ITT FACHLEHRGÄNGE

— Elektronik-Seminare —

"Einführung in die Mikroprozessortechnik"

ITT Fachlehrgänge
753 Pforzheim, Postfach 1570

ITT Fachlehrgänge

Einführung in die Mikroprozessortechnik

Inhaltsverzeichnis

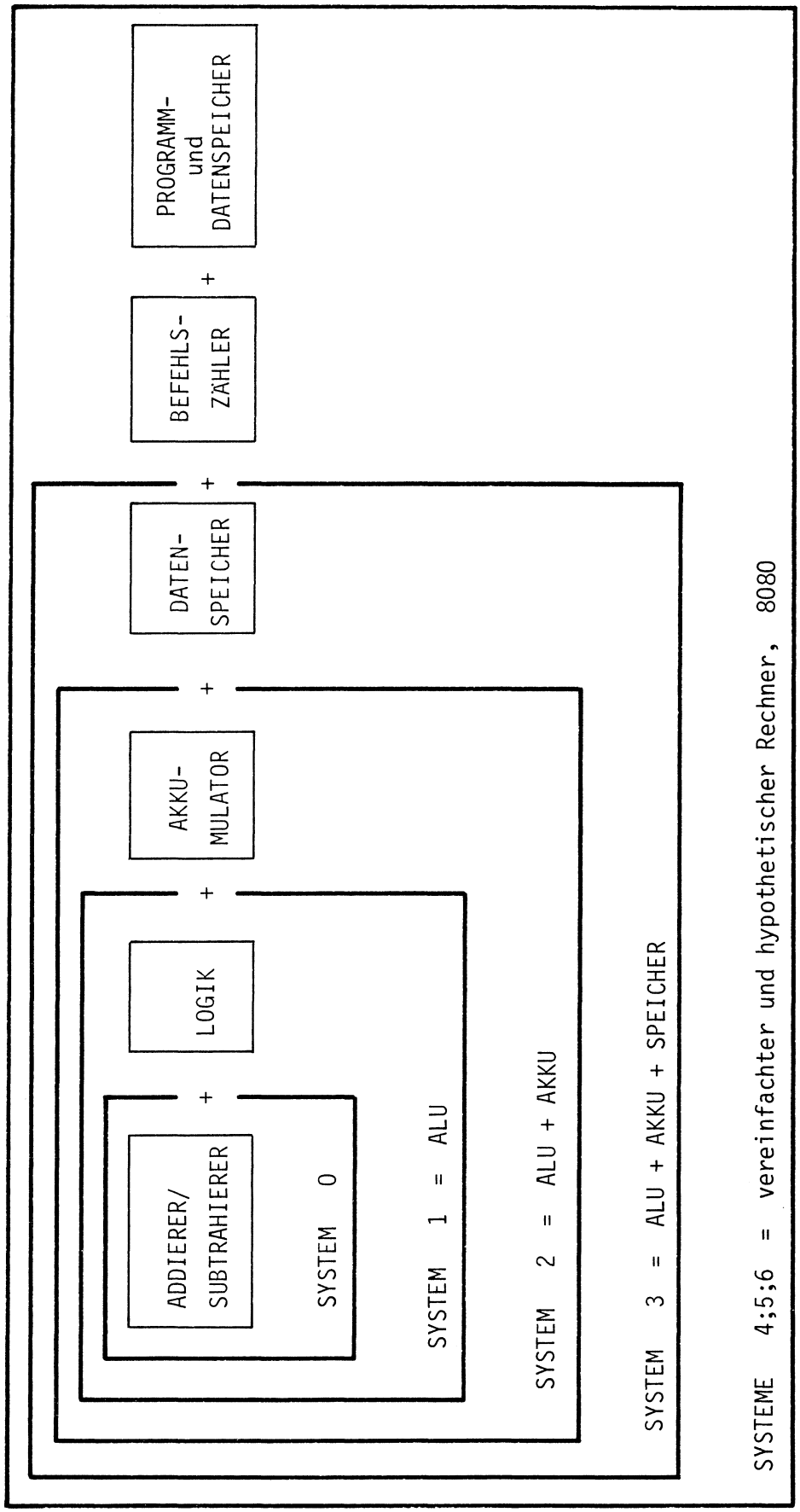
Sehr geehrter Lehrgangsteilnehmer!

Wir möchten Sie darauf aufmerksam machen, daß diese als Manuskript gedruckten Arbeitsunterlagen nur für Ihren persönlichen Gebrauch bestimmt sind. Eine Weitergabe oder Vervielfältigung bedarf daher unserer schriftlichen Zustimmung.

Wir wünschen Ihnen einen angenehmen Aufenthalt und hoffen auf eine gute Zusammenarbeit während des Seminars.

Ihre ITT Fachlehrgänge

(Westerholt) (Imhoff)


ADDIERER/ SUBTRAHIERER
+
LOGIK
+
AKKU- MULATOR
+
DATEN- SPEICHER
+
BEFEHLS- ZÄHLER
+
PROGRAMM- und DATENSPEICHER
SYSTEM 0
SYSTEM 1 = ALU
SYSTEM 2 = ALU + AKKU
SYSTEM 3 = ALU + AKKU + SPEICHER
SYSTEME 4;5;6 = vereinfachter und hypothetischer Rechner, 8080

Rechnerarithmetik

Zahlensysteme

Das Dezimal- oder Zehnersystem ist sicher das am meisten benutzte und uns allen am besten bekannte Zahlensystem. Es baut auf der Zahl 10 auf. 10 ist die Basis des Dezimalsystems, die Ziffern 0 bis 9, also insgesamt 10 Zeichen, werden zur Darstellung benutzt.

Ein Beispiel einer Dezimalzahl ist:

$$2\,305{,}51 = 2 \cdot 10^3 + 3 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 + 5 \cdot 10^{-1} + 1 \cdot 10^{-2}$$

In einem Digitalrechner müssen wir die Zahlen durch bestimmte Werte physikalischer Größen darstellen, z. B. durch Spannung, Strom, Magnetisierung usw. Wir könnten beispielsweise der Spannung 1 V die Zahl 1, der Spannung 2 V die Zahl 2 usw. zuordnen und hätten so im Bereich 0 bis 9 V die 10 Ziffern des Dezimalsystems dargestellt. Eine solche Zuordnung ist technisch durchaus möglich, aber kompliziert und teuer. Viel einfacher ist es, wenn man nur 2 Zustände einer bestimmten physikalischen Größe unterscheidet und diesen beiden Zuständen dann 2 Zahlen, nämlich 0 und 1, zuordnet. Z. B. auf die folgende Art:

0 ≙ kein Strom, keine Spannung, Magnetisierung im Uhrzeigersinn, Logikzustand L
1 ≙ Spannung, Strom vorhanden, Magnetisierung entgegen dem Uhrzeigersinn, Logikzustand H

Moderne Digitalrechner arbeiten ausnahmslos nach diesem Prinzip. Die interne Darstellung von Zahlen basiert somit auf einem Zahlensystem, das mit den beiden Zahlensymbolen 0 und 1 auskommt, dem sog. binären Zahlensystem. Für diese Binärzahlen gelten sinngemäß dieselben Regeln wie für die Dezimalzahlen.
Eine Binärzahl ist z. B. folgendermaßen zu verstehen:

$$1\,0\,1\,1,0\,1 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

Auch das binäre und das dezimale Zählen unterscheiden sich vom Prinzip her nicht. Wenn beim Zählen der Zahlenvorrat einer Stelle überschritten wird, ergibt sich ein Übertrag in die nächste Stelle; im Binärsystem, wenn beim Zählen die Zahl 1 in einer Stelle überschritten wird, im Dezimalsystem, wenn die Zahl 9 in einer Stelle überschritten wird:

Binäre Zahlenreihe	Dezimale Zahlenreihe
0	0
1	1
1 0	2
1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	10
1 0 1 1	11
1 1 0 0	12
.	.
.	.
.	.

Wenn verschiedene Zahlensysteme nebeneinander verwendet werden, so wird im allgemeinen zur Unterscheidung die Basis des Zahlensystems als Index angeschrieben:

$1\,001_{10}$ bedeutet die Dezimalzahl $1 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0$
$1\,0\,0\,1_2$ bedeutet die Binärzahl $1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

Eine Zahl im Binärsystem benötigt ungefähr 3mal so viele Stellen wie dieselbe Zahl im Dezimalsystem, wie nachfolgendes Beispiel zeigt:

$$2\,049_{10} \triangleq 1\,0\,0\;\;0\,0\,0\;\;0\,0\,0\;\;0\,0\,1_2$$

Binäre Zahlen werden deshalb durch ihre Länge sehr schnell unhandlich. Beim praktischen Arbeiten mit Rechnern und binären Zahlen wurde deshalb eingeführt, die Stellen von binären Zahlen in Gruppen von 3 bzw. 4 zusammenzufassen. Man kommt dadurch zu dem Oktal- bzw. dem Hexadezimalsystem. Beim Oktalsystem verwendet man folgende Zuordnung:

Binär	Oktal
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Im Oktalsystem gibt es somit $2^3 = 8$ verschiedene Zahlensymbole, nämlich die Ziffern 0 bis 7. Es hat den Vorteil, ebenso wie das Dezimalsystem, zu Zahlen handlicher Größe zu führen, andererseits aber ist die Übersetzung binär → oktal sehr einfach. Der Programmierer muß lediglich die Binärzahlen des Rechners in Dreiergruppen zusammenfassen, um dann die einzelnen Gruppen in Oktalstellen umzuwandeln:

Binär	1 0 1	1 1 0	0 1 1	1 1 1	,	1 0 1	0 1 0
Oktal	5	6	3	7	,	5	2

$$1\,0\,1\;\;1\,1\,0\;\;0\,1\,1\;\;1\,1\,1,1\,0\,1\;\;0\,1\,0_2 = 5\,6\,3\,7,5\,2_8$$

Im Hexadezimalsystem werden die Binärzahlen zu Vierergruppen zusammengefaßt. Man benötigt also $2^4 = 16$ verschiedene Zahlensymbole, die den Binärzahlen 0 0 0 0 bis 1 1 1 1 zugeordnet werden. Es ist üblich, dafür die Ziffern 0 bis 9 und die Buchstaben A bis F auf die folgende Art und Weise zu verwenden:

Binär	Hexadezimal
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Ein Beispiel einer Hexadezimal → Binärumwandlung ist:

Binär:	1 1 0 1	1 1 1 1	0 0 1 1	0 1 0 1
Hexadezimal:	D	F	3	5

also: $1\ 1\ 0\ 1\ \ 1\ 1\ 1\ 1\ \ 0\ 0\ 1\ 1\ \ 0\ 1\ 0\ 1_2 = D\ F\ 3,\ 5_{16}$

In der Programmierpraxis und der Literatur über Mikroprozessoren werden alle 4 bisher behandelten Zahlensysteme nebeneinander verwendet, manche Hersteller bzw. Autoren entscheiden sich für bestimmte Systeme, andere verwenden von Fall zu Fall das jeweils am besten geeignete Zahlensystem.

Rechnen mit binären Zahlen

In allen Zahlensystemen gelten sinngemäß dieselben Rechenregeln, es ist also möglich, die vom Rechnen mit Dezimalzahlen gewohnten Verfahren auf das Rechnen mit Zahlen in anderen Zahlensystemen zu übertragen. In diesem Abschnitt sollen die 4 Grundrechnungsarten im Binärsystem sowie die logischen Verknüpfungen binärer Zahlen betrachtet werden.

Addition

Bei der Addition binärer Zahlen erhält man immer dann einen Übertrag (carry) von einer Stelle in die nächste, wenn die Summe in einer Spalte gleich oder größer als 2 wird.

	1 0 0 1 0 1	
+	1 0 1 1 1	
	1 1 1	Übertrag
	1 1 1 1 0 0	

Subtraktion

Bekanntlich wird bei dieser Rechenart vom Minuenden *M* der Subtrahend *S* abgezogen und das Ergebnis ist die Differenz *D*. Dies macht auch im Binärsystem keine Schwierigkeiten, wenn pro Stelle der Minuend gleich oder größer als der Subtrahend ist. Ist dies nicht der Fall, muß von der nächstwerthöheren Stelle eine **Entlehnung** erfolgen. Hierzu ein einfaches Beispiel:

2^4	2^3	2^2	2^1	2^0	Stellenwertigkeit
1	0	0	1	0	
0	1	1	0	1	
1	1	0	1		Entlehnung
0	0	1	0	1	

In der 2^0-Stelle muß von 0 eine 1 abgezogen werden. Aufgeschlüsselt bedeutet dies $0 \cdot 2^0 - 1 \cdot 2^0$. Dies ist nur möglich, wenn von der 2^1-Stelle eine Entlehnung erfolgt. Gedanklich wird also die 1 der 2^1-Stelle eine Stelle nach rechts gerückt, und wir erhalten aufgeschlüsselt die Rechenoperation $1 \cdot 2^1 - 1 \cdot 2^0 = 2 - 1 = 1$. Damit erscheint als Differenz in dieser Spalte eine 1. Die erfolgte Entlehnung wird in der 2^1-Stelle durch eine 1 in der Entlehnungszeile ausgedrückt. In dieser Stelle lautet jetzt der Rechengang $1 - 0 - 1 = 0$. In der 2^2-Stelle wiederholt sich der beschriebene Vorgang. In der 2^3-Stelle sind von 0 einmal 1 und von diesem Ergebnis noch einmal 1 durch die Entlehnung zu subtrahieren. Wir lassen zunächst die Entlehnung außer acht und rechnen $0 - 1 = 1$ mit 1 als Entlehnung aus der 2^4-Stelle. Von diesem Zwischenergebnis wird die Entlehnung subtrahiert, und wir erhalten als Ergebnis 0. In der 2^4-Stelle ergeben Minuend und Entlehnung die Differenz 0.

Multiplikation

Die binäre Multiplikation wird nach denselben Regeln durchgeführt wie die dezimale Multiplikation. Man bildet die Produkte mit den einzelnen Stellen des Multiplikators und summiert stellenrichtig auf:

```
1 0 0 1 · 1 1 0 1
  1 0 0 1
    1 0 0 1
      0 0 0 0
        1 0 0 1
-------------------
  1 1 1 0 1 0 1
```

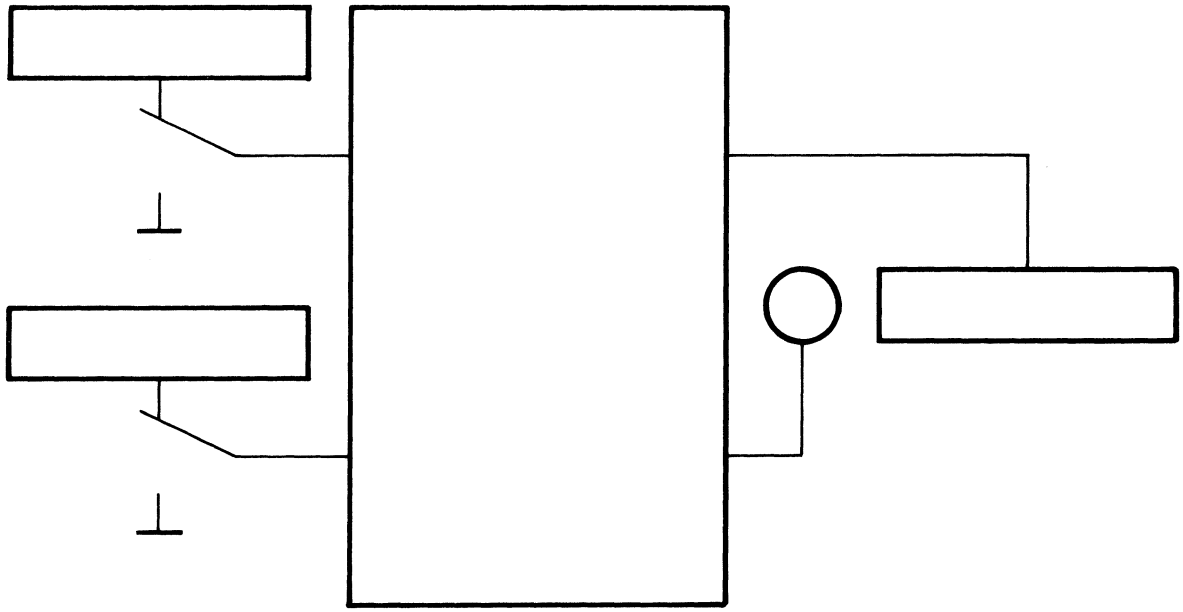
Hierbei ergibt sich eine Vereinfachung gegenüber einer dezimalen Multiplikation. Da die Stellen des Multiplikators nur die Zahlenwerte 0 oder 1 annehmen können, muß der Multiplikand nur mit 0 oder 1 multipliziert werden, d. h., bei der binären Multiplikation kommen als Teiloperationen nur die Addition und die Verschiebung vor.

