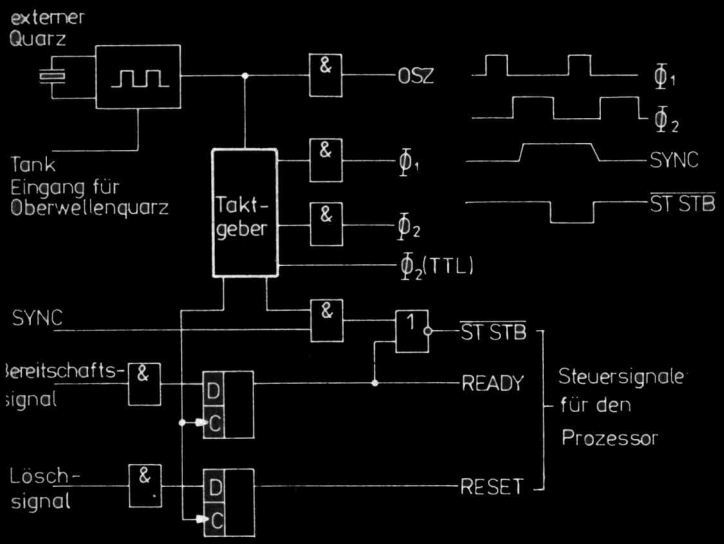
amateurreihe

electronica



Barthold/Bäurich

Mikroprozessoren – Mikroelektronische Schaltkreise und ihre Anwendung (Teil 1)

222/223

electronica · Band 222/223

HANS BARTHOLD
DR. HEINZ BÄURICH

Mikroprozessoren-Mikroelektronische Schaltkreise und ihre Anwendung

Teil 1:
Grundlagen der Mikrorechentechnik

MILITÄRVERLAG
DER DEUTSCHEN DEMOKRATISCHEN
REPUBLIK

Barthold , H.; Dr. Bäurich, H.:
Mikroprozessoren – Mikroelektronische Schaltkreise und ihre Anwendung.
Teil 1: Grundlagen der Mikrorechentechnik. –
3., überarbeitete Aufl., – Berlin:
Militärverlag der DDR (VEB), 1985. –
184 S.: 108 Bilder – (electronica: 222/223

3. Auflage, 1985

Lizenz-Nr. 5
Printed in the German Democratic Rebublic
Gesamtherstellung: Druckerei Märkische Volksstimme Potsdam
Lektor: Steffen Würtenberger
Zeichnungen: Johanna Goernemann
Typografie: Martina Schwarz
Redaktionsschluß: 20. September 1984
LSV 3539
Bestellnummer: 746 692 4
00380

Inhaltsverzeichnis

Vorwort

Der Mikroprozessor ist sehr schnell zu einem Begriff geworden, der aus der Elektronik nicht mehr wegzudenken ist. In der Fachliteratur spricht man in diesem Zusammenhang von Bausteinen der künftigen Automatisierungstechnik. Selbst solche Aussagen wie „Nervensystem der Technik" werden im Zusammenhang mit dem Mikroprozessor gebraucht. Unsere Massenmedien berichten in Wort und Bild, daß durch den Mikroprozessor ein Teil der Arbeitswelt völlig verändert wird. Während in den kapitalistischen Ländern die Rationalisierung mit Hilfe dieser Elektronik dazu führt, daß Arbeitsplätze abgeschafft werden und dadurch die Arbeitslosigkeit weiter steigt, dient der Einsatz der Mikrorechentechnik in den sozialistischen Ländern zum Nutzen der gesamten Gesellschaft und somit dem Wohl jedes einzelnen.

Aus allen diesen Informationen heraus drängen sich die Fragen auf, was ist ein Mikroprozessor, was kann er wirklich leisten? Das gesamte Gebiete der Elektronik hat sich seit 1945 sehr rasch entwickelt. Eine wesentliche Rolle dabei spielt die Entwicklung der Bauelemente. Von den anfänglichen Bauelementen der Informationstechnik bis zu den hochintegrierten Schaltkreisen haben sich Platz- und Energiebedarf für ein aktives Bauelement sehr stark verringert, gleichzeitig sind Zuverlässigkeit und Betriebssicherheit in sehr hohem Maße angestiegen.

Alle Gebiete der Elektronik wurden von dieser schnellen Bauelementeentwicklung beeinflußt. Dabei ist auf dem Gebiet der elektronischen Rechentechnik diese Entwicklung besonders sichtbar. Aus diesem Fachgebiet kommen auch die programmierbaren Schaltkreise, wobei der bekannteste Vertreter der Mikroprozessor ist. Er stellt den eigentlichen Rechenschaltkreis dar; mit ihm lassen sich alle Verknüpfungen von Zahlen und logischen Größen realisieren.

Die Verknüpfung solcher Größen bildet gleichzeitig die Grundlage der Automatisierungstechnik. Im Grunde genommen können mit jedem elektronischen Rechner die Probleme der Automatisierung gelöst werden. Aber erst durch die Möglichkeit, den Rechner als Bauelement (Chip) herzustellen, wurde die stürmische Entwick-

lung in der Automatisierungstechnik erreicht. Es läßt sich einschätzen, daß diese Entwicklung noch am Anfang steht. Aber schon in den nächsten Jahren wird der Mikroprozessor in vielen technischen Geräten eine Selbstverständlichkeit sein.
Auf Grund der großen Nachfrage erscheinen die Hefte „Mikroprozessoren – Mikroelektronische Schaltkreise und ihre Anwendung" in einer überarbeiteten Form als zwei Doppelhefte. Sie sind als Einheit zu betrachten. Bei der Überarbeitung wurde das Kapitel über den Mikroprozessor *U 808* herausgenommen, da er für Neuentwicklungen nicht mehr eingesetzt wird. Hinzugekommen sind ein Kapitel über die Assemblersprache MAPS-K 1520 sowie ein Kapitel mit Programmbeispielen zum *U 880*.

1. Erläuterungen zu den Abkürzungen

1.1. Aufstellung häufig verwendeter Formelzeichen und Abkürzungen

B	Zahlenbasis
E	Exponent einer Gleitkommazahl
H	hoher logischer Spannungspegel
L	niedriger logischer Spannungspegel
m	Mantisse einer Gleitkommazahl
P_j	Elementarkonjunktion, die der Dualzahl j zugeordnet ist
T	logischer Term (logischer Ausdruck)
Q, $\overline{Q}$	Ausgänge eines Flip-Flop
V	Vorzeichen einer Zahl
X_i	Ziffern einer Zahl
$X(Y)$	Wert einer Zahl
X'	Zweierkomplement der Zahl X

1.2. Wertzuweisung bei logischen Signalen

In den verwendeten Schaltbildern sind die logischen Signale durch Abkürzungen eingezeichnet. Die Abkürzungen werden im Text erläutert. Ist am Eingang oder am Ausgang eines Bausteins ein logisches Signal S durch S gekennzeichnet, heißt das: Das Signal hat Hochpegel, wenn dem Signal der logische Wert 1 zugeordnet ist. Wird es durch $\overline{S}$ gekennzeichnet, heißt das: Das Signal hat Tiefpegel, wenn dem Signal der logische Wert 1 zugeordnet ist. In den Funktionstabellen bedeuten die Bezeichnungen H bzw. L hoher bzw. niedriger Spannungspegel. Dagegen kennzeichnen 1 oder 0 die den Eingängen bzw. Ausgängen zugeordneten logischen Werte. Die Bezeichnung H und L wurde in den Fällen gewählt, wo es sich um die Beschreibung der Signalpegel in einer Schaltung handelt. 0 und 1 stehen in den Fällen, wo aus vorgegebenen logischen Bedingungen das logische Schaltbild entworfen wird, ohne daß dabei die Signalzuordnung H oder L erforderlich ist.

