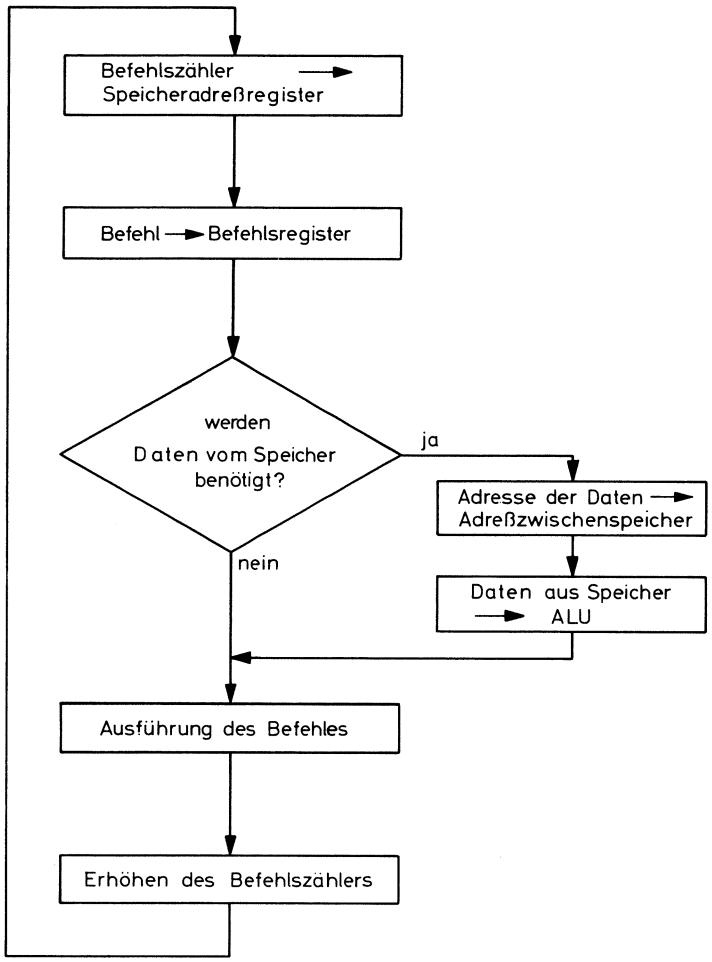


Bild 3.6.3
Ablaufdiagramm eines
Rechnerbefehles

Anzahl von Datenselektoren. Um dies zu vermeiden, wird häufig eine andere Struktur benutzt (Bild 3.6.4).
Diese Struktur basiert auf dem Konzept einer bi-direktionalen Datenbus. Unter Bus versteht man eine Datenleitung, an der mehrere Einheiten gleichzeitig angeschlossen sind. Bi-direktional

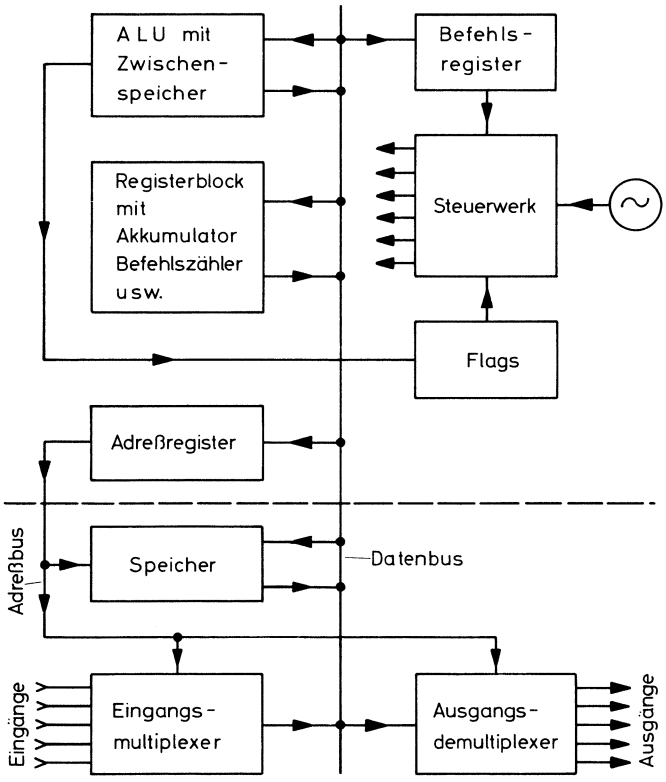


Bild 3.6.4
Busstrukturierter Rechner

bedeutet in diesem Zusammenhang, daß die Daten in beiden Richtungen übertragen werden können. Je nach System gibt es zwischen 1 und 3 Daten- und Adreßbusse, die nach einem Zeitmultiplexbetrieb gesteuert werden. Die verschiedenen Baustufen des Rechners oder Mikroprozessors können Daten auf eine Bus geben und Daten davon abrufen. Das Steuerwerk sorgt dafür, daß zum selben Zeitpunkt nur von einer Baustufe Daten auf die Bus gelangen. Nach diesem System können Daten beliebig innerhalb des Mikroprozessors übertragen werden. Da dieses Verfahren etwas langsamer ist als das Verfahren mit Einzelleitungen, wird in der Praxis häufig eine Mischung zwischen den beiden Systemen benutzt.