Division

Die einzelnen Operationen bei einer binären Division sind Subtraktion und Verschiebung. Der Algorithmus (Rechenvorschrift) ist derselbe wie bei der dezimalen Division:

```
  1 1 1 0 1 0 1 : 1 0 0 1 = 1 1 0 1
- 1 0 0 1
-----------
  0 1 0 1 1
- 1 0 0 1
-----------
    0 0 1 0 0
  - 0 0 0 0
-------------
        1 0 0 1
      - 1 0 0 1
-----------------
        0 0 0 0
```

Funktionsschaltung "Addierer" im System 0



Addieren Sie folgende Binärzahlen!

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

b) 1 0 1 1
+ 0 0 1 1

c) 1 1 1 1 1 1 1 1
+ 0 0 0 0 0 0 0 1

d) 1 1 0 1 1 1 0 0
+ 1 0 1 1 1 0 0 1

Arithmetik mit positiven ganzen Zahlen (Integer Arithmetic)

Wir haben bis jetzt nur die Zahlen mit positiven Vorzeichen betrachtet. Außerdem haben wir bis jetzt nicht berücksichtigt, daß jeder Rechner eine bestimmte Wortlänge hat, d. h., daß er nur binäre Zahlen mit einer bestimmten Anzahl von Stellen darstellen kann. Ein 8-bit-Mikroprozessor z. B. kann die Zahlen von 0 0 0 0 0 0 0 0 bis 1 1 1 1 1 1 1 1, also von 0 bis $2^8 - 1 = 255$, verarbeiten. Wenn dieser Zahlenvorrat nicht ausreicht, muß das bei der Programmierung berücksichtigt werden. Für folgende Betrachtungen wollen wir annehmen, wir hätten einen Rechner mit einer Wortlänge von 4 bit, d. h., der Zahlenvorrat geht von 0 0 0 0 bis 1 1 1 1, also von 0 bis $2^4 - 1 = 15$.
Man kann die arithmetischen Operationen anhand des Zahlenstrahles grafisch darstellen (Bild 2.4.1).

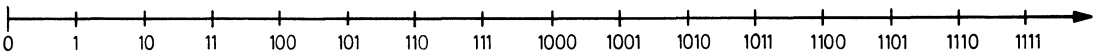


Bild 2.4.1
Zahlenstrahl

Jeder positiven Binärzahl entspricht ein Punkt auf diesem Zahlenstrahl, man kann z. B. die Addition zweier Binärzahlen zurückführen auf die Addition von 2 Strecken auf dem Zahlenstrahl.
Die Operationen in dem 4-bit-Rechner kann man grafisch verdeutlichen durch einen Zahlenkreis (Bild 2.4.2).

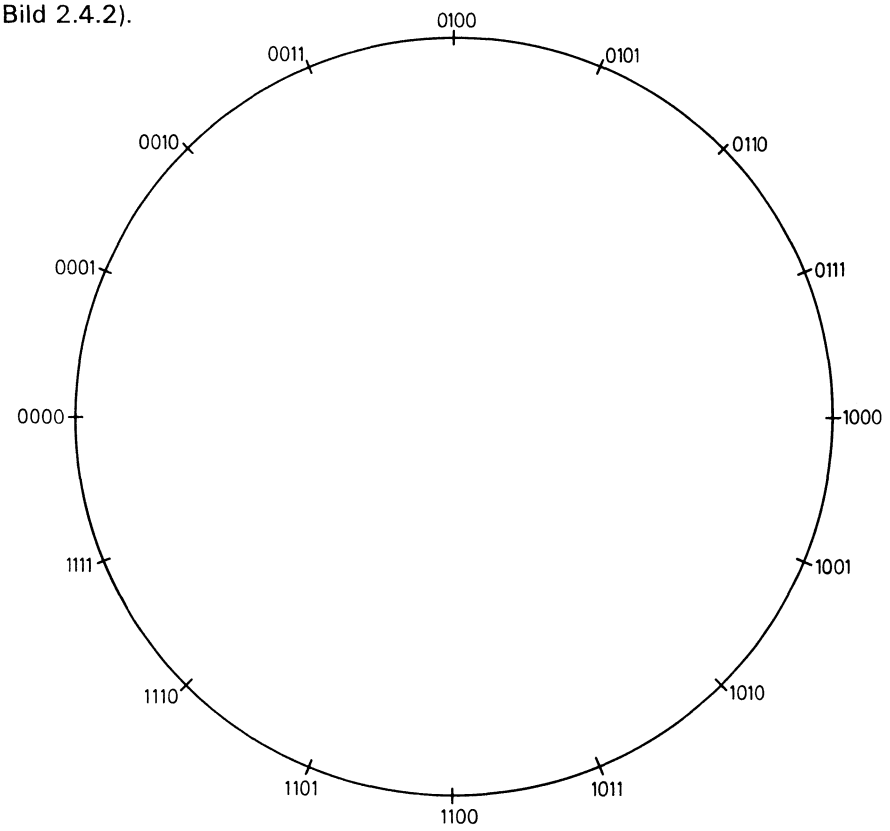


Bild 2.4.2
Darstellung positiver ganzer Zahlen in einem 4-bit-Register anhand eines Zahlenkreises

Die Addition zweier Binärzahlen entspricht hier der Addition zweier Kreisbögen. Solange man den zugelassenen Zahlenvorrat von 0 0 0 0 bis 1 1 1 1 nicht überschreitet, sind die Verhältnisse gleich wie beim Zahlenstrahl. Addiert man jedoch z. B.

$$\begin{array}{r} 1\,1\,1\,1 \\ +\,0\,0\,1\,0 \\ \hline 1\,0\,0\,0\,1 \end{array}$$

so sieht man, daß die Summe 5 bit benötigt, da sich in der vierten Stelle ein **Übertrag** (carry) in die fünfte Stelle ergibt. Diese Zahl paßt jedoch nicht in einen 4-bit-Rechner, man erhält vielmehr ein falsches Ergebnis, nämlich 0 0 0 1. Dieses Ergebnis ist auch auf dem Zahlenkreis abzulesen. Das ist der Fall des sog. **Überlaufes**. Um beim Arbeiten mit dem Rechner diesen Fall erkennen und gegebenenfalls korrigieren zu können, geht der Übertrag in die fünfte Stelle nicht verloren, er wird vielmehr in einem Flipflop gespeichert, dem sog. Carry-FF, kurz Carry-Flag genannt (engl.: flag = Flagge, Signal, Zeichen). Das Carry-Flag kann über ein Programm nach einer arithmetischen Operation geprüft werden, so daß im Falle eines Überlaufes die entsprechenden Maßnahmen ergriffen werden können.
Diese Betrachtungen an dem 4-bit-Beispiel gelten sinngemäß für Rechner mit beliebiger Wortlänge. Beim Rechnen mit positiven Zahlen wird ein Überlauf angezeigt durch einen Übertrag von der werthöchsten binären Stelle. Dieser Übertrag wird in einem Flag gespeichert.

Zweierkomplementarithmetik (Two's Complement Arithmetic)

Positive und negative Zahlen werden auf der Zahlengeraden folgendermaßen dargestellt (Bild 2.5.1).

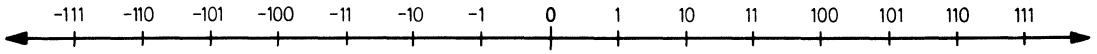


Bild 2.5.1
Darstellung von positiven und negativen Zahlen auf der Zahlengeraden

Bevor wir diese Zahlendarstellung auf den Zahlenkreis übertragen, sollen zuerst einige grundlegende Betrachtungen über negative Zahlen angestellt werden. Es gibt mehrere Möglichkeiten, eine negative Binärzahl darzustellen. Im Hinblick auf den Mikroprozessor soll hier nur die **Zweierkomplementdarstellung** näher erläutert werden. Allgemein versteht man unter Komplement die Ergänzung zur größten gleichstelligen Zahl. Auf das Dezimalsystem übertragen bedeutet dies, daß z.B. das Komplement $\overline{A}$ der Zahl A = 1977 sich ergibt zu:

$$\overline{A} = 9999 - 1977 = 8022$$

Auf Binärzahlen angewandt bedeutet dies, daß das Komplement $\overline{A}$ der Invertierung der Binärzahl entspricht. Beispiel:

$$A = 0\,1\,1\,0\,0\,1$$
$$\overline{A} = 1\,0\,0\,1\,1\,0$$

Hierbei erfolgt also immer eine Ergänzung zu 1. Aus diesem Grunde wird $\overline{A}$ bei Binärzahlen auch als **Einerkomplement** bezeichnet. Addiert man eine Zahl mit ihrem Einerkomplement, so hat das Ergebnis immer die Form 1 1 1 . . . 1 1. Addiert man zu diesem Ergebnis eine 1, so entsteht als neues Ergebnis eine Binärzahl mit der Form **1** 0 0 0 . . . 0 0. Abgesehen von der 1 in der werthöchsten Stelle, die bei einer n-bit-Zahl den Übertrag in die n+1-Stelle darstellt, ist für die n bit das Ergebnis 0. Abgesehen von diesem Übertrag gilt also:

$$A + (\overline{A} + 1) = 0$$

bzw.

$$\overline{A} + 1 = -A$$

Damit ist $\overline{A}$ + 1 eine Darstellung für $-A$. Der Ausdruck $-A$ wird als **Zweierkomplement** bezeichnet. Grundsätzlich gilt: Die Addition einer Binärzahl A mit ihrem Zweierkomplement $-A$ ergibt immer das Ergebnis Null.
Folgendes Beispiel verdeutlicht die Bildung des Zweierkomplementes:

A:	0 1 0 1 1 0 1 0	
$\overline{A}$:	1 0 1 0 0 1 0 1	(Einerkomplement)
+1:	1	
$-A = \overline{A} + 1$:	1 0 1 0 0 1 1 0	(Zweierkomplement)

Wenn wir nun $-A$ mit A addieren, erhalten wir folgendes Ergebnis:

$$\begin{array}{rr} A: & 0\,1\,0\,1\,1\,0\,1\,0 \\ -A: & +1\,0\,1\,0\,0\,1\,1\,0 \\ \hline A+(-A): & \boxed{1}\,0\,0\,0\,0\,0\,0\,0\,0 \end{array}$$

Wenn Sie weitere Beispiele nach diesem Schema durchrechnen, werden Sie feststellen, daß als Ergebnis immer eine Zahl der Form $\boxed{1}\,0\,0\,0\,\ldots\,0\,0$ entsteht.
Wenn nun eine negative Binärzahl über das Zweierkomplement dargestellt werden kann, ist es möglich eine Subtraktion von Binärzahlen auf eine Addition zurückzuführen. Es ist bekannt, daß z. B. die Subtraktionsaufgabe $9 - 3 = 6$ auch als $9 + (-3) = 6$ geschrieben werden kann. Beziehen wir dieses Beispiel auf Binärzahlen, so erhalten wir:

$$\begin{array}{ll} 1\,0\,0\,1 & \triangleq 9 \\ +1\,1\,0\,1 & \text{(Zweierkomplement von } 0\,0\,1\,1 \triangleq 3) \\ \hline \boxed{1}\,0\,1\,1\,0 & \triangleq 6 \end{array}$$

Vom Übertrag 1 abgesehen, erhalten wir als Ergebnis $0\,1\,1\,0 = 6$, also das richtige Ergebnis.
Als nächstes Beispiel wird eine Aufgabe nach dieser Methode gerechnet, die ein negatives Ergebnis hat:

$0\,1\,1\,0 - 1\,0\,0\,1 =$ $\quad \triangleq 6 - 9 = -3$

$$\begin{array}{ll} 0\,1\,1\,0 & \\ +0\,1\,1\,1 & \text{(Zweierkomplement von } 1\,0\,0\,1) \\ \hline 1\,1\,0\,1 & \triangleq 13 \end{array}$$

Dieses Ergebnis ist offensichtlich falsch. Dem **Betrag** nach muß bei dieser Aufgabe als Ergebnis $0\,0\,1\,1$ herauskommen. Wenn wir von $0\,0\,1\,1$ das Zweierkomplement bilden, so erhalten wir:

$$\begin{array}{rll} A: & 0\,0\,1\,1 & \\ \overline{A}: & 1\,1\,0\,0 & \\ & +\,1 & \\ \hline -A: & 1\,1\,0\,1 & \text{(Zweierkomplement von } 0\,0\,1\,1) \end{array}$$

Aus diesem Beispiel geht hervor, daß das erste Ergebnis dem Zweierkomplement der tatsächlichen Lösung entspricht. Da aber nach der Beziehung $\overline{A} + 1 = -A$ durch das Zweierkomplement ein negatives Ergebnis ausgedrückt wird, ist die Lösung $1\,1\,0\,1$ richtig. Wesentlich ist hierbei, daß dieses Ergebnis als Zweierkomplement interpretiert wird. Bilden wir einmal die Zweierkomplemente der Zahlen $0\,0\,0\,1$ bis $0\,1\,1\,1$ (1 bis 7):

A	$-A$
0 0 0 1	1 1 1 1
0 0 1 0	1 1 1 0
0 0 1 1	1 1 0 1
0 1 0 0	1 1 0 0
0 1 0 1	1 0 1 1
0 1 1 0	1 0 1 0
0 1 1 1	1 0 0 1

Die linke Reihe bildet die positiven Zahlen von $0\,0\,0\,1$ bis $0\,1\,1\,1$, die rechte die negativen Zahlen von $-0\,0\,0\,1$ bis $-0\,1\,1\,1$. An diesem Beispiel ist zu erkennen, daß die positiven Zahlen in ihrer werthöchsten Stelle eine 0, die negativen Zahlen eine 1 haben. Durch diese Überlegung ist man nun in der Lage, negative Binärzahlen durch eine 1 in der werthöchsten Stelle zu kennzeichnen. Diese Art der negativen Zahlendarstellung heißt **Zweierkomplementdarstellung**.

Merke:
In der Zweierkomplementdarstellung wird durch eine 0 in der werthöchsten Stelle eine positive Zahl, durch eine 1 eine negative Zahl gekennzeichnet.

Dieser Sachverhalt ist in Bild 2.5.2 dargestellt.

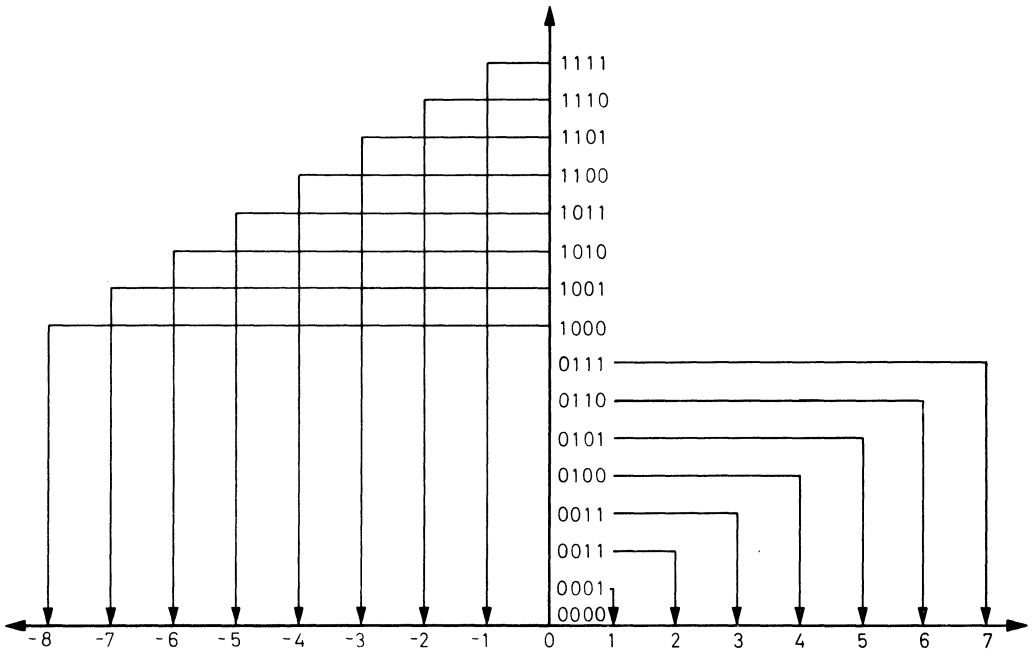


Bild 2.5.2
Zweierkomplementdarstellung

Wenn wir dieses Schema auf den Zahlenkreis übertragen, ergibt sich eine Darstellung nach Bild 2.5.3.