2. Grundlagen der Rechentechnik

2.1. Aufbau eines Rechners

Der Rechner war das Vorbild für die Entwicklung der Mikroprozessoren. Viele Vorgänge in Natur und Technik laufen nach mathematischen oder logischen Regeln ab. Der Rechner ist die Basis zur Nachbildung solcher Vorgänge. Durch die Möglichkeit, ihn zu programmieren, kann man in ihm Algorithmen speichern und zu jeder Zeit abarbeiten lassen.
Der Rechner ist ein universelles Hilfsmittel zur Realisierung von Steuerungen, zur Nachbildung von Modellen sowie zur Lösung von mathematischen Aufgaben, für die man sich bisher umfangreicher elektronischer Schaltungen bedienen mußte (z. B. Digitaluhr, Digitalvoltmeter, Zähler). Durch seine Programmierbarkeit hat er gegenüber anderen Lösungen den Vorteil, daß er sich ohne Änderung der *Hardware* (fest verdrahtete Schaltung) an das jeweilige Programm anpassen läßt. Sein Einsatz für eine bestimmte Lösung hängt im Prinzip nur vom Kostenverhältnis zwischen Rechner und konkreter Schaltungstechnik ab.
Ein Rechner soll eine Aufgabe nach einer Lösungsvorschrift abarbeiten. Dabei ist der Ablauf derselbe wie bei einer Handrechnung. Entsprechend der Lösungsvorschrift, die aus einer Reihe von Anweisungen besteht, werden die Daten durch Rechenoperationen miteinander verknüpft, eventuelle Zwischenergebnisse notiert und die Ergebnisse auf einem gesonderten Formular zusammengefaßt.
In der gleichen Weise arbeitet ein Rechner. Die Lösungsvorschrift, die aus einer Reihe von Anweisungen (*Befehlen*) besteht, ist das *Programm*. Man muß es, damit es abgearbeitet werden kann, in einen Speicher eingeben. Ebenso müssen die Ausgangsdaten, die Zwischenwerte sowie die Resultate gespeichert werden. Das *Rechenwerk,* das im wesentlichen die 4 Grundrechenarten Addition, Subtraktion, Multiplikation und Division sowie die logischen Operationen UND, ODER und NEGATION ausführt, verknüpft die Daten. Für die Steuerung des gesamten Ablaufs gibt es ein *Steuerwerk.* Dieses Steuerwerk liest Anweisung für Anweisung der

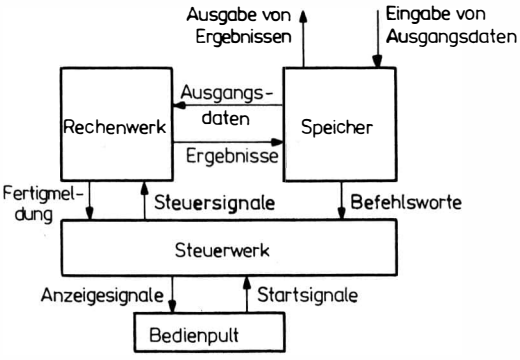


Bild 2.1 Grundsätzliche Struktur eines Rechners

Lösungsvorschrift aus dem Speicher und gibt dem-Rechenwerk Signale zur Ausführung der in der Anweisung vorgegebenen Funktion. Rechenwerk und Steuerwerk bilden die CPU (*C*entral *P*rozessor *U*nit). Ein Rechner besteht also aus den Hauptbestandteilen Speicher, Rechenwerk und Steuerwerk.
Außerdem gehören zum Rechner noch Ein- und Ausgabegeräte zur Eingabe des Programms und der Ausgangsdaten einer Aufgabe sowie zur Ausgabe der Ergebnisse. Ferner ist an jedem Rechner ein Bedienpult angeschlossen, über das bestimmte Funktionen (Starten eines Programms, Ein- und Ausschalten von Teilgeräten, Sichtanzeigen) gesteuert werden können.
Bild 2.1 zeigt die grundsätzliche Struktur eines solchen Rechners.

2.1.1. Speicher

Der Speicher hat die Aufgabe, alle Informationen, die zur Lösung einer Aufgabe benötigt werden, zu speichern.
Dazu gehören das Programm, das aus einer Folge von Befehlen besteht, die Ausgangsdaten, die Zwischenresultate und die Endergebnisse.
Ausgangsdaten, Zwischenresultate und Endergebnisse sind im allgemeinen Zahlen oder alphanumerische Zeichen (Text). Zur Darstellung einer Zahl oder eines Befehls dient innerhalb des Rechners ein sogenanntes *Maschinenwort,* das aus einer Bit-Folge

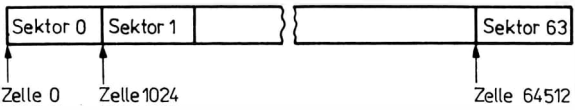


Bild 2.2 Logische Aufteilung eines 64-K-Speichers in 64 Sektoren

besteht. Es hat eine vorgegebene Wortlänge. Den Platz, der notwendig ist, um ein solches Wort im Rechner zu speichern, nennt man eine *Speicherzelle*. Der Speicher besteht aus einer größeren Anzahl solcher Speicherzellen, die durchnumeriert sind. Die Nummer der Speicherzelle nennt man die *Adresse*.
In der Rechentechnik ist die Mengenangabe Kilo (Kilobyte oder Kiloworte) üblich. Ein Kilo kennzeichnet hier jedoch nicht die tausendfache Menge, sondern das 1024fache (1 Kbyte = 1024 Byte), da die Kapazität eines Speichers fast immer einer Zweierpotenz entspricht. Zu einer solchen Speicherplatzanzahl ist eine optimale Adreßentschlüsselung möglich. Manchmal wird der Speicher in Seiten oder Sektoren aufgeteilt. Entsprechend unterteilt sich dann auch die Adresse in Seitenadresse (Sektoradresse) und Zellennummer innerhalb der Seite (Sektor). Bild 2.2. zeigt die Aufteilung eines Speichers mit 64K-Zellen in 64 Sektoren.
Technisch werden Speicher meistens als Ferritkernspeicher oder Halbleiter ausgeführt (s. Abschnitt 3.4.3.).

2.1.2. Rechenwerk

Das Rechenwerk dient zur Ausführung von Rechenoperationen. Es bekommt durch das Steuerwerk eine Folge von Schaltsignalen, die es auf die gerade auszuführenden Rechenoperationen umschalten. Vom Speicher erhält es die Zahlen, die zur Ausführung der Rechenoperationen notwendig sind.
Die Zahlen werden im Rechenwerk zwischengespeichert. Dazu hat das Rechenwerk mehrere Register, in denen die Ausgangsdaten einer Rechenoperation gespeichert sind.
Nach der Ausführung der Rechenoperation wird das Ergebnis ebenfalls in einem Register gespeichert. Bild 2.3. zeigt das Grundprinzip eines Rechenwerks.

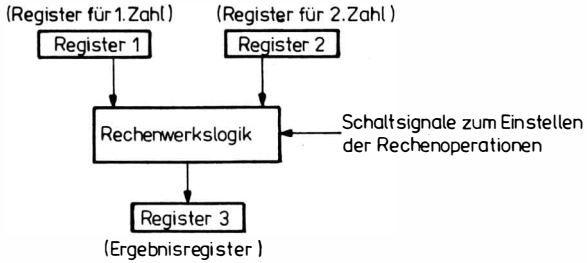


Bild 2.3 Grundprinzip eines Rechenwerks

Zu den Operationen, die ein Rechenwerk eines Rechners ausführt, gehören:

a) Arithmetische Rechenoperationen für die im Rechner verdrahteten Zahlendarstellungen (ADDITION, SUBTRAKTION, MULTIPLIKATION und DIVISION). Bei Mikrorechnern sind im allgemeinen nur Addition und Subtraktion möglich. Multiplikation und Division müssen programmiert werden.

b) Logische Operationen
(UND, ODER, EXKLUSIV-ODER,
NEGATION, VERSCHIEBUNG)

c) Zahlenumwandlungen.