Exp. 5

3.7 Vollständiger Rechner

Bei jedem Mikroprozessor gibt es eine ganze Reihe zusätzlicher Befehle, um den Programmablauf zu steuern. Der wichtigste Befehl ist der **Sprungbefehl**. Mit einem Sprungbefehl wird der Programmzähler mit einem im Programm gegebenen Wert geladen. Normalerweise zählt der Programmzähler bei jedem Befehl um einen Schritt weiter, um die Befehle der Reihe nach abzuarbeiten. Mit Hilfe eines Sprungbefehles dagegen kann ein Teil des Programms (oder auch ein Datenteil) übersprungen werden. Auch ist es möglich rückwärts zu springen und einen Teil des Programms zu wiederholen. Dadurch werden Programmschleifen ermöglicht, die sehr häufig gebraucht werden. Beispiele davon werden in den nächsten Abschnitten gebracht. Damit eine Schleife nicht fortlaufend ausgeführt wird, muß es auch noch sog. **bedingte Sprünge** geben. Hier wird der Sprung zu einer gegebenen Adresse nur dann ausgeführt, wenn eine im Befehl spezifizierte Bedingung erfüllt wird, z. B. wenn das Übertrags-Flag gleich 1 ist. Andere Bedingungen werden durch zusätzliche Flags ermöglicht. Typische Flags in einem Mikroprozessorsystem sind:

- **Übertrags-Flag oder C-Flag (C = Carry)**

Dieses Flag zeigt das Verlassen des Zahlenbereiches bei der Integer Arithmetic an.

- **Arithmetischer-Übertrag-Flag oder V-Flag (V = Overflow)**

Dient zur Anzeige beim Verlassen des Zahlenbereiches bei Zweierkomplementarithmetik.

- **Null-Flag oder Z-Flag (Z = Zero)**

Zeigt den Nullzustand eines Ergebnisses an.

- **Negativ-Flag oder N-Flag**

Zeigt ein negatives Ergebnis an.

- **Paritäts-Flag oder P-Flag**

Zeigt die Parität des Ergebnisses an. Darunter versteht man hier, ob das Ergebnis eine gerade oder ungerade Anzahl von Einsen enthält.

Die aufgeführten Flags sind nicht immer alle vorhanden.
Die meisten Mikroprozessoren haben auch Befehle, um die Flags künstlich zu beeinflussen, z. B. um diese zu setzen oder zu löschen.
Ein weiterer wichtiger Befehl ist auch der Sprung-zum-Unterprogramm-Befehl. Hierbei handelt es sich um einen Befehl, nicht nur zu einer bestimmten Adresse zu springen, sondern zusätzlich die momentane Adresse abzuspeichern. Damit besteht die Möglichkeit, später auf die sog. **Rücksprungadresse** zurückzuspringen. Der Begriff Unterprogramm wird in den folgenden Abschnitten ebenfalls noch näher erläutert.

Fragen zu den Abschnitten 3.2 bis 3.6

1. Welche Funktion realisiert das Addier-Subtrahierwerk nach Bild 3.2.1, wenn die Steuereingänge S_4 bis S_0 gleich 1 0 0 1 1 sind? Wie unterscheidet sich diese Funktion von der Funktion, die durch S_4 bis S_0 gleich 1 0 0 0 0 gebildet wird? (Überlegen Sie sich, was die einzelnen UND- und EXCLUSIV-ODER-Gatter im Bild 3.2.1 tatsächlich erzeugen).

2. Welches Steuermuster S_7 bis S_0 wird bei der in Bild 3.3.3 dargestellten ALU für die Funktion $A - B$ benötigt, bzw. welchen Inhalt muß das ROM in der Adresse 1 0 0 0 aufweisen?

3. Mit dem Akkumulator nach Bild 3.4.1 soll die Funktion $1 - 2 \cdot B$ erzeugt werden. Welche Befehle sind hierfür nach Tab. 3.4.1 erforderlich, damit für diese Aufgabe ein Programm geschrieben werden kann?

4. Mit dem Akkumulator nach Bild 3.5.1 soll die EXCLUSIV-ODER-Verknüpfung der unter den Adressen 4_{16} und 5_{16} im Datenspeicher gespeicherten Informationen gebildet werden. Geben Sie hierfür das Programm an, wobei der Maschinencode hexadezimal zu schreiben ist.

5. Die Aufgabe nach 4. soll gelöst werden, ohne dafür den XOR-Befehl zu benützen. Schreiben Sie auch hierfür ein Programm.

Hinweise:

$$A \forall B = (A \wedge \overline{B}) \vee (B \wedge \overline{A})$$

Die 2 UND-Verknüpfungen sind zuerst zu bilden. Hierbei muß eine UND-Verknüpfung zwischengespeichert werden, während die andere gebildet wird. Benutzen Sie hierfür irgendeine Adresse, z. B. F_{16}.

6. Schreiben Sie ein Programm für den vereinfachten Rechner nach Bild 3.6.2, um den Ausdruck $(1 + P - Q) \forall R$ zu berechnen. Hierbei sind P, Q und R im Speicher unter den Adressen D_{16}, E_{16} und F_{16} zu finden.
Überprüfen Sie das Programm mit dem Experimentiersystem und benutzen Sie für P, Q und R irgendwelche Zahlen.

6.

Adresse	Maschinencode	Befehl	Kommentar
00	1 0	SP1	setze Akku = 1
01	7 D	ADD D	bilde 1 + P
02	8 E	SUB E	bilde 1 + $P - Q$
03	B F	XOR F	bilde $(1 + P - Q) \forall R$
04	F 0	HLT	halte an