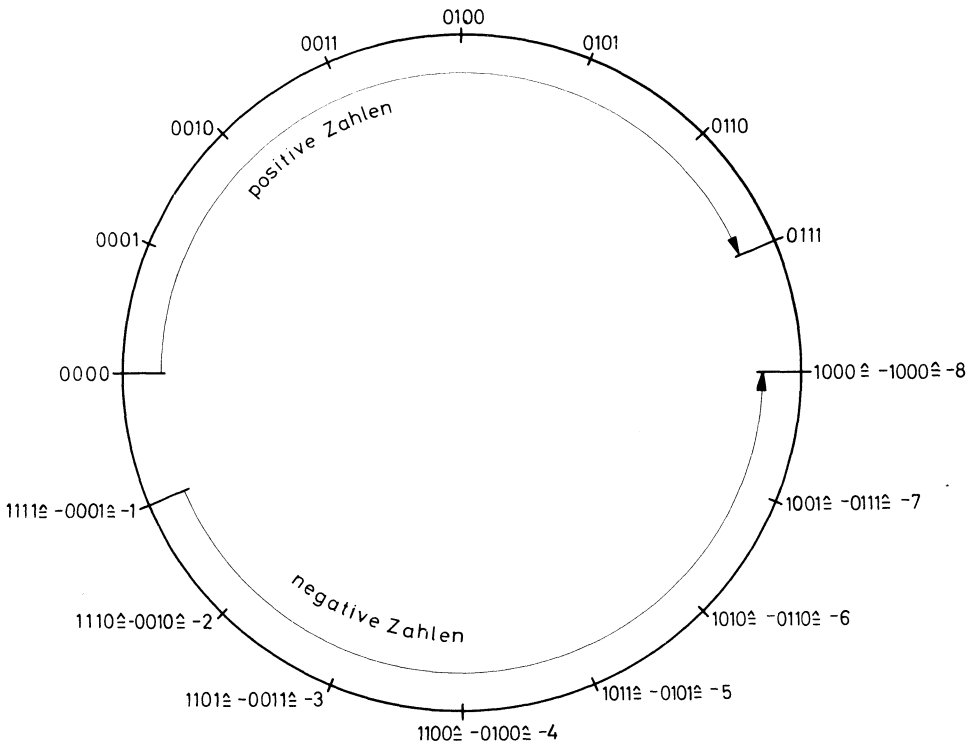


Bild 2.5.3
Darstellung der Zweierkomplementzahlen am Beispiel von 4-bit-Zahlen

Bei dieser Darstellungsart liegen die positiven Zahlen auf dem oberen Halbkreis, die negativen auf dem unteren Halbkreis. Aus diesem Beispiel geht auch deutlich hervor, daß das wert-höchste bit das Vorzeichen bestimmt.
Die dem **Betrag** nach größte negative Zahl ist beim Beispiel der 4-bit-Zahl die Zahl 1 0 0 0, bei einer 8-bit-Zahl ist es 1 0 0 0 0 0 0 0, also immer eine Zahl der Form 1 0 0 0 . . . 0 0. Diese Zahl spielt eine Sonderrolle in dem Zahlenvorrat. Dies soll am Beispiel einer 8-bit-Zahl demonstriert werden, es gilt aber allgemein für beliebige Wort-längen. Bildet man das Zweierkomplement nach der Beziehung $-A = \overline{A} + 1$, so erhält man:

$$\begin{array}{rr} A: & 1\,0\,0\,0\,0\,0\,0\,0 \\ \overline{A}: & 0\,1\,1\,1\,1\,1\,1\,1 \\ 1: & + \qquad\qquad 1 \\ \hline -A: & 1\,0\,0\,0\,0\,0\,0\,0 \end{array}$$

Man erhält also das merkwürdige Ergebnis, daß diese Zahl gleich ihrem negativen Wert ist, eine Eigenschaft die sonst nur die Zahl 0 hat. Dieser Punkt muß vom Programmierer beim Rechnen mit Zweierkomplementzahlen bedacht werden, da er zu falschen Rechenergebnissen führen kann.
Beim Rechnen mit positiven ganzen Zahlen hatten wir die Möglichkeit eines Überlaufes betrachtet. Auch beim Rechnen mit Zweierkomplementzahlen kann ein arithmetischer Über-lauf, also ein Überschreiten des zulässigen Zahlenvorrates, vorkommen. Betrachten wir als Beispiel wieder einen 4-bit-Rechner und führen folgende Operation aus:

$$\begin{array}{r} 0\,0\,1\,0 \\ +\,0\,0\,1\,1 \\ \hline 0\,1\,0\,1 \end{array} \qquad \text{oder dezimal} \qquad \begin{array}{r} 2 \\ +\,3 \\ \hline 5 \end{array}$$

Wir sind bei dieser Operation im oberen Halbkreis des Zahlenkreises geblieben, das Ergebnis ist richtig. Betrachten wir dagegen folgende Aufgabe:

$$\begin{array}{r} 0\,0\,1\,1 \\ +\,0\,1\,1\,0 \\ \hline 1\,0\,0\,1 \end{array} \qquad \begin{array}{r} 3 \\ +\,6 \\ \hline -\,7 \end{array}$$

Wir erhalten in diesem Falle eine Summe, die im unteren Halbkreis liegt, d.h., die Summe ist nach Bild 2.5.3 eine negative Zahl, was ein offensichtlich falsches Ergebnis ist. Diese Operation führte zu einem **arithmetischen** Überlauf.
Hätten wir diese Aufgabe am Zahlenkreis nach Abb. 2.4.2, d.h., also nur mit positiven Zahlen, gerechnet, so wäre das richtige Ergebnis entstanden. Legt man hingegen wie in Bild 2.5.3 die Zweierkomplementdarstellung, d.h. also positive und negative Binärzahlen, zugrunde, so wird der dem **Betrag** nach vorhandene Zahlenbereich halbiert. Damit hat ein 4-bit-Rechner vom **Betrag** her nur eine 3-bit-Kapazität, so daß bei Überschreiten der positiven Zahl 0 1 1 1 $\triangleq$ 7 bereits ein arithmetischer Überlauf entsteht.

Allgemein gilt:
Der Zahlenvorrat eines n-bit-Rechners ist bei Darstellung von
- nur **positiven** Zahlen begrenzt auf 0 bis $2^n - 1$
- positiven **und** negativen Zahlen auf $-(2^{n-1})$ bis $+(2^{n-1}-1)$

Das Ergebnis 1 0 0 1 des letzten Beispieles würde ein Rechner, der nach der Zweier-komplementdarstellung arbeitet, als $-0\,1\,1\,1 = -7$ interpretieren. Manche Mikroprozes-soren enthalten ein weiteres Flag-FF, das sog. V-Flag, das beim Auftreten eines arithmetischen Überlaufes, also eines Überlaufes bei Zweierkomplementarithmetik, gesetzt wird. Die Stellung dieses V-Flags kann wie das Carry-Flag über ein Programm geprüft werden, so daß im Falle des arithmetischen Überlaufes geeignete Maßnahmen ergriffen werden können. In der Integer Arithmetic reicht z.B. bei einem 8-bit-Rechner der Zahlenbereich von 0 bis 256. Per Definition entsteht dann ein Überlauf, wenn z.B. eine Additionsaufgabe das Ergebnis 258 bringt.
Ein 8-bit-Rechner in Zweierkomplementarithmetik hat einen Zahlenbereich von −128 bis +127. Wenn z.B. die Aufgabe 73 + 58 = 131 gerechnet werden soll, entsteht ein Überlauf

in den negativen Zahlenbereich. Ein Überlauf in den positiven Zahlenbereich würde entstehen, wenn wir (−73) + (−58) = −131 rechnen würden. In beiden genannten Fällen wird der vorhandene Zahlenbereich überschritten. Kein Überlauf entsteht, wenn z. B. die Aufgabe −20 + 30 = +10 rechnen. Nachfolgend ist dieses Beispiel mit Binärzahlen durchgeführt:

$$\begin{array}{r} 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \triangleq -20 \text{ in Zweierkomplementarithmetik} \\ +\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \triangleq +30 \text{ in Zweierkomplementarithmetik} \\ \hline 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \triangleq +10 \text{ in Zweierkomplementarithmetik} \end{array}$$

Auf den ersten Blick scheint es hier so, als ob ein Überlauf entstehen würde. Da der Rechner aber nur 8 bit Wortlänge hat, nimmt die 1 im 9. bit keinen Einfluß auf das Ergebnis. Die 0 im 8. bit sagt aus, daß es sich um ein positives Ergebnis mit dem Betrag $1\ 0\ 1\ 0_2 = 10_{10}$ handelt.
Ein weiteres Beispiel:

$$\begin{array}{r} 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1 \triangleq -33 \text{ in Zweierkomplementarithmetik} \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0 \triangleq +56 \text{ in Zweierkomplementarithmetik} \\ \hline 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \triangleq +23 \text{ in Zweierkomplementarithmetik} \end{array}$$

Auch hier hat die 1 im 9. bit keinen Einfluß auf das Ergebnis, ein Überlauf entsteht ebenfalls nicht. Im nächsten Beispiel werden 2 negative Binärzahlen addiert:

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \triangleq -97 \text{ in Zweierkomplementarithmetik} \\ +\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1 \triangleq -89 \text{ in Zweierkomplementarithmetik} \\ \hline 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \triangleq +70 \text{ in Zweierkomplementarithmetik} \end{array}$$

In diesem Falle wird der negative Zahlenbereich überschritten, es entsteht also ein Überlauf. Zum Abschluß noch eine Subtraktionsaufgabe:

$$\begin{array}{r} 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \triangleq +91 \text{ in Zweierkomplementarithmetik} \\ -\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1 \triangleq -69 \text{ in Zweierkomplementarithmetik} \end{array}$$

Durch den Rechenbefehl − bildet der Rechner vom Subtrahenden das Zweierkomplement und addiert dieses zum Minuenden. Die Aufgabe lautet dann:

$$\begin{array}{r} 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ +\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1 \triangleq +69 \text{ in Zweierkomplementarithmetik} \\ \hline 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \triangleq -96 \text{ in Zweierkomplementarithmetik} \end{array}$$

Auch hier erfolgt ein Überlauf, da das richtige Ergebnis 91 − (−69) = 91 + 69 = 160 den positiven Zahlenbereich überschreitet.

1. Bilden Sie zu folgenden Binärzahlen das Einer- und das Zweierkomplement!
   a) 1 0 0 1
   b) 0 1 1 1 0 0 1
   c) 0 0 0 0 0 0 0 0
   d) 1 1 1 1 1

2. Geben Sie an, welche der folgenden 8-bit-Zweierkomplementzahlen positiv und welche negativ sind!
   a) 1 0 1 1 0 1 1 1
   b) 1 1 1 1 0 0 0 0
   c) 0 1 0 1 1 1 1 1
   d) 0 0 0 0 0 0 0 0

3. Berechnen Sie nachfolgende Ausdrücke durch Addition des Zweierkomplementes!
   a)
   $$\begin{array}{r} 1\,1\,0\,1\,1\,0\,1\,1 \\ -\,0\,1\,1\,0\,1\,0\,1\,1 \\ \hline \end{array}$$
   b)
   $$\begin{array}{r} 1\,0\,1\,1 \\ -\,0\,0\,1\,1 \\ \hline \end{array}$$
   c)
   $$\begin{array}{r} 0\,1\,1\,0\,1\,0\,1\,1 \\ -\,1\,1\,0\,1\,1\,0\,1\,1 \\ \hline \end{array}$$

4. Was ist bei arithmetischen Operationen ein Überlauf, und was versteht man unter einem Übertrag?

5. Bei welchen der nachfolgenden Operationen in Integer Arithmetic entsteht bei einem 8-bit-Rechner ein Überlauf bzw. Übertrag?
   a)
   $$\begin{array}{r} 1\,1\,0\,1\,1\,1\,1\,1 \\ +\,0\,0\,1\,1\,1\,0\,0\,0 \\ \hline \end{array}$$
   b)
   $$\begin{array}{r} 0\,1\,0\,1\,1\,0\,1\,1 \\ +\,1\,0\,1\,1\,1\,0\,1\,1 \\ \hline \end{array}$$
   c)
   $$\begin{array}{r} 1\,0\,0\,0\,1\,0\,1\,1 \\ +\,0\,0\,1\,1\,1\,0\,10 \\ \hline \end{array}$$

6. Bei welcher der nachfolgenden Aufgaben in Zweierkomplementarithmetik entsteht bei einem 8-bit-Rechner ein Überlauf?
   a)
   $$\begin{array}{r} 0\,1\,0\,1\,1\,1\,1\,1 \\ +\,0\,1\,1\,0\,0\,0\,0\,1 \\ \hline \end{array}$$
   b)
   $$\begin{array}{r} 1\,1\,1\,0\,1\,0\,0\,0 \\ +\,1\,1\,1\,1\,0\,1\,1\,1 \\ \hline \end{array}$$
   c)
   $$\begin{array}{r} 1\,0\,0\,1\,1\,0\,1\,0 \\ -\,0\,0\,0\,1\,0\,1\,1\,1 \\ \hline \end{array}$$

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.

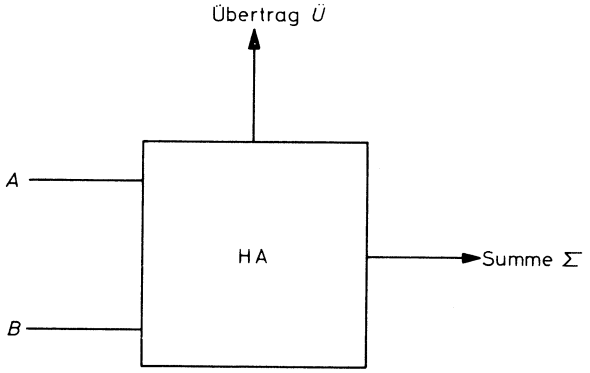
Der nächste Schritt in dieser Entwicklungsrichtung war der, mittels der sog. **L**arge **S**cale **I**ntegration (LSI), was mit hohem Integrationsgrad übersetzt werden kann, durch mehrere tausend Bauelemente die **Zentraleinheit CPU** (CPU = **C**entral **P**rocessing **U**nit) eines Mikrocomputers auf 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.