Das Rechenwerk eines Mikrorechners wird im allgemeinen als *arithmetisch-logische Einheit* (ALU – *a*rithmetic-*l*ogic *u*nit) bezeichnet. Die ALU ist eine Logikschaltung, die durch Steuersignale so eingestellt werden kann, daß ein oder zwei Eingangsbitmuster entsprechend der eingestellten Operation verarbeitet werden.

2.1.3. Steuerwerk

Das Steuerwerk übernimmt die Befehle eines Programms in der vorgegebenen Reihenfolge aus dem Speicher, entschlüsselt sie und bildet daraus die Steuersignale für das Rechenwerk. Bild 2.4 zeigt den logischen Aufbau des Steuerwerks.
Der Befehlszähler enthält die Adresse des zu holenden Befehls. Der vom Speicher übernommene Befehl wird zur Befehlsentschlüsselung im Befehlsregister zwischengespeichert. Das Spei-

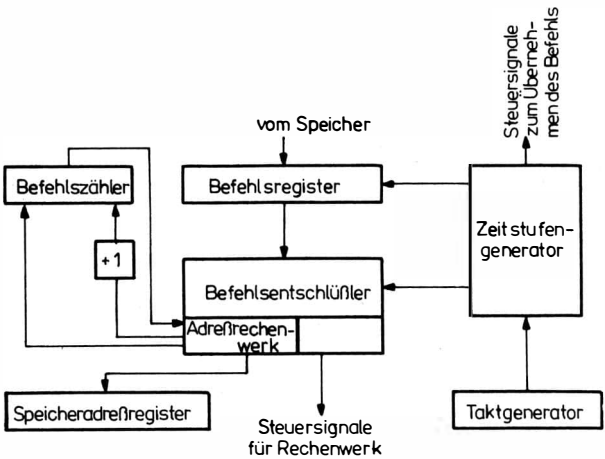


Bild 2.4 Aufbau des Steuerwerks eines Rechners

cheradreßregister enthält die Adresse des zum Befehl benötigten Operanden. (Mehrere Operanden [Zahlen] werden nacheinander aus dem Speicher geholt.)

Der Zeitstufengenerator erzeugt eine Reihe von Zeitsignalen, die den Ablauf der Befehlsabarbeitung festlegen. Den Grundtakt dazu liefert der Taktgenerator. Die wichtigsten Schritte einer Befehlsabarbeitung sind:

- Übernehmen des Befehls aus dem Speicher;
- Entschlüsseln des Befehls;
- Ermitteln der Operandenadresse;
- Holen der Operanden aus dem Speicher;
- Ausführung des Befehls;
- Ermittlung der Adresse des nächsten Befehls.

Da der Rechner aus Speicher, Rechen- und Steuerwerk besteht, ergibt sich die Möglichkeit, automatisch, d. h. programmgesteuert, beliebige Funktionen oder Aufgaben abzuarbeiten. Zu diesem Zweck enthält jede Aufgabe 2 Teilinformationen. Die erste Teilinformation beinhaltet die Daten bzw. Zahlen, die verarbeitet werden sollen. Der zweite Teil der Information sind die Anweisungen (Befehle), die aussagen, wie die Daten verarbeitet werden sollen. Beide Teilinformationen gibt man über externe Geräte (Eingabetastatur, Schreibmaschine, Lochbandleser, Lochkarteneingabe

usw.) in den Speicher ein. Vom Speicher werden die Anweisungen in einer festgelegten Reihenfolge ins Steuerwerk übernommen, entschlüsselt und vom Rechenwerk abgearbeitet. Zwischen- und Endresultate gelangen in den Speicher zurück und können von dort aus wieder über externe Geräte (Anzeige, Schreibmaschine, Drucker) ausgegeben werden.

2.2. Darstellung von Daten

Da der Speicher aus einzelnen Zellen besteht, wobei jede Zelle eine Bit-Folge mit vorgegebener Länge speichern kann, müssen die zu verarbeitenden Daten (Zahlen) in Bit-Folgen umgewandelt werden. Dafür gibt es eine Reihe von Darstellungsarten. Je nach Länge einer Speicherplatzzelle und Anzahl der notwendigen Stellen für die Darstellung einer Zahl werden für die Speicherung einer Zahl eine oder mehrere Zellen benötigt. Das gleiche gilt für die Speicherung von alphanumerischem Text.

2.2.1. Zahlendarstellung

Ausgangspunkt für die Darstellung von Zahlen in Rechnern ist das *Dualsystem.* Dabei handelt es sich um ein Zahlensystem, das auf nur 2 Ziffern basiert (0 und 1). Zur Speicherung einer Dualziffer sind also nur 2 stabile Zustände notwendig, die sich leicht realisieren lassen.

2.2.1.1. Ganze Zahlen im Dualsystem

Um eine Dezimalzahl in das Dualsystem zu übertragen, zerlegt man sie in Potenzen zur Basis 2, wobei jeder neuen Potenz von 2 eine neue Stelle zugeordnet wird.
Beispiel
Die Zahl 27 soll dual dargestellt werden. Aus Tabelle 2.1. ist zu ersehen, daß die größte in 27 enthaltene Zweierpotenz $2^4 = 16$ ist, d. h., der Stellenwert 2^4 wird benötigt. Der Stellenwert $2^3 = 8$ ergibt mit 2^4 zusammen 24, der Rest ist 3. Der Stellenwert 2^2 tritt also nicht, d. h. 0mal, auf, während die Stellenwerte 2^1 und 2^0 je einmal

Tabelle 2.1. Tabelle der Zweierpotenzen

2^n	n	2^{-n}
1	0	1
2	1	0,5
4	2	0,25
8	3	0,125
16	4	0,0625
32	5	0,03125
64	6	0,015625
128	7	0,0078125
256	8	0,00390625

erforderlich sind. Die Zahl 27 stellt sich im Dualsystem also folgendermaßen dar:

Stelle 2^4	Stelle 2^3	Stelle 2^2	Stelle 2^1	Stelle 2^0
1	1	0	1	1

Der Leser führe selbst die Umwandlung der folgenden Dezimalzahlen ins Dualsystem aus:

$41 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \triangleq 101001$ (dual);

$15 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \triangleq 1111$ (dual).

Umwandlung Dezimal → Dual

Eine weitere Methode besteht in der fortlaufenden Division durch 2.

Beispiel

Es soll die Dezimalzahl 93 durch fortlaufende Division durch 2 dual dargestellt werden.

93 : 2 = 46 Rest 1
46 : 2 = 23 Rest 0
23 : 2 = 11 Rest 1
11 : 2 = 5 Rest 1
5 : 2 = 2 Rest 1
2 : 2 = 1 Rest 0
1 : 2 = 0 Rest 1

2^6	2^5	2^4	2^3	2^2	2^1	2^0
						1
					0	
				1		
			1			
		1				
	0					
1						
1	0	1	1	1	0	1

93 (dezimal) ≙ 1011101 (dual).

Umwandlung Dual →Dezimal
Hier geht man von einem Rechenschema aus, in dem die 2er Potenzen wieder dezimal dargestellt werden.

Beispiel
$10111 \triangleq 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
$= 16 + 0 + 4 + 2 + 1 = 23.$

2.2.1.2. Gebrochene Zahlen im Dualsystem

Bei der Darstellung von Zahlen, die Stellen hinter dem Komma haben, wird ähnlich wie bei den ganzen Zahlen verfahren. Man benötigt dazu eine Tabelle der Potenzen von 2 mit negativen Exponenten. Zum Beispiel kann man die Zahl 0,625 folgendermaßen schreiben:
$0{,}625 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} \triangleq 0{,}101$ (dual).
Ein Dezimalbruch läßt sich ins Dualsystem durch fortlaufende Multiplikation mit der Grundzahl 2 überführen. Steht nach der Multiplikation mit 2 vor dem Komma eine 0, so ist die nächste Dualstelle eine 0. Steht eine 1 vor dem Komma, dann ist die nächste Dualstelle eine 1.

Beispiel
Umwandlung der Dezimalzahl 0,625
$0{,}625 \cdot 2 = 1{,}250 \quad 1 \triangleq$ 1. Dualstelle nach dem Komma;
$0{,}25 \cdot 2 = 0{,}50 \quad 0 \triangleq$ 2. Dualstelle nach dem Komma;
$0{,}50 \cdot 2 = 1{,}00 \quad 1 \triangleq$ 3. Dualstelle nach dem Komma.
Die Dezimalzahl 0,625 lautet dual 0,101. Die Begründung des Verfahrens liegt darin, daß bei jeder Multiplikation mit 2 die Zweierpotenzen um 1 erhöht werden und dabei die erste Ziffer vor das Komma rückt. Diese Ziffer wird als Dualstelle übernommen.
Bei gemischten Zahlen wandelt man den ganzen Teil und den gebrochenen Teil – jeden für sich – in eine Dualzahl um.
Beispiel
Die Zahl 3,25 soll dual dargestellt werden. Die Zerlegung von 3,25 ergibt 3 + 0,25. Der Zahl 3 entspricht die Dualzahl 11, der Zahl 0,25 entspricht die Dualzahl 0,01.
Damit lautet die Zahl 3,25 dual 11,01.

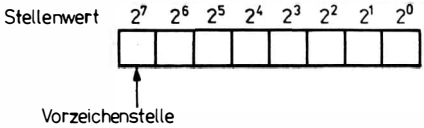


Bild 2.5
Darstellung von positiven und negativen ganzen Zahlen mit Hilfe von 8 Dualstellen

2.2.1.3. Darstellung von negativen Zahlen

In der Umgangssprache unterscheiden sich die negativen Zahlen von den positiven Zahlen durch ein negatives Vorzeichen. Für die Darstellung dieses Vorzeichens im Rechner kann man eine zusätzliche Dualstelle als Vorzeichenstelle einführen und z. B. folgende Vereinbarung treffen:
Vorzeichenstelle = 0 heißt positives Vorzeichen.
Vorzeichenstelle = 1 heißt negatives Vorzeichen.
Diese Darstellungsart wird in der Literatur mit „Betrag und Vorzeichen" bezeichnet.
In den meisten Rechnern wird jedoch eine andere Darstellung, die *Komplementdarstellung*, verwendet, in der die negativen Zahlen in den positiven Zahlenbereich transformiert werden. Der Vorteil besteht darin, daß sich die Subtraktion auf eine Addition zurückführen läßt.
In den meisten Mikroprozessoren stehen 8 Dualstellen (7 Dualstellen für die positiven Zahlen und eine Dualstelle für die Vorzeichenstelle) zur Verfügung (Bild 2.5). Werden diese 8 Dualstellen zur Darstellung ganzer Zahlen verwendet, so hat die Zahl +5 die Belegung in Zeile 1 von Bild 2.6. Die duale Darstellung der Zahl −5 mit Betrag und Vorzeichen ist aus Zeile 2 zu ersehen. Bei der Komplementdarstellung wird bei negativen Zahlen das Komplement zu 2^8 gebildet. Das heißt, im Rechner wird anstelle der Zahl

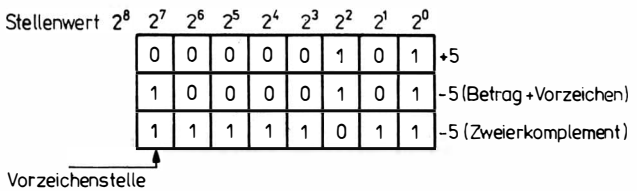


Bild 2.6 Darstellung der Zahlen +5 und −5 im Zweierkomplement

$-x$ die Zahl $2^8 - x$ gespeichert. Für die Zahl -5 entsteht dabei die Belegung entsprechend Zeile 3 in Bild 2.6.
Für die Komplementbildung kann man folgende Regel aufstellen:
Ausgehend von der positiven Dualzahl, werden zur Komplementbildung alle Stellen negiert und anschließend zum niedrigsten Stellenwert eine 1 addiert.

2.2.1.4. Mathematische Formulierung der Komplementbildung

Der positive Zahlenbereich der nach Bild 2.7 dargestellten Zahlen ist

$$0 \leqq x \leqq \sum_{i=-M}^{N} 2^i = 2^{N+1} - 2^{-M}.$$

Da die negativen Zahlen in den positiven Zahlenbereich transformiert werden sollen, gilt für die Zahlen x' im Rechner:

$$x' = \begin{cases} x, & \text{falls } x \geqq 0 \\ x + k, & \text{falls } x < 0 \end{cases}.$$

wobei k eine Komplementärzahl ist. Für die Komplementärzahl k ergeben sich folgende Bedingungen:

1. Es soll $x' > 0$ sein.

2. Positive und negative Zahlen müssen unterscheidbar sein.

Damit muß für negative Zahlen x

$$k + x > 2^{N+1} - 2^{-M}$$

sein, d. h.

$$k > 2^{N+1} - 2^{-M} - x = 2^{N+1} - 2^{-M} + |x|,$$

und mit

$$|x|_{max} = 2^{N+1} - 2^{-M}$$

wird

$$k > 2 \cdot 2^{N+1} - 2 \cdot 2^{-M} = k'.$$

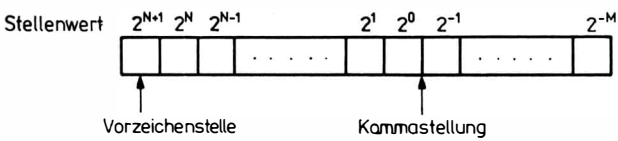


Bild 2.7 Duale Zahlendarstellung im Rechner

Die kleinstmögliche Komplementärzahl ist damit

$$k_1 = k' + 2^{-M} = 2^{N+2} - 2^{-M}.$$

Negative Zahlen, die mit dieser Komplementärzahl gebildet werden, heißen *Einerkomplement.*
Die nächstmögliche Komplementärzahl ist:

$$k_2 = k_1 + 2^{-M} = 2^{N+2}.$$

Zahlen. die mit k_2 gebildet werden, nennt man *Zweierkomplement*. In unserem Beispiel nach Bild 2.5 sind $M = 0$ und $N = 6$ und damit $k_2 = 2^8$.

2.2.1.5. Zahlensysteme mit der Basis 8 (Oktalsystem) und 16 (Hexadezimalsystem)

Sehr häufig werden zur übersichtlichen Darstellung von Dualzahlen dem Dualsystem verwandte Zahlensysteme verwendet. Hierzu gehören das Oktalystem und das Hexadezimalsystem.

Oktaldarstellung
Beim Oktalsystem beträgt die Basis $B = 8$. Da $8 = 2^3$ ist, bilden immer 3 Dualziffern eine Oktalziffer. Im Oktalsystem werden 7 Ziffern (0 bis 7) benötigt.

Beispiel
Die Dezimalzahl 201 lautet als Dualzahl 11001001 und als Oktalzahl 311 (11/001/001).

Hexadezimaldarstellung
Das Hexadezimalsystem hat die Basis $B = 16$. Da $16 = 2^4$ ist, bilden 4 Dualziffern eine Hexadezimalziffer. Im Hexadezimalsystem werden 16 Ziffern benötigt. Für die Ziffern (10), (11), (12), (13), (14), (15) setzt man gewöhnlich die Zeichen A, B, C, D, E, F ein.

Beispiel
Die Dezimalzahl 201 lautet als Hexadezimalzahl
C 9 (1100/1001).