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:

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

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

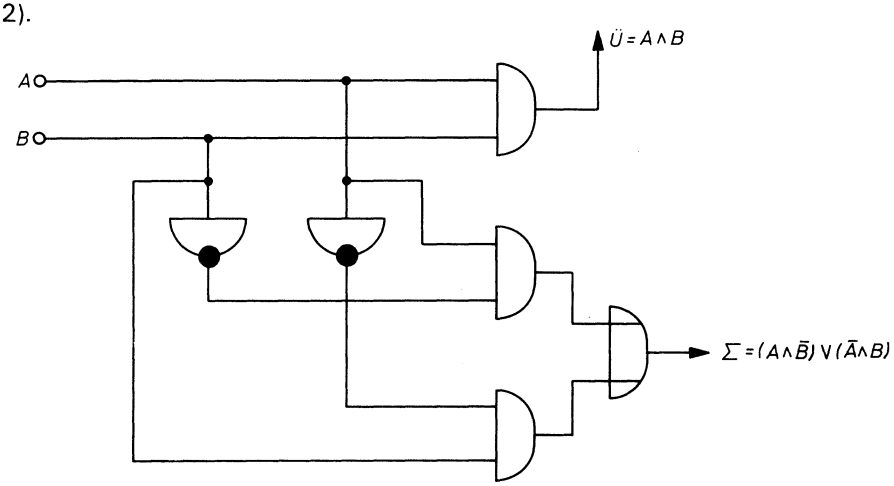
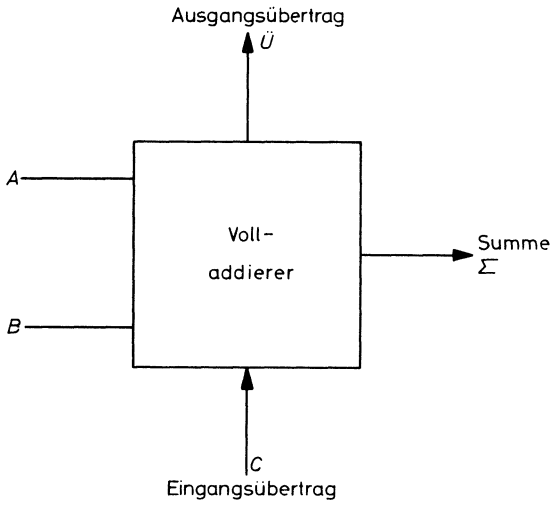


Bild 3.1.2
Schaltung eines Halbaddierers

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	$\ddot{U}$
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.

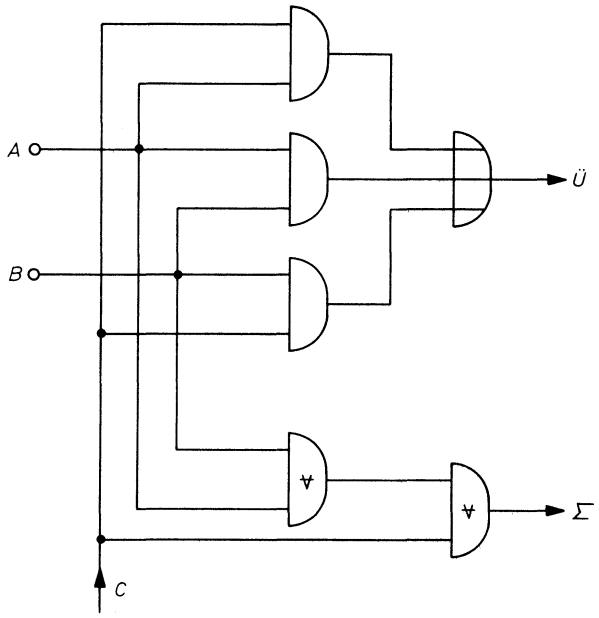


Bild 3.1.4
Schaltung eines Volladdierers

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.

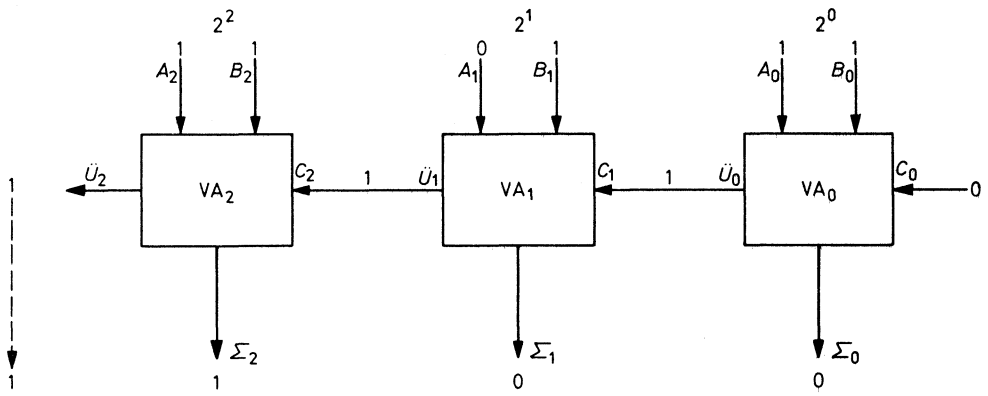


Bild 3.1.5
Addierwerk für 2 3-bit-Binärzahlen

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 **Incrementiereingang** INC (von Increment = Zuwachs). Ein Addierer mit einem Incrementiereingang wird dann als **Ripple-Carry-Addierer** bezeichnet. In Bild 3.1.6 ist ein *n*-bit-Ripple-Carry-Addierer dargestellt.

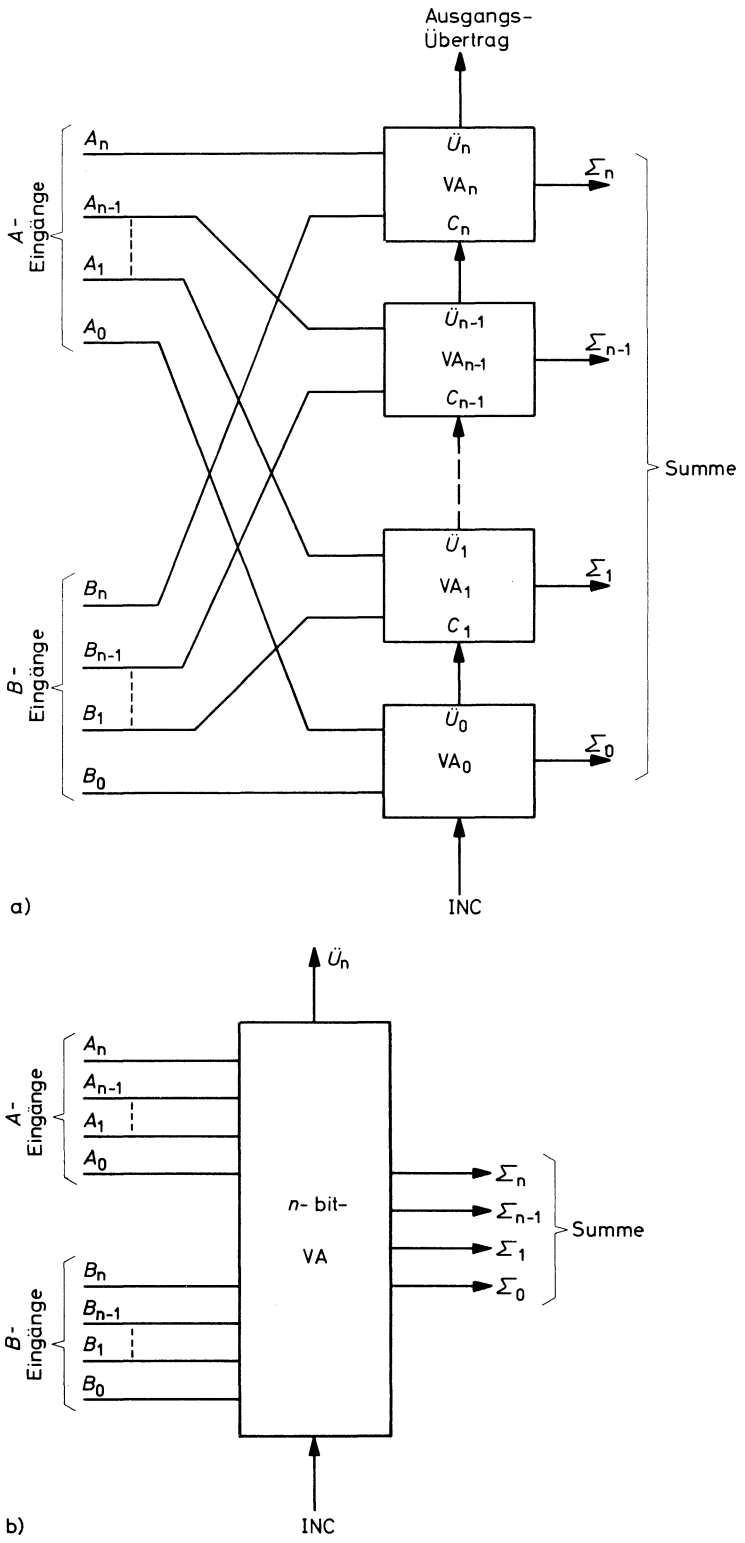


Bild 3.1.6
Ripple-Carry-Addierer
a) Funktionsschaltbild
b) Blockschaltbild

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.

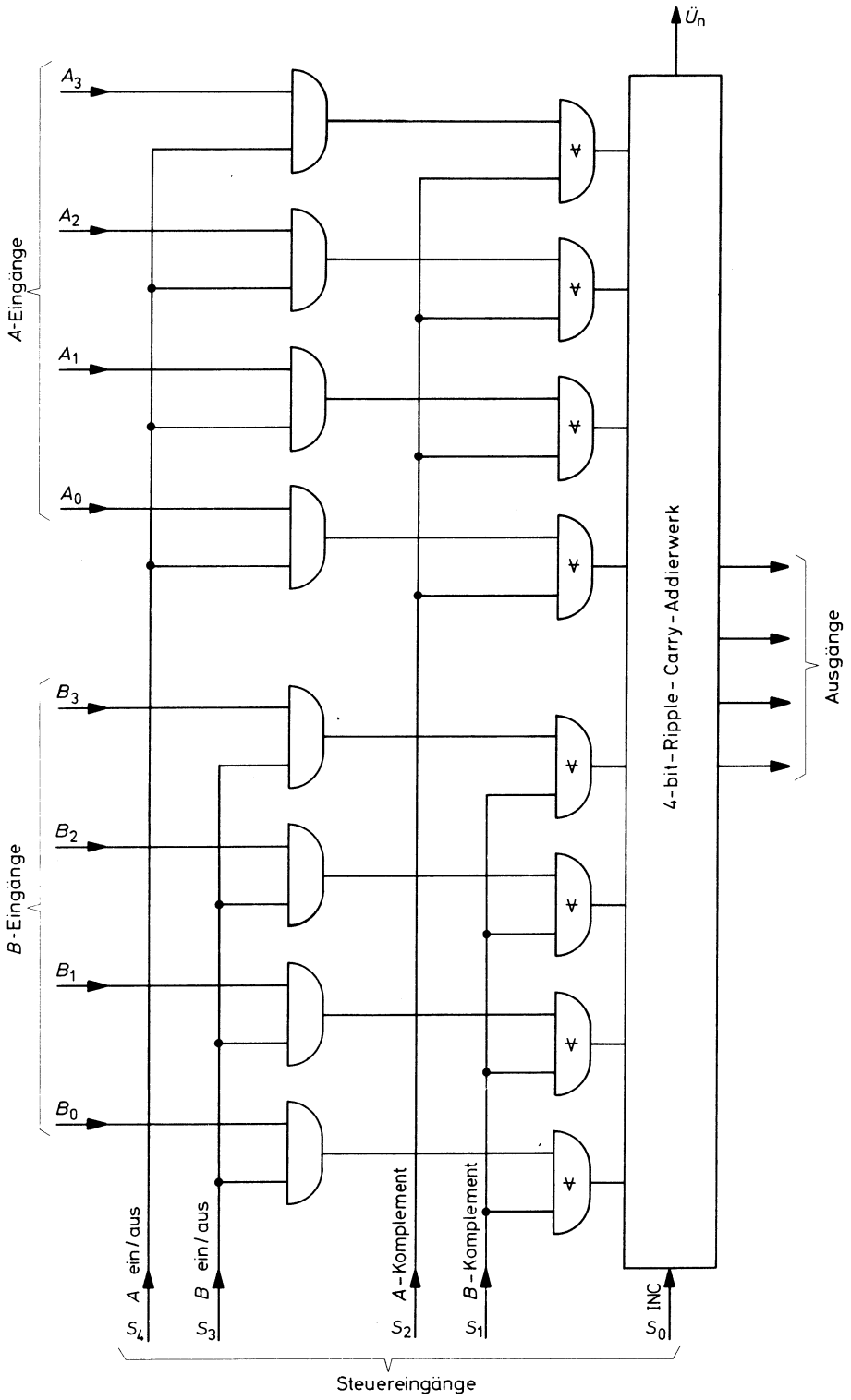


Bild 3.2.1
4-bit-Addier-Subtrahierwerk

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.

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.

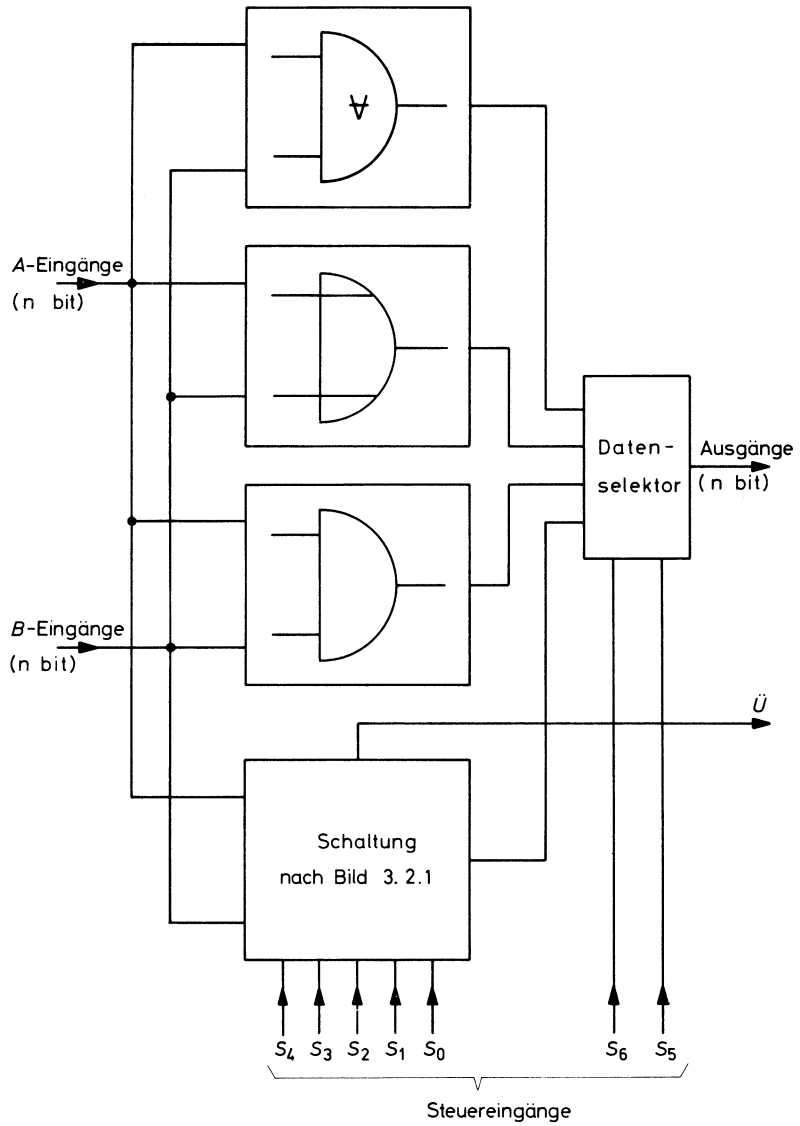


Bild 3.3.1
Arithmetic-Logic-Unit

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.

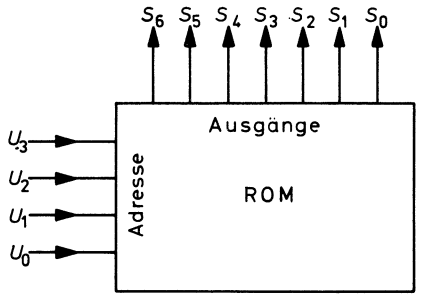


Bild 3.3.2
ROM zur Umcodierung der Steuereingänge

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.