2.2.1.6. Zifferncode

Um Zahlen binär auszudrücken, gibt es außer der reinen Dualdarstellung noch gemischte Formen, sogenannte Codierungen, in

denen die Ziffern einer Dezimalzahl getrennt durch Binärziffern dargestellt werden.
Der einfachste Code ist der *BCD-Code* (Binär-Code), in dem jede Ziffer durch die entsprechende Dualzahl dargestellt wird. Für eine Ziffer werden dabei 4 Dualstellen benötigt.

Beispiel

3791 ≙	0011	0111	1001	0001
	3	7	9	1

Die für eine Dezimalziffer notwendigen Dualstellen nennt man eine *Tetrade*. Mit diesen 4 Stellen lassen sich außer den Ziffern 0 bis 9 noch die Zahlen 10 bis 15 realisieren. Tritt innerhalb der Tetrade eine Kombination von Binärstellen auf, die einer der Zahlen 10 bis 15 entspricht (z. B. 1101 ≙ 13), so nennt man diese Kombination eine *Pseudotetrade*. **Pseudotetraden sind Kombinationen von Binärziffern, deren Wert keiner der Ziffern 0 bis 9 zugeordnet ist.**
Außer dem BCD-Code gibt es noch weitere Codierungsvorschriften für Ziffern, in denen andere Kombinationen von Binärziffern den Dualziffern 0 bis 9 zugeordnet sind.

3-Exzeß-Code
Codewert = Dualwert + 3.
Die 3-Exzeß-Code-Verschlüsselung ergibt für die Dezimalziffern 0 bis 9 folgende Zuordnung:

0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Aiken-Code
Codewert = Dualwert, falls Zahlenwert <5,
Codewert = Dualwert $+6$, falls Zahlenwert $\geqq 5$.

Der Aiken-Code ergibt für die Dezimalziffern 0 bis 9 folgende Zuordnung:

0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

Außer der Darstellung von Dezimalziffern durch 4 Dualstellen gibt es noch solche mit mehr als 4 Dualstellen.
Beispiele solcher Codierungen sind der *Walking-Code* und der *Positionscode*.

Walking-Code (2-aus-5-Code)

0 = 0 0 0 1 1
1 = 0 0 1 0 1
2 = 0 0 1 1 0
3 = 0 1 0 1 0
4 = 0 1 1 0 0
5 = 1 0 1 0 0
6 = 1 1 0 0 0
7 = 0 1 0 0 1
8 = 1 0 0 0 1
9 = 1 0 0 1 0

Positionscode (1-aus-10-Code)

0 = 0 0 0 0 0 0 0 0 0 1
1 = 0 0 0 0 0 0 0 0 1 0
2 = 0 0 0 0 0 0 0 1 0 0
3 = 0 0 0 0 0 0 1 0 0 0
4 = 0 0 0 0 0 1 0 0 0 0
5 = 0 0 0 0 1 0 0 0 0 0
6 = 0 0 0 1 0 0 0 0 0 0
7 = 0 0 1 0 0 0 0 0 0 0
8 = 0 1 0 0 0 0 0 0 0 0
9 = 1 0 0 0 0 0 0 0 0 0

2.2.1.7. Zahlenbereich im Rechner

Bei der Darstellung von Zahlen im Rechner muß man die Tatsache berücksichtigen, daß ein Rechner nur eine bestimmte Stellenzahl hat. Stehen z. B. nur 7 Stellen für positive Dualzahlen zur Verfügung, wie es bei vielen Mikroprozessoren der Fall ist, so kann man damit nur ganze Zahlen zwischen 0 und 127 darstellen. Sollen auch gebrochene Zahlen dargestellt werden, so kann man das Komma vor eine dieser Stellen setzen. Dabei wird der Zahlenbereich nicht erweitert, sondern nur verschoben. Steht bei 7 Stellen das Komma nach der 4. Stelle, so kann man nur Zahlen zwischen 0 und 15,875 in Schritten zu 0,125 darstellen.
Damit der Zahlenbereich an die praktischen Erfordernisse angepaßt wird, gibt es noch verschiedene Darstellungsformen innerhalb eines Zahlensystems, die wichtigsten sind die *Festkomma-* und die *Gleitkommazahlen.*

Festkommazahlen
Dabei handelt es sich um Zahlen, bei denen eine feste Anzahl von Ziffern vor und nach dem Komma vereinbart wird. Die Stellung des Kommas und die Gesamtzahl der Ziffern hängt vom Rechnertyp ab. Es gibt auch Rechner, bei denen das Komma durch Tasten oder selbständig gesetzt wird.

Beispiel
In jedem Taschenrechner sind Dezimalzahlen der Form 3 7 2 8 3 · 6 1 3 üblich. In diesem Fall rechnet der Taschenrechner mit 8 Dezimalstellen in Festkommadarstellung.
Im Dualsystem besteht das gleiche Problem. In einem Rechner läßt sich wegen der technischen Gegebenheiten nur eine feste Anzahl von Dualstellen speichern. Dabei kann das Komma rechts von der niedrigsten Stelle (dann handelt es sich um ganze Zahlen) oder links vor der höchsten Stelle stehen (dann handelt es sich um echt gebrochene Zahlen), oder das Komma trennt einen ganzen und einen gebrochenen Teil.
Bei der Addition und Subtraktion von Festkommazahlen müssen die Zahlen so verschoben werden, daß die Kommas untereinander stehen. Bei Multiplikation und Division muß man die Stelle des Kommas besonders bestimmen.

Gleitkommazahlen
Jede Zahl Z läßt sich in folgender Form darstellen:

$$Z = m \cdot B^E$$

(z. B. $0{,}19 \cdot 10^{-18}$; $m = 0{,}19$; $E = -18$; $B = 10$).

Dabei nennt man m die *Mantisse*, E den *Exponenten*, und B ist die *Basis* des Zahlensystems.

Bei der dualen Darstellung von Gleitkommazahlen gilt für m bei den meisten Rechnern folgende Vorschrift:

1. $-1 < m < +1$.
2. Die 1. Stelle nach dem Komma soll nicht 0 sein.

Zahlen, die diesen Bedingungen genügen, nennt man *normalisierte Zahlen*. Durch diese Vorschrift werden die Mantisse m und der Exponent E eindeutig bestimmt. Es gibt aber auch Vorschriften für die Bildung der Mantisse m, die von der genannten Vorschrift abweichen.

Beispiel 1

Man schreibe die Zahl 25,211 als Dezimalzahl in Gleitkommadarstellung.

Lösung: $25{,}211 = 0{,}25211 \cdot 10^2$.

Die Mantisse lautet also $m = 0{,}25211$ und der Exponent $E = 2$.

Beispiel 2

Man schreibe die Zahl 4,25 als Gleitkommazahl in dualer Darstellung.

Lösung: $4{,}25 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$.

Das ergibt die Dualzahl 100,01.

Als Gleitkommazahl entsprechend obiger Vorschrift wird daraus $0{,}10001 \cdot 2^3 = 0{,}10001 \cdot 2^{11}$.

Die Mantisse lautet also $m = 0{,}10001$ und der Exponent $E = 11$.

2.2.2. Darstellung von Text (Zeichencode)

Damit der Rechner beliebige Textinformationen speichern kann, müssen alphanumerische Zeichen auch durch Bit-Folgen (Folgen aus den Ziffern 0 und 1) dargestellt werden. Dabei ergibt sich die Frage, wieviel Dualstellen insgesamt zur Codierung aller vorkommenden Zeichen notwendig sind. Aus der Mathematik ist bekannt, daß mit n Dualstellen 2^n unterschiedliche Bit-Folgen, bestehend

aus den Ziffern 0 und 1, dargestellt werden können. Nimmt man an, daß die Anzahl der verwendeten Zeichen (Buchstaben, Ziffern und Syntaxzeichen) kleiner als 128 ist, so benötigt man dazu 7 Dualstellen, da $2^7 = 128$ ist. Meistens kommt zu diesen 7 Bit eine Kontrollbit hinzu. Die Bit-Folge von 8 Dualstellen nennt man 1 *Byte*. Innerhalb dieser 8 Stellen wird jedem Zeichen eine feste Folge aus den Ziffern 0 und 1 zugeordnet. Diese Folge ist der Code des betreffenden Zeichens.
Es gibt mehrere internationale Festlegungen für solche Zuordnungen. Die in der DDR am häufigsten verwendeten Zeichencode sind:

- SIF-1000-Code (Standard-Interface-Code für Datenverarbeitungsperipherie);
- ASCII-Code (American Standard Code for Information Interchange – amerikanischer Code für Informationsaustausch);
- R-300-Code;
- ISO-7-Bit-Code.

In Mikrorechnern wird vorwiegend der ASCII-Code und der SIF-1000-Code angewendet.

2.2.3. Rechnen mit Dualzahlen

Addition
Während bei Dezimalzahlen ein Übertrag entsteht, wenn die Summe zweier Ziffern größer als 9 ist, tritt im Dualsystem der Übertrag bereits auf, wenn die Summe größer als 2 ist. Im Dualsystem gelten folgende Grundregeln:
0 + 0 = 0 mit Übertrag 0;
0 + 1 = 1 mit Übertrag 0;
1 + 0 = 1 mit Übertrag 0;
1 + 1 = 0 mit Übertrag 1.

Beispiel
Bilden der Summe 13 + 7.

	1 Übertrag		1111 Übertrag
Dezimal:	13	Dual:	13 ≙ 1101
	+ 7		+ 7 ≙ 111
	20		20 ≙ 10100

Subtraktion

Gelingt die Subtraktion in einer Stelle nicht direkt, so muß der Minuend durch „Borgen" aus der nächsthöheren Stelle erhöht werden. Im Dualsystem wird das „Borgen" aus der nächsthöheren Stelle dadurch realisiert, daß von der höherwertigen Stelle, wie auch im Dezimalsystem, eine 1 abgezogen und dafür die betreffende Stelle um 10 erhöht wird

Beispiel

Bilden der Differenz 91−53.

	1 geborgt							1 geborgt
Dezimal:	91	Dual:	91 ≙	1	0	1	1	0	1	1
	− 53		− 53 ≙		1	1	0	1	0	1
	38		38 ≙		1	0	0	1	1	0

Werden negative Dualzahlen im Komplement dargestellt, dann realisiert man die Subtraktion durch die Addition im Zweierkomplement. Die Zahl −53 sieht in Zweierkomplementdarstellung folgendermaßen aus:

−53 ≙ 1 1 0 0 1 0 1 1 ,

wobei das vorderste Bit die Vorzeichenstelle ist.

Das Beispiel 91−53 sieht, wenn statt 91−53 91 + (−53) gerechnet wird, so aus:

```
  91 ≙ 0 1 0 1 1 0 1 1
−53 ≙ 1 1 0 0 1 0 1 1
       0 0 1 0 0 1 1 0
```

Der Übertrag in die Stelle vor dem Vorzeichen wird dabei nicht mehr berücksichtigt, da er über die vorgegebene Stellenzahl hinausgeht. Zur Begründung der angewandten Methode setzt man für eine negative Zahl y in der Komplementdarstellung das Zeichen y'. Dann ist $y' = y + k_2$, mit $k_2 = 2^8$, wenn es sich um das Zweierkomplement handelt.

Die Zahl $k_2 = 2^8$ läßt sich mit den vorgegebenen Stellen nicht darstellen, da der höchste Stellenwert 2^7 ist.

Für $x - y$ gilt:

$x - y = x + (-y) = x + y' = x - y + k_2$.

Ist $x - y$ negativ, so steht das Ergebnis mit $x - y + k_2$ richtig im Register. Ist $x - y$ positiv, so ergibt sich ebenfalls ein richtiges Ergebnis, da die Stelle für k_2 nicht vorhanden ist.

Sind x und y *negativ, dann gilt:*
$-x - y = x' + y' = -x - y + 2k_2 = -x - y + k_2 + k_2$.
Der Wert $-x - y + k_2$ ist die Komplementdarstellung von $-x - y$. Das zweite k_2 wird nicht im Register dargestellt, da die dafür notwendige Stelle nicht vorhanden ist.

Multiplikationen
Zur Multiplikation müssen die beiden Operanden positiv sein. Die Multiplikation von positiven Zahlen gleicht der im Dezimalsystem, wobei folgende Grundregeln gelten:
$0 \cdot 0 = 0$;
$0 \cdot 1 = 0$;
$1 \cdot 0 = 0$;
$1 \cdot 1 = 1$.
Die Multiplikation ist eine fortgesetzte Addition, wobei die einzelnen Summanden entsprechend ihrem Stellenwert verschoben sind.

Beispiel
Bilden des Produkts 12 * 5 .
1100 * 101
0000
1100
111100 ≙ 60
Auch hier kann wie beim Rechnen mit Dezimalzahlen im angegebenen Beispiel die Nullzeile weggelassen werden.

Division
Die Division ist im wesentlichen eine fortgesetzte Subtraktion. In den Quotienten wird dann eine Ziffer 1 eingetragen, wenn der Subtrahend kleiner als der Minuend ist. Da die Subtraktion auch als Addition im Komplement realisiert werden kann, ergeben sich unterschiedliche Verfahren für die Division, die jedoch im Prinzip auf der gleichen Grundlage beruhen.

Beispiel 1
Bilden der Division 108 : 12 nach dem normalen Handrechenverfahren.

1 1 0 1 1 0 0 : 1 1 0 0 = 1 0 0 1	
− 1 1 0 0	geht zu subtrahieren (Subtrahend ist kleiner als Minuend), Ergebnis ist 1;
0 0 0 1 1 0 0	Rest

Divisor 1 Stelle nach rechts	– 1 1 0 0	geht nicht zu subtrahieren, Ergebnis ist 0;
	0 0 0 1 1 0 0	Rest
Divisor 1 Stelle nach rechts	– 1 1 0 0	geht nicht zu subtrahieren, Ergebnis ist 0;
	0 0 0 1 1 0 0	Rest
Divisor 1 Stelle nach rechts	– 1 1 0 0	geht zu subtrahieren, Ergebnis ist 1;
	0 0 0 0 0 0 0	Rest

Da der Rest 0 ist, lautet das Ergebnis 1 0 0 1 .

Beispiel 2

Bilden der Division 108 : 12 mit Rückstellung des Restes unter Verwendung der Komplementdarstellung.

1 1 0 1 1 0 0 : 1 1 0 0 = 1 0 0 1

Stellenzahl	8 7 6 5 4 3 2 1 0	
	0 1 1 0 1 1 0 0	Divident
	+ 1 0 1 0 0 0 0 0	Zweierkomplement des Divisors
	0 0 0 0 1 1 0 0	Ergebnis ist positiv – ergibt eine 1 im Quotienten
	+ 1 1 0 1 0 0 0 0	Divisor um eine Stelle nach rechts verschoben (Zweierkomplement)
	1 1 0 1 1 1 0 0	Ergebnis ist negativ – ergibt eine 0 im Quotienten
	+ 1 1 0 0	Da das vorangegangene Ergebnis negativ ist, muß der Divisor wieder dazugezählt werden (Rückstellung des Restes)
	0 0 0 0 1 1 0 0	
	+ 1 1 1 0 1 0 0 0	Divisor um eine Stelle nach
	1 1 1 1 0 1 0 0	rechts verschoben, Ergebnis ist negativ – ergibt eine 0 im Quotienten

+	1100	Da das vorangegangene Ergebnis negativ ist, muß der Divisor
	00001100	wieder dazugezählt werden (Rückstellung des Restes)
+	11110100	Divisor um 1 nach rechts verschoben
	00000000	Ergebnis positiv – ergibt 1 im Quotienten

Der verbleibende Rest ist 0, die Division geht auf und ergibt den Quotienten 1001.
Geht man von der Tatsache aus, daß Multiplikation und Division durch fortgesetzte Additionen bzw. Subtraktionen realisiert werden, so läßt sich zeigen, daß im Dualsystem weniger Additionen und Subtraktionen notwendig sind als im Dezimalsystem.