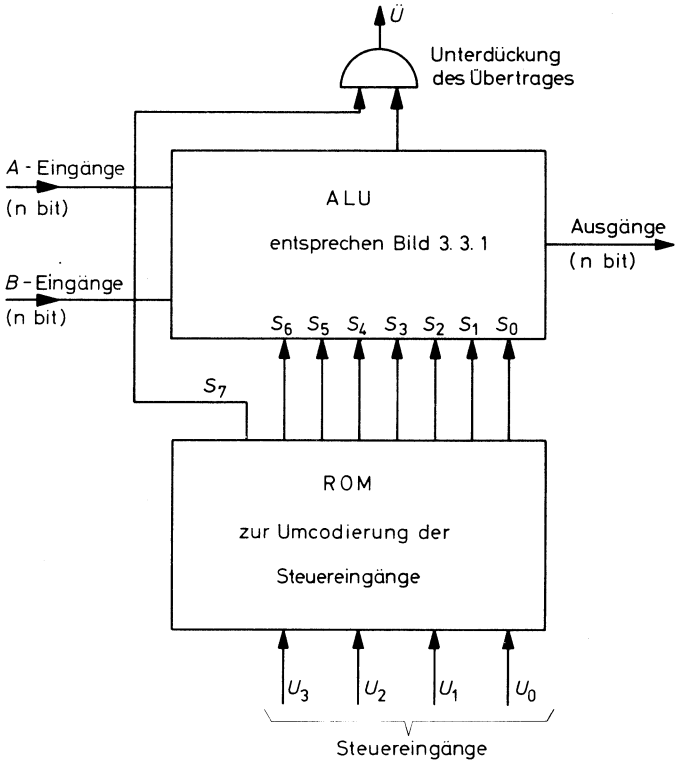


Bild 3.3.3
Komplette ALU mit Umcodierung der Steuereingänge

Akkumulator

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

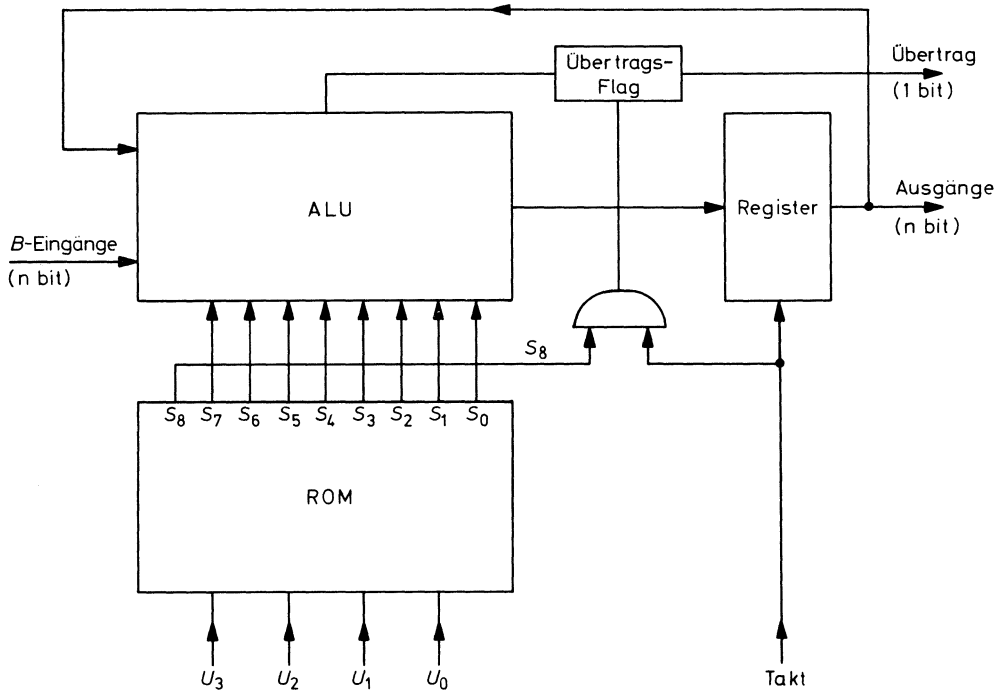


Bild 3.4.1
Akkumulator mit Übertrags-Flag

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.

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).

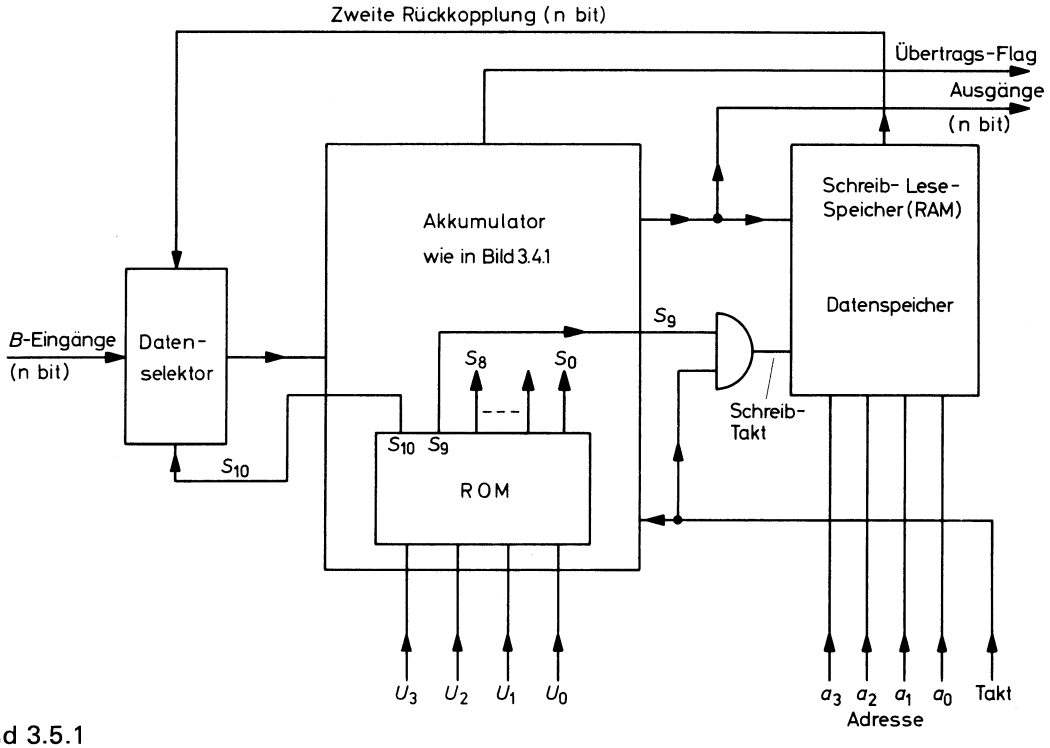


Bild 3.5.1
Akkumulator mit Datenspeicher

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.

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).

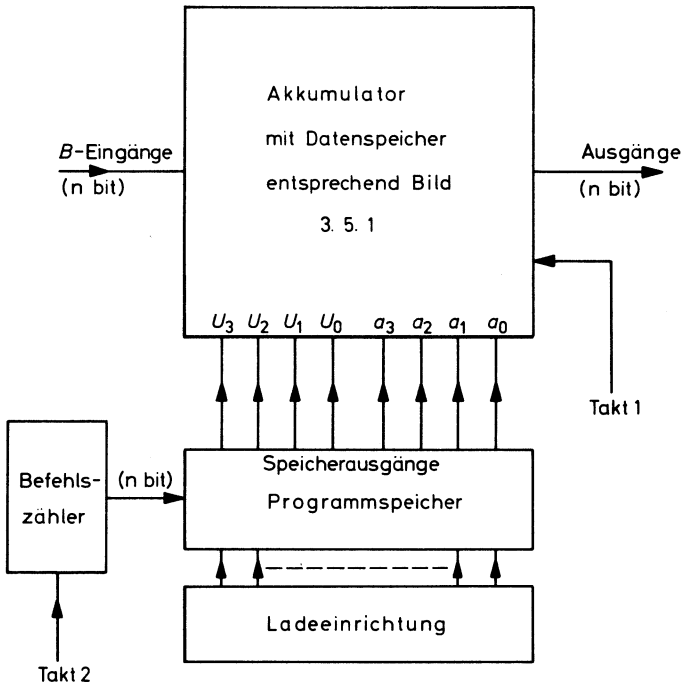


Bild 3.6.1
Anwendung von Befehlszähler und Programmspeicher

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 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 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 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.

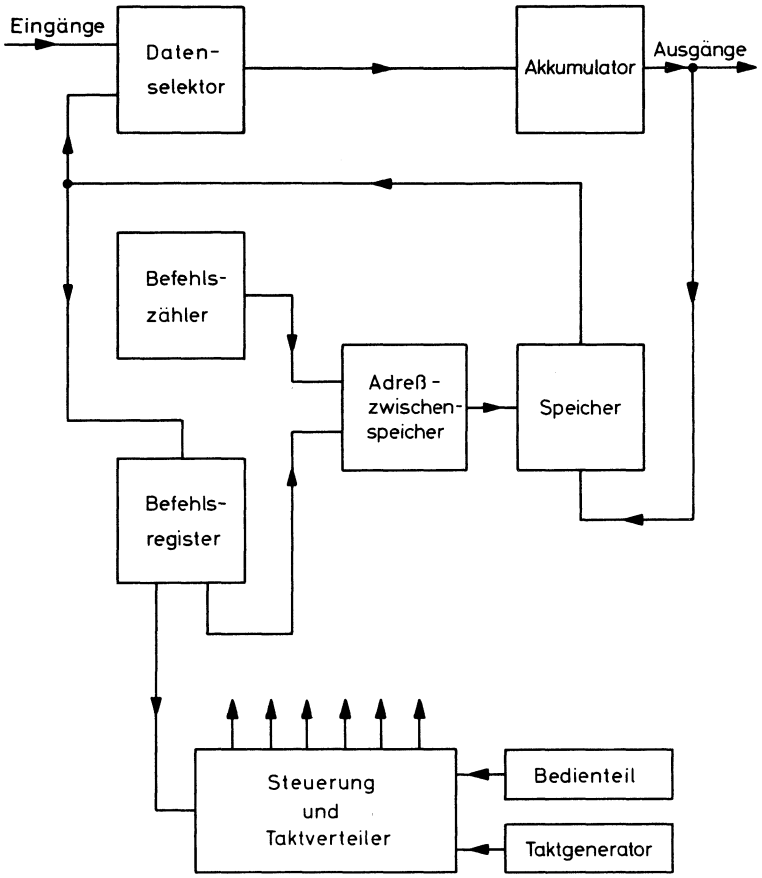


Bild 3.6.2
Einfacher Rechner mit gemeinsamem Daten- und Programmspeicher

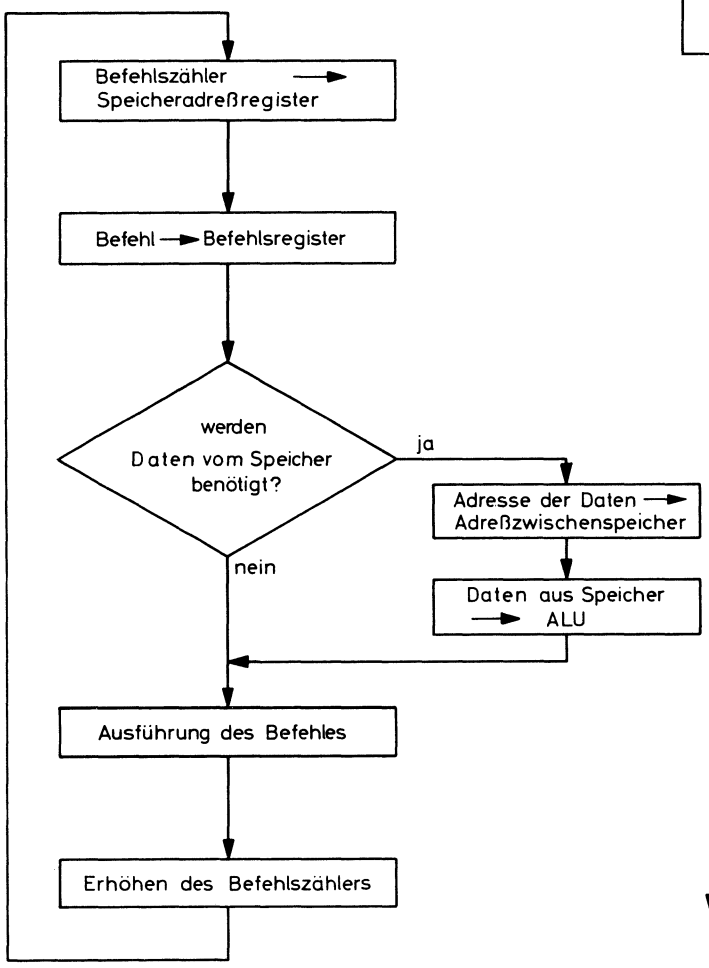


Bild 3.6.3
Ablaufdiagramm eines Rechnerbefehles

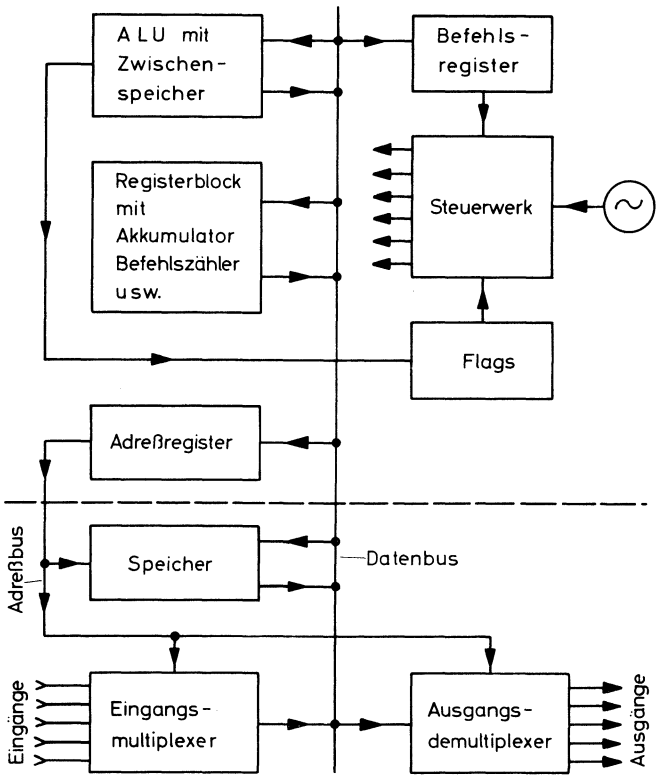


Bild 3.6.4
Busstrukturierter Rechner

Bedienungshinweise für die Systeme 4 und 5

1. Laden von Programmen oder Daten

   RUN und Single Step auf "0"

   Anfangsadresse einstellen und Load Adr. takten

   mit ↑ Deposit: Daten nach der Adresse, die der Befehlszähler enthält

   mit ↓ Deposit: Befehlszähler erhöhen

2. Programmkontrolle

   RUN und Single Step auf "0"

   Anfangsadresse einstellen und Load Adr. takten

   mit ↑ Examine: Daten der Adresse, die der Befehlszähler enthält, in die Anzeige bringen

   mit ↓ Examine: Befehlszähler erhöhen

3. Korrektur falsch geladener Daten oder Befehle

   RUN und Single Step auf "0"

   Adresse des Fehlers einstellen und Load Adr. takten

   mit Deposit neue Daten laden

   Achtung! Deposit nur einmal takten, da sonst auch die auf den Fehler folgenden Adressen neu geladen werden.

4. Programm abarbeiten

   Deposit und Examine auf "0"

   Anfangsadresse einstellen und Load Adr. takten

   a) Single Step: Schalter C_4 auf "1" und mit RUN takten

   b) Freier Lauf: Schalter C_4 auf "0", ↑ RUN startet das Programm

Umwandlungstabelle Dezimal - Hexadezimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

Beispiel:
Größer/kleiner-Vergleich von 2 Zahlen.