Rechnen mit codierten Zahlen

In diesem Abschnitt wird nur auf BCD-codierte Zahlen eingegangen, da der BCD-Code der am häufigsten verwendete Code ist. Für die Rechenoperationen im BCD-Code gibt es bei den Mikroprozessoren *8080* und *U 880* spezielle Befehle. Werden 2 Dezimalziffern im BCD-Code addiert, so können folgende Fälle auftreten:

1. Die Summe ist ≦9.
2. Die Summe ist >9 aber ≦15.
3. Die Summe ist >15.

Im Fall 1 ist die Ergebnisziffer richtig.

Beispiel

Summe 4 + 5: 4 ≙ 0100
5 ≙ 0101
9 ≙ 1001

Im Fall 2 ist die Ergebnisziffer eine Pseudotetrade. Die richtige Ziffer erhält man durch Addition von 6.

Beispiel

Summe 7 + 6: 7 ≙ 0111
6 ≙ 0110
1101 = Pseudotetrade
\+ 110 (Addition 6)
13 ≙ 10011

Im Fall 3 ergibt sich bereits der richtige Übertrag, aber die Ergebnisziffer muß noch um 6 erhöht werden.

Beispiel

Summe 8 + 9: 8 ≙ 1000
9 ≙ 1001

Übertrag in die nächste Dezimalziffer ⟶ 1 0001
\+ 110 (Addition 6)
17 ≙ 10111

Für die Addition im BCD-Code lassen sich also folgende Regeln aufstellen:
Ergibt die Summe zweier Ziffern plus Übertrag von der vorherigen Stelle keine Pseudotetrade und keinen Übertrag in die nächste Dezimalstelle, so ist die Ergebnisziffer richtig.
Ergibt die Summe zweier Ziffern eine Pseudotetrade oder einen Übertrag in die nächste Stelle, dann muß die Ergebnisziffer um 6 erhöht werden.

Beispiel

Summe 2985 + 4936 = 7921.

1←	1←	1←	Übertrag in
0 0 1 0	1 0 0 1	1 0 0 0	0 1 0 1 nächste Tetrade
0 1 0 0	1 0 0 1	0 0 1 1	0 1 1 0
0 1 1 1	0 0 1 1	1 1 0 0 ≙ PS*	1 0 1 1 ≙ PS*
	1 1 0 (+6)	1 1 0 (+6)	1 1 0 (+6)
0 1 1 1	1 0 0 1	0 0 1 0	0 0 0 1 ≙ 7921

* PS = Pseudotetrade

Rechnen mit Gleitkommazahlen

Addition und Subtraktion von Gleitkommazahlen

Die Addition bzw. Subtraktion der Mantissen zweier Zahlen ist nur dann möglich, wenn die Exponenten beider Zahlen gleich sind.
Für die Berechnung von
$Z_1 + Z_2 = m_1 \cdot 10^{E1} + m_2 \cdot 10^{E2}$
muß unter der Voraussetzung, daß $E_1 < E_2$ ist, m_1 um $E_2 - E_1$

Stellen nach rechts verschoben werden. Die neue Mantisse sei mit m_1' bezeichnet, dann ist
$Z_1 + Z_2 = m_1' \cdot 10^{E2} + m_2 \cdot 10^{E2} = (m_1' + m_2) \cdot 10^{E2}$.
Die neue Mantisse ist also $m_1' + m_2$, der neue Exponent E_2.

Beispiel
$Z_1 = 0{,}25 \cdot 10^2$;
$Z_2 = 0{,}35 \cdot 10^4$.
$E_2 - E_1 = 2$, d. h., die Mantisse von Z_1 muß um 2 Stellen nach rechts verschoben werden. Es gilt also:
$Z_1 + Z_2 = 0{,}25 \cdot 10^2 + 0{,}35 \cdot 10^4$
$= 0{,}0025 \cdot 10^4 + 0{,}35 \cdot 10^4 = 0{,}3525 \cdot 10^4$.
Bei der Addition im Dualsystem verfährt man analog.
Bei der Subtraktion ist in entsprechender Weise vorzugehen.

Multiplikation von Gleitkommazahlen
Die Multiplikation von Gleitkommazahlen läuft nach der Regel
$Z_1 \cdot Z_2 = m_1 \cdot 10^{E1} \cdot m_2 \cdot 10^{E2} = m_1 \cdot m_2 \cdot 10^{E2 + E2}$
ab. Die neue Mantisse ist also $m_1 \cdot m_2$, der neue Exponent $E_1 + E_2$. Die neue Mantisse $m_1 \cdot m_2$ darf ebenfalls keine 0 nach dem Komma aufweisen, d. h., gegebenenfalls ist die 0 nach dem Komma durch Verschiebung von $m_1 \cdot m_2$ um ein Stelle nach links zu beseitigen, wobei der Exponent $E_1 \cdot E_2$ um 1 zu erniedrigen ist.

Beispiel 1 (Dezimalsystem)
$Z_1 = 0{,}2 \cdot 10^2$;
$Z_2 = 0{,}3 \cdot 10^3$;
$Z_1 \cdot Z_2 = 0{,}2 \cdot 0{,}3 \cdot 10^{2+3} = 0{,}06 \cdot 10^5 = 0{,}6 \cdot 10^4$.
(Die „0" nach dem Komma bei 0,06 wurde durch Verschiebung um eine Stelle nach rechts und Erniedrigung des Exponenten 5 um 1 beseitigt.)
Im Dualsystem wird in entsprechender Weise verfahren.

Beispiel 2 (Dualsystem)
$3 \cdot 4 = 12$.
$3 \triangleq 11 = 0{,}11 \cdot 2^2 = 0{,}11 \cdot 2^{10}$; $4 \triangleq 100 = 0{,}1 \cdot 2^3 = 0{,}1 \cdot 2^{11}$;
$0{,}11 \cdot 2^{10} \cdot 0{,}1 \cdot 2^{11} = 0{,}11 \cdot 0{,}1 \cdot 2^{10+11} = 0{,}011 \cdot 2^{101}$;
$= 0{,}11 \cdot 2^{(101-1)} = 0{,}11 \cdot 2^{100}$.

Division von Gleitkommazahlen
Die Division von Gleitkommazahlen läuft nach der Regel

$$\frac{Z_1}{Z_2} = \frac{m_1 \cdot 10^{E1}}{m_2 \cdot 10^{E2}} = \frac{m_1}{m_2} \cdot 10^{E1-E2}$$

ab. Die neue Mantisse ist also $m_1 : m_2$, der neue Exponent $E_1 - E_2$. Da die neue Mantisse der Vorschrift $-1 < m < +1$ genügen muß, ist gegebenenfalls die Ergebnismantisse um eine Stelle nach rechts zu verschieben und der Exponent dabei um 1 zu erhöhen.