In diesem Beispiel werden 2 Zahlen *A* und *B* miteinander verglichen. Die jeweils größere Zahl wird dabei zur Anzeige gebracht. Als Zahl *A* wählen wir $2\,0_{16} \triangleq 1\,0\,0\,0\,0\,0_2$. Die Zahl *A* wird im Speicher abgespeichert. Die Zahl *B* kann an den Schaltern B_7 bis B_0 eingestellt werden. Sie wird dann mit *A* verglichen. Ist *A* größer als *B*, wird *A* angezeigt und umgekehrt. Der Rechner bildet die Differenz $B - A$ und trifft aufgrund des Ergebnisses die Entscheidung, welche Zahl in R_7 bis R_0 angezeigt wird. Der Ablauf ist in einem Flußdiagramm dargestellt (Bild 1).

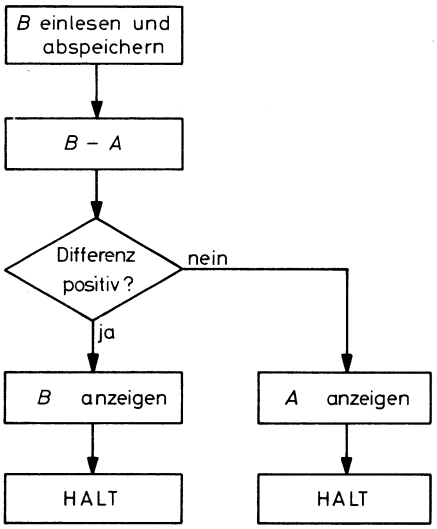


Bild 1
Flußdiagramm zum 4. Beispiel

Wir geben zunächst das Programm für diese Aufgabe an und besprechen anschließend die einzelnen Schritte (Tab. 1).

Adresse		Inhalt		Befehl	Kommentar
hexadez.	binär	hexadez.	binär		
0	0 0 0 0	D *x*	1 1 0 1 *x x x x*	INP	*B*-Eingänge in den Akku laden
1	0 0 0 1	E E	1 1 1 0 1 1 1 0	STA	Inhalt Akku in Adresse E abspeichern
2	0 0 1 0	8 F	1 0 0 0 1 1 1 1	SUB	Subtraktion *B* minus Inhalt Adresse F
3	0 0 1 1	9 D	1 0 0 1 1 1 0 1	AND	höchste Stelle ausblenden
4	0 1 0 0	7 C	0 1 1 1 1 1 0 0	ADD	Entscheidungsaddition
5	0 1 0 1	E 7	1 1 1 0 0 1 1 1	STA	Ergebnis in Adresse 7 abspeichern
6	0 1 1 0	3 F	0 0 1 1 1 1 1 1	LDA	Zahl *A* in Akku laden
7	0 1 1 1	– – –	– – – – – – – –	ADD/HLT	Entscheidungsadresse
8	1 0 0 0	3 E	0 0 1 1 1 1 1 0	LDA	Zahl *B* in Akku laden
9	1 0 0 1	F *x*	1 1 1 1 *x x x x*	HLT	System HALT
A	1 0 1 0		*x x x x x x x x*		nicht belegt
B	1 0 1 1		*x x x x x x x x*		nicht belegt
C	1 1 0 0		0 1 1 1 0 0 0 0	DATEN	Daten für Entscheidungsaddition
D	1 1 0 1		1 0 0 0 0 0 0 0		Maske für Ausblendung
E	1 1 1 0		– – – – – – – –		Adresse für Zahl *B*
F	1 1 1 1		0 0 1 0 0 0 0 0		Zahl *A*

Tab. 1

Entsprechend der Aufgabenstellung soll die im Speicher abgespeicherte Zahl $2\,0_{16}$ mit einer an B_7 bis B_0 eingestellten Zahl verglichen werden. Die Zahl A ist in Adresse F abgespeichert. Wird jetzt an B_7 bis B_0 eine Zahl B eingestellt, so muß diese zunächst in den Akku geladen werden (INP-Befehl).
Mit dem STA-Befehl wird dann die Zahl B in der Adresse E abgespeichert. Dabei wird der Inhalt des Akkus nicht verändert, d.h., B steht weiterhin auch im Akku.
Durch den SUB-Befehl wird die Differenz $B - A$ gebildet. Dabei entsteht bei $B > A$ ein positives und bei $B < A$ ein negatives Ergebnis. Nach der Zweierkomplementarithmetik wird eine positive Zahl durch eine 0, eine negative Zahl durch eine 1 in der werthöchsten Stelle gekennzeichnet. Entscheidungskriterium ist also das werthöchste bit des Ergebnisses der Subtraktion.
Aus diesem Grunde wird jetzt mit einem AND-Befehl das werthöchste bit ausgeblendet.
Beispiel:

```
   0 x x x x x x x      beliebige positive Zahl
∧  1 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0

   1 x x x x x x x      beliebige negative Zahl
∧  1 0 0 0 0 0 0 0
   1 0 0 0 0 0 0 0
```

In Adresse D ist 1 0 0 0 0 0 0 0 gespeichert. Der in Adresse 3 gespeicherte AND-Befehl bildet die UND-Verknüpfung zwischen Inhalt Akku und Inhalt Adresse D. Als Ergebnisse können nur 0 0 0 0 0 0 0 0 (positives Ergebnis der Subtraktion) bzw. 1 0 0 0 0 0 0 0 (negatives Ergebnis der Subtraktion im Akku) erscheinen.
Im nächsten Programmschritt wird nun eine Entscheidungsaddition durchgeführt. Entsprechend dem ADD-Befehl in Adresse 4 wird zu dem ausgeblendeten Ergebnis der Inhalt von Adresse C = 0 1 1 1 0 0 0 0 addiert.
Ist $B > A$ erhalten wir:

```
   0 0 0 0 0 0 0 0
+  0 1 1 1 0 0 0 0
   0 1 1 1 0 0 0 0
```

Ist $B < A$ erhalten wir:

```
   1 0 0 0 0 0 0 0
+  0 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0
```

Entsprechend der Befehlsstruktur des einfachen Rechners stellen die 4 werthöheren bit den OP-Code, die 4 wertniedrigeren bit die Adresse dar. Entscheidend ist, daß der OP-Code 1 1 1 1 den HLT-Befehl darstellt. Wenn wir nun das Ergebnis der Addition über den STA-Befehl in Adresse 7 abspeichern, wird der Rechner beim Abarbeiten des Programmes bei der Adresse 7 entweder anhalten (bei $B < A$) oder, wenn $B > A$ ist, eine Addition zwischen Akku-Inhalt und Inhalt Adresse 0 durchführen. Wenn der Befehlszähler bei $B < A$ gestoppt wird, muß die größere Zahl A angezeigt werden. Dies geschieht dadurch, daß in Adresse 6 der Akku über den LDA-Befehl mit dem Inhalt der Adresse F = $2\,0_{16}$ geladen wird.
Ist dagegen $B > A$, erfolgt bei Adresse 7 eine Addition von Akku-Inhalt und Inhalt Adresse 0, deren Ergebnis aber keine Bedeutung hat. In diesem Falle muß die in Adresse E gespeicherte Zahl B über einen LDA-Befehl in den Akku geladen werden, damit sie in R_7 bis R_0 angezeigt werden kann. In Adresse 9 ist für diesen Fall dann der HLT-Befehl programmiert.
Sicherlich werden Ihnen die hier dargelegten Gedankengänge kompliziert erscheinen. Das liegt ganz einfach daran, daß der vereinfachte Rechner noch keine direkten Entscheidungsbefehle enthält. Wir müssen ihn vielmehr so programmieren, daß er bei einem bestimmten Kriterium selbst einen HALT-Befehl erzeugt. Bevor wir das Programm mit konkreten B-Zahlen noch einmal durchsprechen, muß das Programm zunächst geladen werden:

1. Alle Schalter auf Null stellen
2. Schalter LOAD-ADR. takten

3. Schalter B_7 bis B_4 auf 1 1 0 1 einstellen und DEPOSIT takten
4. B_7 bis B_0 auf 1 1 1 0 1 1 1 0 einstellen und DEPOSIT takten
5. B_7 bis B_0 auf 1 0 0 0 1 1 1 1 einstellen und DEPOSIT takten
6. B_7 bis B_0 auf 1 0 0 1 1 1 0 1 einstellen und DEPOSIT takten
7. B_7 bis B_0 auf 0 1 1 1 1 1 0 0 einstellen und DEPOSIT takten
8. B_7 bis B_0 auf 1 1 1 0 0 1 1 1 einstellen und DEPOSIT takten
9. B_7 bis B_0 auf 0 0 1 1 1 1 1 1 einstellen und DEPOSIT takten
10. DEPOSIT takten. Damit springt der Befehlszähler auf Adresse 8
11. B_7 bis B_0 auf 0 0 1 1 1 1 1 0 einstellen und DEPOSIT takten
12. B_7 bis B_0 auf 1 1 1 1 *x x x x* einstellen und DEPOSIT takten
13. DEPOSIT 2mal takten. Damit springt der Befehlszähler auf Adresse C
14. B_7 bis B_0 auf 0 1 1 1 0 0 0 0 einstellen und DEPOSIT takten
15. B_7 bis B_0 auf 1 0 0 0 0 0 0 0 einstellen und DEPOSIT takten
16. DEPOSIT takten, der Befehlszähler springt auf Adresse F
17. B_7 bis B_0 auf 0 0 1 0 0 0 0 0 einstellen und DEPOSIT takten

Damit steht das Programm im Speicher. Der Befehlszähler steht wieder bei Adresse 0 (L_7 bis L_4). Als Beispiel 1 wollen wir einen Vergleich zwischen $B = 7_{16}$ und $A = 2\ 0_{16}$ durchführen. Damit die einzelnen Schritte nachvollzogen werden können, System auf SINGLE-STEP-Betrieb schalten ($C_4 = 1$). An B_7 bis B_0 wird 0 0 0 0 0 1 1 1 $\triangleq 7_{16}$ eingestellt. Jetzt wird das System mit dem RUN-Schalter schrittweise getaktet.

1. Takt: In R_7 bis R_0 erscheinen die Daten B_7 bis B_0
2. Takt: Daten B_7 bis B_0 werden in Adresse E gespeichert
(Kontrollieren Sie über EXAMINE-Funktion. Vergessen sie nicht Befehlszähler über LOAD-ADR wieder auf Adresse 2 zurückzustellen)
3. Takt: R_7 bis R_0 gleich 1 1 1 0 0 1 1 1 $= -1\ 9_{16}$
4. Takt: R_7 bis R_0 gleich 1 0 0 0 0 0 0 0. Das werthöchste bit wird ausgeblendet
5. Takt: R_7 bis R_0 gleich 1 1 1 1 0 0 0 0. Durch die Addition wird der HLT-Befehl für Adresse 7 gebildet
6. Takt: R_7 bis R_0 bleibt, Akku-Inhalt wird in Adresse 7 abgespeichert
7. Takt: Inhalt Adresse F wird in den Akku geladen und erscheint in R_7 bis R_0
8. Takt: Keine Änderung, da HLT-Befehl. Auch ein weiteres Takten hat keinen Einfluß

Jetzt führen wir den Vergleich mit der Zahl $B = 3\ F_{16}$ durch. Über LOAD-ADR Befehlszähler auf Adresse 0 einstellen, und B_7 bis B_0 auf 0 0 1 1 1 1 1 1 $\triangleq 3\ F_{16}$ einstellen.

1. Takt: In R_7 bis R_0 erscheinen die Daten B_7 bis B_0
2. Takt: Daten werden in Adresse E gespeichert
3. Takt: R_7 bis R_0 gleich 0 0 0 1 1 1 1 1 $\triangleq 1\ F_{16}$
4. Takt: R_7 bis R_0 gleich 0 0 0 0 0 0 0 0 (Ausblenden)
5. Takt: R_7 bis R_0 gleich 0 1 1 1 0 0 0 0. Entspricht hier einem Additionsbefehl für Adresse 7
6. Takt: R_7 bis R_0 bleibt, Akku-Inhalt wird in Adresse 7 gespeichert
7. Takt: Inhalt Adresse F wird in Akku geladen und erscheint in R_7 bis R_0
8. Takt: R_7 bis R_0 gleich 1 1 1 1 0 0 0 0. Dieses Resultat ergibt sich aus der Addition von Akku-Inhalt und Inhalt Adresse 0

Inhalt Adresse 0:	1 1 0 1 0 0 0 0
Inhalt Akku:	+ 0 0 1 0 0 0 0 0
	1 1 1 1 0 0 0 0

Es kann auch ein anderes Ergebnis erscheinen, wenn bei der Programmierung der Adresse 0 die bit b_3 bis b_0 einen anderen Wert gehabt haben. Vom Programm ändert sich nichts, da der INP-Befehl keine bestimmte Adresse spezifiziert (*x x x x*)
9. Takt: Die Zahl B (Inhalt Adresse E) wird in den Akku geladen. R_7 bis R_0 gleich 0 0 1 1 1 1 1 1
10. Takt: Keine Änderung, da HLT-Befehl. Auch ein weiteres Takten hat keinen Einfluß

Entscheidend bei diesem Programm ist der Gedankengang, über eine bestimmte Operation in einer bestimmten Adresse unter einer bestimmten Voraussetzung einen HALT-Befehl zu erzeugen.

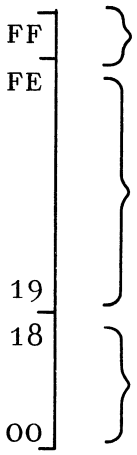
Blockschaltung zum hypothetischen Rechner (System 5)

Die nachfolgende Blockschaltung wurde speziell auf die Eigenschaften des hypothetischen Rechners zugeschnitten. Ein Teil der Funktionen dieses Systems, die später noch näher erläutert werden, ist hier als Hardware eingezeichnet. Beim Betrieb des Systems 5 werden diese Funktionen jedoch durch Software, d.h. durch Hilfsprogramme, realisiert. Da die Ausführungszeit für diese Hilfsprogramme im Millisekundenbereich liegt, erscheinen die einzelnen Funktionen dem Anwender des Systems als sofortige Reaktion auf die Betätigung der einzelnen Schalter.

Zunächst sollen die Eigenschaften der einzelnen Funktionsblöcke erläutert werden.

1. Speicheraufteilung

Das System 5 verfügt über einen 8 bit-Adressbus, d.h. es können 256_{1o} Speicherplätze adressiert werden (1/4 KByte). Davon sind jedoch nicht alle für den Anwender frei verfügbar. Das folgende Schema soll die Aufteilung des gesamten Adressbereiches graphisch verdeutlichen.



2. CPU = central processing unit = Zentraleinheit

Hier sollen in Stichworten die wesentlichen Eigenschaften der CPU des Systems 5 beschrieben werden.

Register:

Befehls-
zähler:

A L U : Arithmetik:

Logik:

Flags: N =
Z =
C =

3. Display-Selector

Im System 5 sind noch keine Datenausgabebefehle vorhanden mit denen Registerinhalte in den beiden Lampenreihen zur Anzeige gebracht werden könnten. Diese Aufgabe übernimmt der Display-Selector. Mit den Display-Codes 000 bis 011 können die Inhalte der Register R0 bis R3 wahlweise in einer der (oder bei gleichen Codes in beiden) Lampenreihen angezeigt werden. Als Display-Selector arbeiten die Schalter B2,B1,B0 für die rechte Lampenreihe und B5,B4,B3 für die linke Lampenreihe. Außer der Anzeige von Registerinhalten bietet der Display-Selector noch die auf der nächsten Seite beschriebenen Möglichkeiten.

Display-
Code

100 : Der aktuelle Stand des Befehlszählers wird angezeigt.

101 : Die Flags werden entsprechend der Schablonenbeschriftung angezeigt.
R = Run steht auf logisch "1" wenn im Rechner ein Programm läuft.

110 : Der Inhalt der im Befehlszähler enthaltenen Adresse wird angezeigt.

111 : Der Inhalt einer beliebigen, mit den A-Schaltern wählbaren, Adresse wird angezeigt.

Ein als Hardware aufgebauter Display-Selector würde einen sehr großen Aufwand an Datenleitungen erfordern, da er Zugriff zu allen darzustellenden Daten haben müsste. Im System 5 ist daher ein Hilfsprogramm vorhanden, das in seiner Wirkung nach zwei verschiedenen Betriebszuständen betrachtet werden muss.

a) Kein laufendes Programm oder Single Step
Die B-Schalter werden zyklisch abgefragt und die selectierten Daten in die Lampenreihen gebracht.

b) Laufendes Programm
Nach der Ausführung jedes einzelnen Befehls werden die selectierten Daten auf den neuesten Stand gebracht.

4. Betriebssystem

Die Betriebsschalter C_4 bis C_0 könnten ebenfalls als Hardware-Einrichtungen aufgefasst werden. Die auch hier wieder verwendeten Hilfsprogramme werden in ihrer Gesamtheit als Betriebssystem eines Rechners bezeichnet. Diese Programme sorgen durch zyklisches Abfragen und Auswerten der C-Schalter dafür, daß beim Betätigen der Schalter die jeweils zugeordneten Funktionen ausgeführt werden.

Die Wirkung der C-Schalter im System 5 unterscheidet sich nur geringfügig von der im System 4.

C_0 = LOAD ADR. auf logisch "1"

Zurücknehmen des Schalters C_0 bringt den Display-Selector wieder in Betrieb!

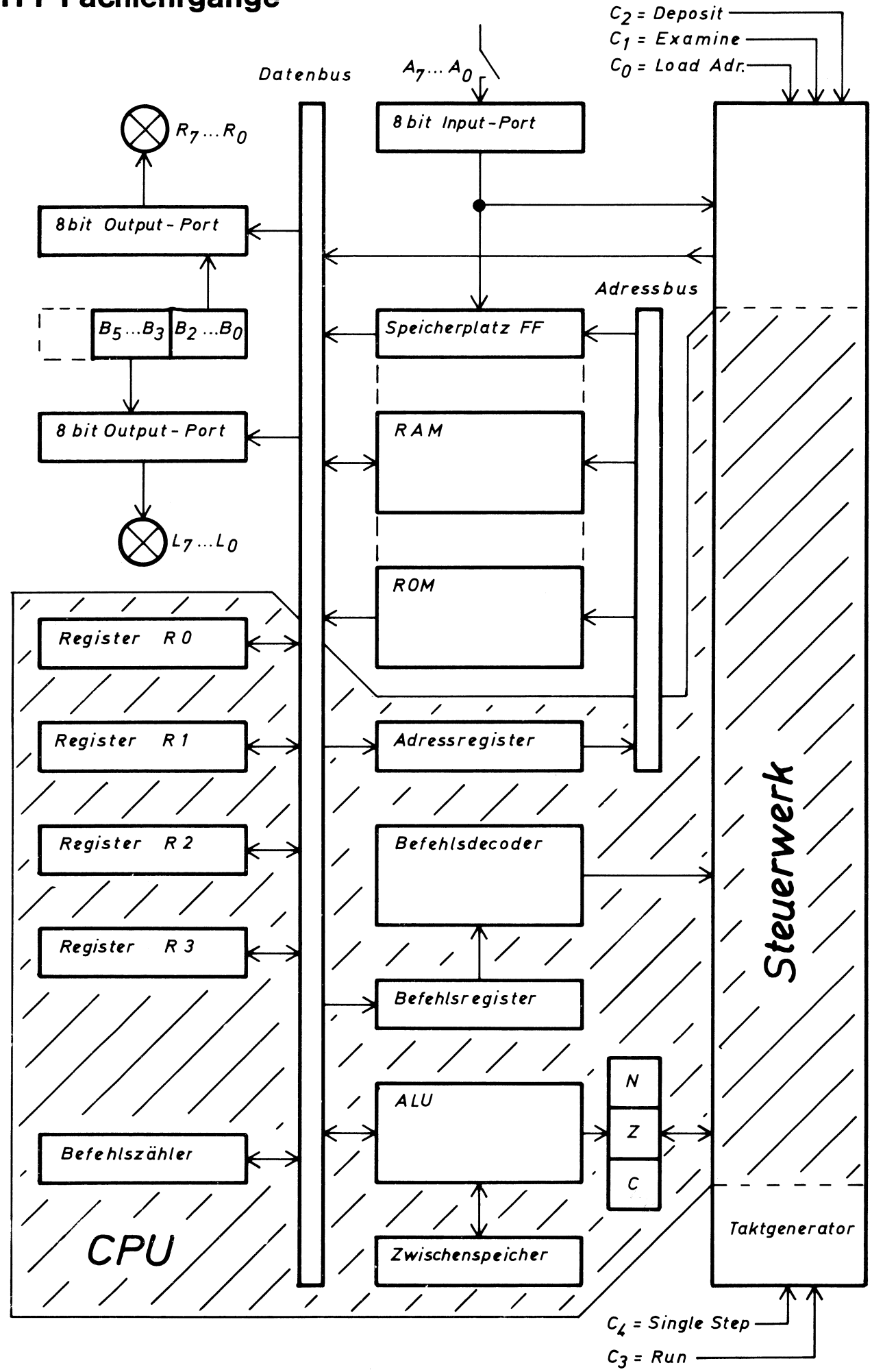
C_1 = EXAMINE auf logisch "1"

Zurücknehmen des Schalters C_1 bringt den Display-Selector wieder in Betrieb und incrementiert den Befehlszähler!

C_2 = DEPOSIT auf logisch "1"

Zurücknehmen des Schalters C_2 bringt den Display-Selector wieder in Betrieb und incrementiert den Befehlszähler!

Die Schalter RUN und SINGLE STEP behalten die gleichen Funktionen wie bereits im System 4!


C2 = Deposit
C1 = Examine
C0 = Load Adr.
Datenbus
A7 ... A0
8 bit Input-Port
R7 ... R0
8bit Output-Port
Adressbus
B5 ... B3
B2 ... B0
Speicherplatz FF
8 bit Output-Port
RAM
L7 ... L0
ROM
Register R0
Register R1
Adressregister
Register R2
Befehlsdecoder
Register R3
Befehlsregister
N
ALU
Z
Befehlszähler
C
CPU
Zwischenspeicher
Steuerwerk
Taktgenerator
C4 = Single Step
C3 = Run

Adress-Modes (Adressierungsarten) im System 5

An dieser Stelle sollen nur die wichtigsten Adressierungsarten, die in den weitern Aufgaben und Beispielen Verwendung finden, erläutert werden.

Eine Adresse ist üblicherweise ein numerischer Ausdruck, der einen bestimmten Speicherplatz bezeichnet. Die Angabe eines Registers ist jedoch auch eine Art von Adressierung, bei der eine 2 bit-Zahl als Adresse verwendet wird. Mit diesen 2 bit können die vier Register im System 5 "adressiert" werden.

a) Register-Adressierung

Diese Adressierungart liegt vor, wenn ausschließlich mit Registerinhalten gearbeitet wird. Da ein Teil der Befehle mit zwei Registern arbeitet, muß außerdem noch festgelegt werden, wo das Ergebnis der Operation erscheinen soll. Daher wird das eine Register mit

ss = Source = Datenquelle

und das zweite, in dem das Ergebnis erscheinen soll, mit

dd = Destination = Datenziel

bezeichnet. Diese Art der Adressierung ist für das Steuerwerk des Rechners aus dem Befehl selbst erkennbar.

b) Immediate Adressierung

Hier wird mit den beiden Mode-Bits mm dem Rechner gesagt, daß im nächsten Byte ein Operand steht, mit dem der Befehl ausgeführt werden soll. Diese Art der Adressierung kann z.B. eingesetzt werden, wenn in einem Programm Konstanten verarbeitet werden sollen.

c) Direkte oder absolute Adressierung

Mit den beiden Mode-Bits wird hier angezeigt, daß im nächsten Byte eine Adresse (im Sinne einer Speicherplatzangabe) steht. Das kann entweder die Adresse eines Operanden sein oder eine Zieladresse für Sprungbefehle.

Zusammenfassung der Adressierungsarten:

Register-Adr. : alle Operanden in Registern

Immediate-Adr.: ein Operand im 2.Byte

Direkt-Adr. : Adresse im 2.Byte

Befehlssatz des System 5

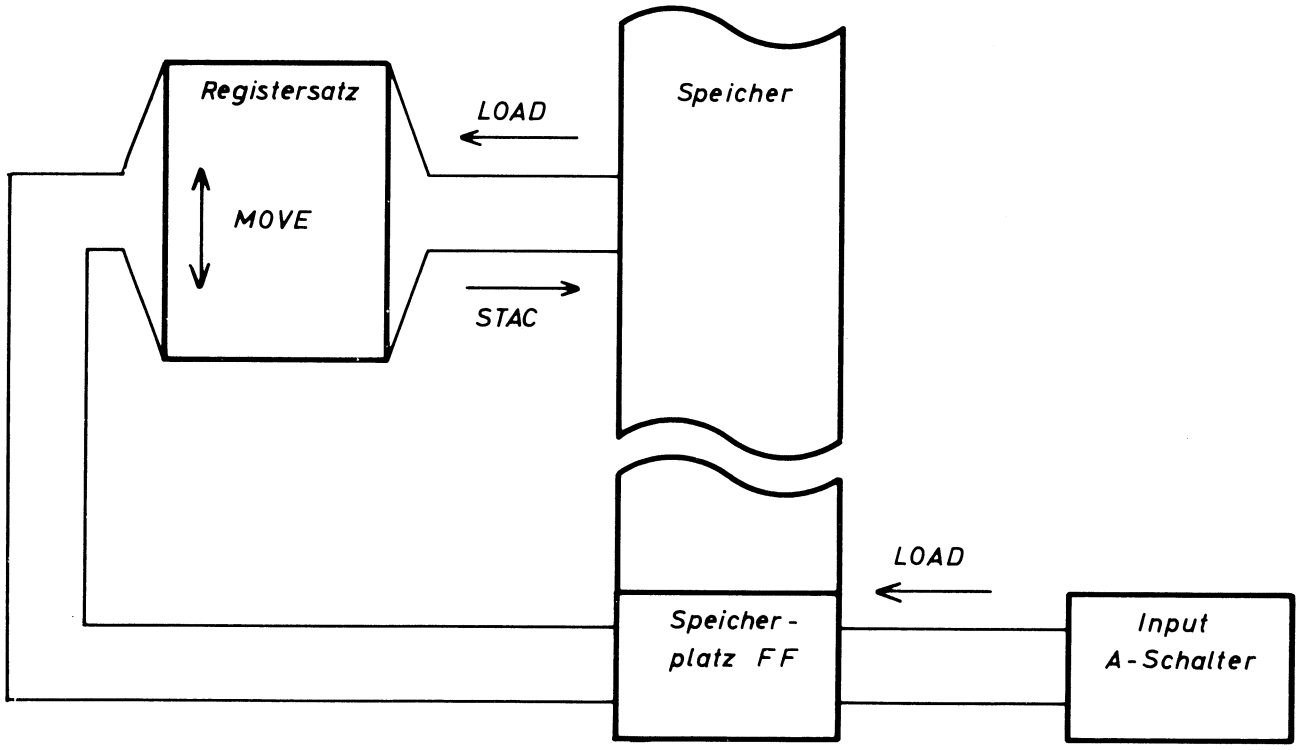
Die Befehle eines Rechnersystems lassen sich auf verschiedene Arten zu Befehlsgruppen zusammenfassen. Es sind jedoch nicht alle Befehle dieses Systems in der folgenden Darstellung enthalten, da zum Verständnis aller Vorgänge noch einige zusätzliche Kenntnisse erforderlich wären.

In der folgenden Gruppierung wurden die Befehle nach ihrer Funktion zusammengefasst:

a) Daten-Transfer
b) Arithmetik/Logik
c) Rotation
d) Programmablauf

zu a) Daten-Transfer-Befehle

In der folgenden Blockschaltung sind die im System 5 möglichen Daten-Transfer-Wege dargestellt.



1. Transfer:

   Befehl :

   Adr.Mode:

2. Transfer:

   Befehl :

   Adr.Mode:

3. Transfer:

   Befehl :

   Adr.Mode:

4. Transfer:

   Befehl :

   Adr.Mode:

Die mnemonische Schreibweise dieser Befehle wurde an den MP 8080 angelehnt. Der Befehl

MOVE R3,R1

bewirkt daher einen Daten-Transfer von Register R1 nach Register R3. Die Transfers werden in der mnemonischen Schreibweise also von rechts nach links ausgeführt.

zu b) Arithmetik/Logik-Befehle

In dieser Befehlsgruppe laufen die Befehle nach folgendem Schema ab(Ausnahmen bilden INCR und DECR):

$$(\text{Destination}) \left\{ \begin{matrix} + \\ - \\ \wedge \\ \vee \\ \not\vee \end{matrix} \right\} (\text{Source}) \longrightarrow \text{Destination}$$

Destination muß immer ein Register sein. Für Source lassen sich zwei Fälle unterscheiden:

1.) Source = Register
Alle Befehle bei denen auch der zweite Operand in einem Register steht, werden durch ein angehängtes R gekennzeichnet.(z.B. ADDR, XORR usw.)

2.) Source = Speicherplatz
Der zweite Operand steht jetzt im Speicher, das bedeutet:
bei Immediate-Adr.: im nächsten Byte
bei direkter Adr. : in der Adresse die im nächsten Byte steht
Diese Befehle werden mit einem angehängten M(Memory) gekennzeichnet.

Sonderfälle

Falls die Inhalte von Source und Destination identisch sind, ergeben sich für die einzelnen Befehle folgende Wirkungen:

ADD :

SUB :

IOR :

XOR :

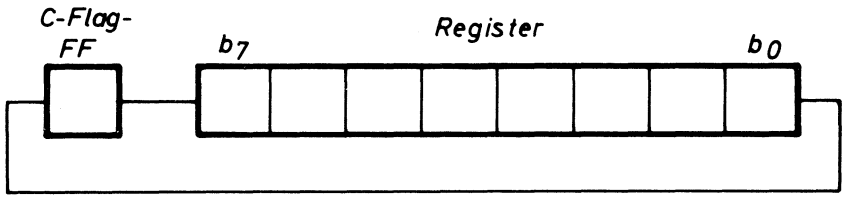
AND :

Increment und Decrement

Bei diesen beiden Befehlen gibt es nur einen Operanden. Er wird in der Befehlskarte mit ss bezeichnet. Somit kommen nur die vier Register als Source in Frage. Sie sind jedoch gleichzeitig auch Destination, da das Ergebnis jeweils im selben Register erscheint.

zu c) Rotation

Die Rotationsbefehle lassen sich am einfachsten an Hand einer Funktionsskizze erläutern.



Das jeweilige Source-Register kann zusammen mit dem C-Flag als 9 bit-Ringschieberegister betrachtet werden. Ein Rotationsbefehl bewirkt eine Verschiebung um eine Position(ein bit) nach rechts bzw. links.

zu d) Programmablauf-Befehle

Erscheint in einem Programm ein HALT-Befehl, wird die Programmbearbeitung an dieser Stelle gestoppt. Der Befehlszähler wird nichtmehr incrementiert, d.h. er enthält die Adresse, in der der HALT-Befehl steht. Zum Verlsssen der HALT-Position gibt es zwei Möglichkeiten:

1.) Ein Takten des Examine-Schalters erhöht den Befehlszähler um eins. Damit würde das Programm mit dem auf HALT folgenden Befehl fortgesetzt.
2.) Mit Load Adr. kann da Programm bei jeder beliebigen Adresse fortgesetzt, bzw. neu begonnen werden.

Der NOP-Befehl kann eingesetzt werden, wenn sich in einem Programm Befehle als überflüssig herausstellen, das ganze Programm jedoch nicht neu geschrieben werden soll. Der Einsatz als Zeitverzögerung ist ebenfalls möglich.

Die JUMP-Befehle sollen zunächst nur mit direkter Adressierung eingesetzt werden. Damit ergeben sich Zwei-Byte-Befehle, da im 2.Byte jeweils die Adresse des Sprungzieles angegeben werden muss. Im System 5 sind folgende Sprünge möglich:

1.) unbedingter Sprung:

JUMP

2.) bedingte Sprünge:

JMPZ	Springe	wenn	Z-Flag	=	1
JMPN	"	"	N- "	=	1
JMPC	"	"	C- "	=	1

Programmierhilfen im System 5

Die folgenden beiden Blätter sollen das Übersetzen von Programmen in die Maschinensprache des hypothetischen Rechners erleichtern.

Anmerkungen zu den einzelnen Befehlstafeln:

Blatt 1 Tafel 1

Da im System 5 vier Register zur Verfügung stehen, sind für die MOVE-Befehle 16 Kombinationen möglich. Wird als Source und Destination dasselbe Register verwendet, so entspricht die Wirkung eines solchen Befehls dem NOP. Solche wirkungslosen Befehle können im Befehlsdecoder erkannt und mit anderen Bedeutungen belegt werden.

Blatt 1 Tafel 2

Einen Sonderfall der LOAD-Befehle stellen im System 5 die Eingabebefehle von den A-Schaltern dar. Da den A-Schaltern die Geräteadresse FF zugeordnet ist, müssen Inputs mit einem direkt adressierten LOAD-Befehl durchgeführt werden.

Blatt 1 Tafel 3

Beim STAC-Befehl muß (im Gegensatz zum MP 8080) zwischen vier möglichen Akkumulatoren als Source unterschieden werden. Wenn die vereinbarte Schreibweise (Datentransfers von rechts nach links) beibehalten werden soll, ergibt sich für diese Befehle die folgende Darstellung:

STAC 7B,R2

Mit diesem Befehl würde der Inhalt von Register R2 in die Adresse 7B gebracht.

Blatt 2 Tafeln 4 und 5

Der aus zwei Hexadezimalziffern bestehende Maschinencode wird hier aus einer Hexziffer für den eigentlichen OP-Code und einer zweiten Hexziffer für die Definition von Source und Destination zusammengesetzt.

Die in Tafel 4 enthaltenen Befehle sind 1 Byte-Befehle, während in Tafel 5 bedingt durch die Adressierungsarten 2 Byte-Befehle entstehen.

Blatt 2 Tafel 6

Hier sind die restlichen Befehle, die in den Programmbeispielen eingesetzt werden, aufgeführt.

Datentransfer Register ⟶ Register (1)

Move to register from register

MOVE R_,R_

Destin. ↑

Source ↑

DESTINATION \ SOURCE	R 0	R 1	R 2	R 3
R 0				
R 1				
R 2				
R 3				

Datentransfer Speicher ⟶ Register (2)

Load to register from memory

LOAD R_, #data

" R_, adr.

" R_,

R_,

Destin. ↑

DESTINATION \					"INPUT" (A-Sch.)
R 0					
R 1					
R 2					
R 3					

Datentransfer Register ⟶ Speicher (3)

Store to memory from register (accu)

STAC ,R_

" adr. ,R_

" ,R_

" ,R_

Source ↑

	SOURCE R 0	R 1	R 2	R3

Arithmetik / Logik Register - Register (4)

ADDR = 1_
SUBR = 2_
IORR = 3_
XORR = 4_
ANDR = 5_

DESTINATION \ SOURCE	R0	R1	R2	R3
R0	_0	_4	_8	_C
R1	_1	_5	_9	_D
R2	_2	_6	_A	_E
R3	_3	_7	_B	_F

Arithmetik / Logik Register - Speicher (5)

ADDM = 9_
SUBM = A_
IORM = B_
XORM = C_
ANDM = D_

DESTINATION \	# (2. Byte)	direkt ((2. Byte))		
R0	_0	_4		
R1	_1	_5		
R2	_2	_6		
R3	_3	_7		

(6)

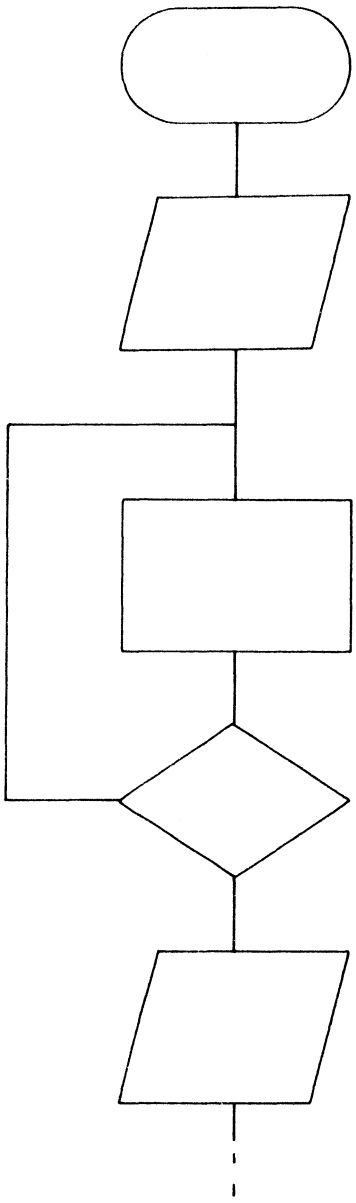
	SOURCE R0	R1	R2	R3
INCR	60	61	62	63
DECR	64	65	66	67
RACL	68	69	6A	6B
RACR	6C	6D	6E	6F

JUMP = E0
JMPZ = E1
JMPN = E2
JMPC = E3

Vorwahlzähler

Unter dem Begriff Vorwahlzähler sind Zähler zu verstehen, die nach dem Eintreffen einer vorwählbaren Anzahl von Impulsen ein Signal abgeben oder einen durch das jeweilige Programm bestimmten Vorgang auslösen. Das Eintreffen externer Impulse kann jedoch auch durch wiederholte Increment- oder Decrement-Befehle ersetzt werden.
Zur Realisierung solcher Zählprogramme bestehen folgende Möglichkeiten:

a) Verwendung des Decrementbefehls

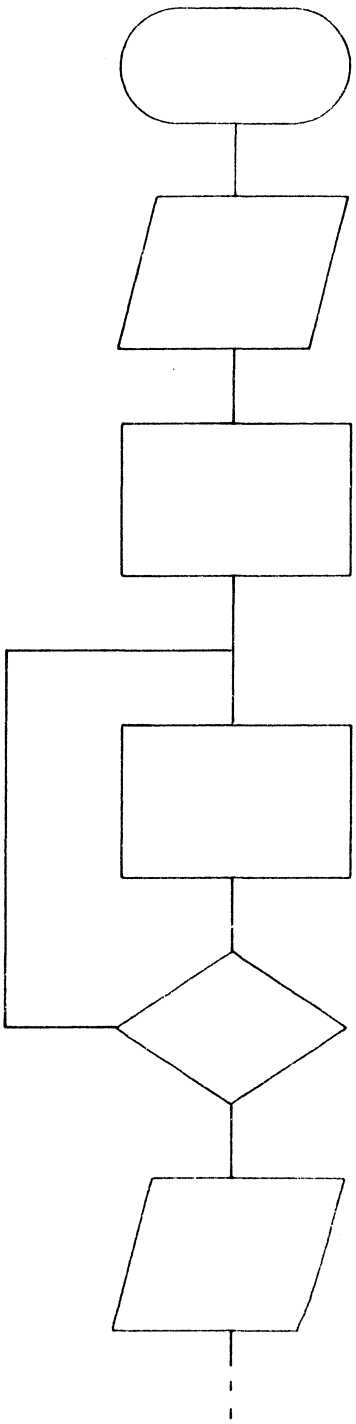


Die vorzuwählende Zahl wird in ein Register geladen. Durch Prüfen des Registers auf den Inhalt 00 kann eine bestimmte Anzahl an Decrement-Befehlen durchgeführt werden.
Diese Art des Programmaufbaus ist am einfachsten zu realisieren, ebenso wie sich bei diesem Prinzip auch der einfachste Hardware-Aufbau ergibt.

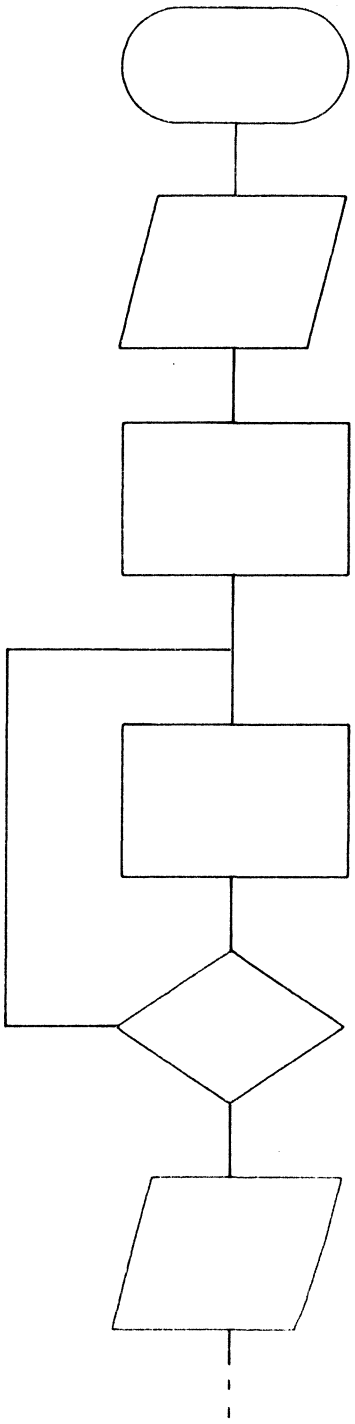
b) Verwendung des Increment-Befehls

Soll ein Vorwahlzählerprogramm mit Increment-Befehlen realisiert werden, so läßt sich das auf zwei Arten durchführen:

1.) Die vorzuwählende Zahl wird als negative Zahl in ein Register gebracht und dann bis 00 incrementiert.

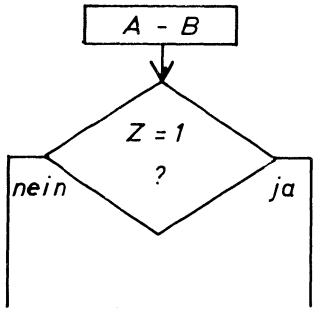


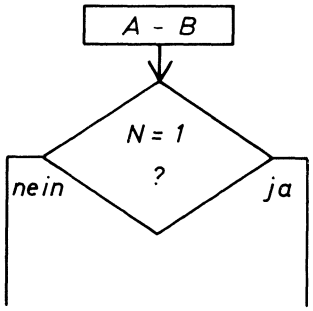
2.) Das zählende Register beginnt mit dem Zustand 00. Das Erreichen der Vorwahl wird durch Vergleich bestimmt.

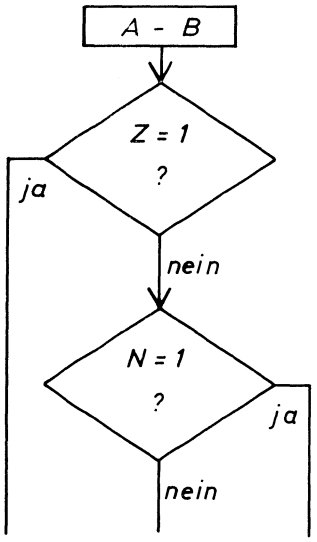


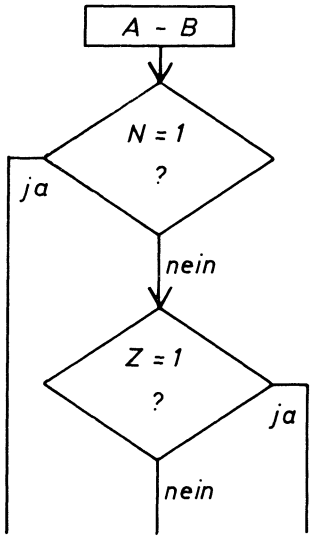
ITT Fachlehrgänge

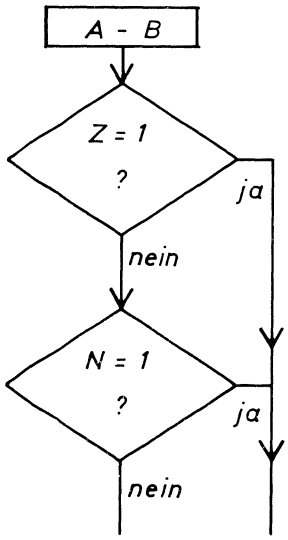
Flußdiagramme für arithmetische Vergleiche (Komparatoren)











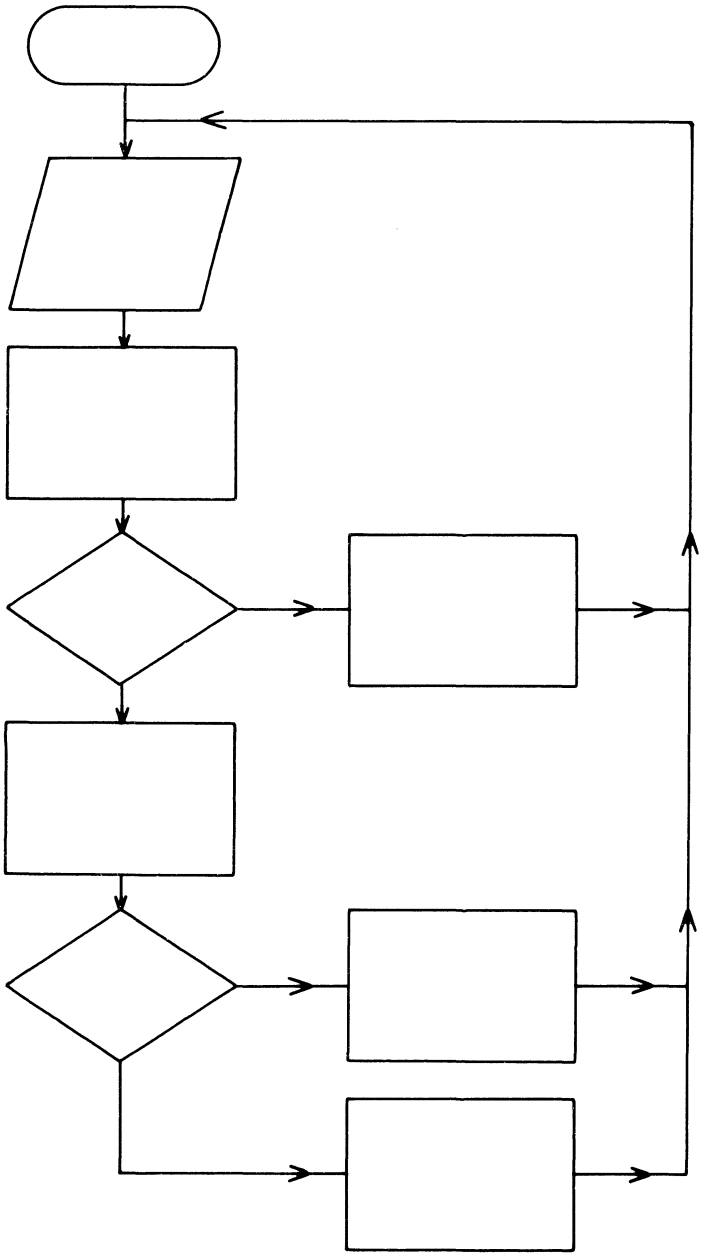
Programm für eine Grenzwertüberwachung (Beispiel Netzspannung)

Von einem A/D-Wandler (simuliert durch die A-Schalter) soll der IST-Wert der Netzspannung an den Rechner gegeben werden. Als Grenzwerte gelten:

MIN = 190V = BE_{16}

MAX = 240V = FO_{16}

Die Überwachung soll ständig erfolgen und den richtigen Wert bzw. die Überschreitungen durch Signale anzeigen.



ADRESSE	INHALT	BEFEHL	;KOMMENTAR
	40	86	LOAD R2,FF
	41	FF	
	42	09	MOVE R1,R2
	43	A1	SUBM R1,# BE
	44	BE	
	45	E2	JMPN 54
	46	54	
	47	09	MOVE R1,R2
	48	A1	SUBM R1,# F1
	49	F1	
	4A	E2	JMPN 50
	4B	50	
	4C	87	LOAD R3,81
	4D	81	
	4E	E0	JUMP 40
	4F	40	
	50	87	LOAD R3,82
	51	82	
	52	E0	JUMP 40
	53	40	
	54	87	LOAD R3,83
	55	83	
	56	E0	JUMP 40
	57	40	
	81	C0	
	82	3C	
	83	03	