Beispiel 1 (Dezimalsystem)
$Z_1 = 0{,}25 \cdot 10^2$;
$Z_2 = 0{,}5 \cdot 10^1$;
$$\frac{Z_1}{Z_2} = \frac{0{,}25 \cdot 10^2}{0{,}5 \cdot 10^1} = \frac{0{,}25}{0{,}5} \cdot 10^{2-1} = 0{,}5 \cdot 10^1.$$

Beispiel 2 (Dualsystem)
$6 : 2 = 3$
$Z_1 = 6 \triangleq 0{,}11 \cdot 2^{11}$;
$Z_2 = 2 \triangleq 0{,}1 \cdot 2^{10}$;
$$\frac{Z_1}{Z_2} = \frac{0{,}11 \cdot 2^{11}}{0{,}1 \cdot 2^{10}} = \frac{0{,}11}{0{,}1} \cdot 2^{11-10} = 1{,}1 \cdot 2^1 = 0{,}11 \cdot 2^{1+1} = 0{,}11 \cdot 2^{10}.$$

2.3. Aufbau der Befehle

Genauso wie die Zahlen durch Bit-Folgen dargestellt werden, muß man auch die Anweisungen (Befehle) in Bit-Folgen umwandeln, um sie zu speichern. Da eine Anweisung (Befehl) eine Information sein soll, aus der hervorgeht, was der Rechner zu tun hat, besteht sie aus 2 Hauptteilen:

- Der 1. Teil sagt aus, was bzw. welche Operation der Rechner ausführen soll.
- Der 2. Teil sagt aus, woher die Daten kommen, die für die Durchführung dieser Operation gebraucht werden.

Der 1. Teil wird *Operationsteil,* der 2. Teil *Adreßteil* genannt (Bild 2.8). Im Adreßteil steht eine Adresse (Einadreßbefehl) oder mehrere Adressen (Mehradreßbefehl) für die benötigten Daten. Im Adreßteil eines Einadreßbefehls steht entweder die fertige Adresse oder, wie es oft der Fall ist, eine Rechenvorschrift, aus der die endgültige Adresse ermittelt wird. Man spricht in diesem Fall

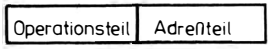


Bild 2.8 Befehlsaufbau

von *Adreßrechnung*. Durch die Adreßrechnung wird aus den im Adreßteil des Befehls stehenden Angaben die endgültige Speicheradresse ermittelt.
Bei der Adreßrechnung gibt es 2 Grundprinzipien, auf die bei nahezu allen Rechnern zurückgegriffen wird. Das erste Prinzip ist unter dem Namen *Index-Rechnung* bekannt. Bei der Index-Rechnung wird die endgültige Speicheradresse ermittelt, indem zu der im Befehl angegebenen Adresse der Inhalt eines speziellen Registers, *Index-Register* genannt, dazugezählt wird. Die endgültige Speicheradresse ist also gleich im Befehl angegebene Adresse plus Inhalt Index-Register (Indexierung):
Endgültige Speicheradresse = im Befehl angegebene Adresse + ⟨Index-Register⟩.
Das zweite Prinzip ist unter dem Namen *indirekte Adressierung* bekannt. Bei der indirekten Adressierung steht die endgültige Speicheradresse in der Zelle, deren Adresse im Befehl steht; oder anders ausgedrückt, die endgültige Adresse ist der Inhalt der im Befehl angegebenen Speicherzelle. Es kommt auch vor, daß die endgültige Adresse nicht in der Speicherzelle, sondern in einem speziellen Register steht.
Endgültige Speicheradresse = Inhalt der im Befehl angegebenen Speicherzelle.

2.4. Befehlsschlüssel eines Rechners

Der Befehlsschlüssel ist eine Zusammenfassung aller Befehle, die ein Rechner ausführen kann. Jedem Befehl wird eine bestimmte Bit-Folge im Operationsteil zugeordnet. Diese Bit-Folge nennt man *Operationscode*. Bei Befehlen, die keinen Speicherzugriff benötigen (*nichtspeicherbezogene Befehle*), wird zur Codierung des Befehls der Adreßteil hinzugezogen. Die folgende Zusammenstellung enthält eine Übersicht über Befehlsarten, die ein Mikroprozessor ausführen kann.

– *Adreßoperationen*
Adreßoperationen sind Operationen zur Ermittlung der endgül-

tigen Speicheradresse aus den Angaben im Adreßteil des Befehls (Indexierung, indirekte Adressierung).

– *Transportoperationen*

Transportoperationen dienen zur Übertragung von Daten vom Speicher in spezielle Register und umgekehrt oder von Speicherzellen in andere Speicherzellen. Man unterscheidet *Einzelworttransfer* und *Blocktransfer*. Einzelworttransfer ist die Übertragung eines Wortes oder Bytes, Blocktransfer ist die Übertragung eines Datenblocks von einem Speicherbereich in einen anderen Speicherbereich.

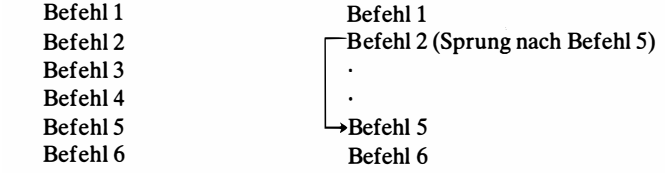
– *Rechen- und logische Operationen*

Diese Operationen dienen zur Verknüpfung von Zahlen bzw. Bit-Folgen. Bei den Rechenoperationen werden Zahlen durch Rechenvorschriften (Addition, Subtraktion, Multiplikation, Division), bei den logischen Operationen durch logische Operatoren (UND, ODER, NEGATION, EXKLUSIV-ODER usw.) verknüpft. Das Resultat steht meistens in einem speziellen Register, dem *Akkumulatorregister*.

Außerdem gibt es Befehle, durch die einzelne Bits einer Bit-Folge verändert werden, sowie Befehle zum Vergleich zweier Bit-Folgen.

Sprungoperationen

Sprungoperationen dienen zur Gestaltung der Programmstruktur. Durch einen Sprungbefehl ist es möglich, von einer Stelle im Programm an eine beliebige andere Stelle zu springen und dort die Abarbeitung fortzusetzen.



Unterprogrammbefehle

Ein Unterprogramm ist ein Programm, das eine spezielle Funktion ausführt, die während der Abarbeitung eines größeren Programms mehrmals notwendig ist (z. B. Berechnung des Logarithmus einer Zahl). Man spricht deshalb vom *Hauptprogramm*, das die gesamte Aufgabe realisiert, und vom *Unterprogramm* zur Lösung einer speziellen Funktion.

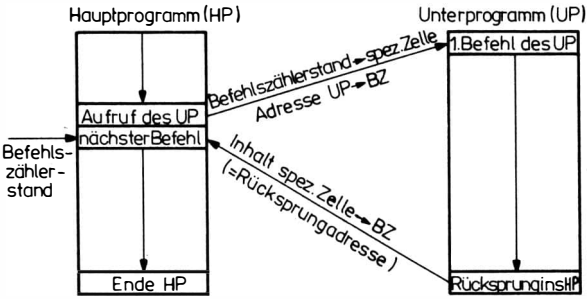


Bild 2.9 Arbeitsweise eines Unterprogramms

Das Unterprogramm steht nur einmal im Speicher und muß so arbeiten, daß man es von verschiedenen Stellen des Hauptprogramms aus aufrufen kann. Nach dem Durchlaufen des Unterprogramms muß an die Stelle im Hauptprogramm zurückgesprungen werden, von der aus ins Unterprogramm gesprungen wurde. Der Sprungbefehl ins Unterprogramm besteht aus 2 Funktionen. Erstens wird ein einfacher Sprung zur Startadresse des Unterprogramms ausgeführt, zweitens muß die Rücksprungstelle ins Hauptprogramm gemerkt werden. Die Rücksprungadresse (Inhalt Befehlszähler + 1) wird in einer speziellen Speicherzelle gemerkt.
Der Rücksprung aus dem Unterprogramm ist ein Sprungbefehl, dessen Adresse aus der Speicherzelle genommen wird, in der die Rücksprungadresse steht (Bild 2.9).

Steueroperationen

Diese Operationen sind spezielle Befehle, die den Zustand des Rechners festlegen. Dazu gehört z. B. der HALT-Befehl.

Ein- und Ausgabeoperation

Sie dienen zur Ansteuerung entsprechender Kanäle, an die Geräte zur Ein- bzw. Ausgabe von Daten angeschlossen sind.

2.5. Befehlsabarbeitung

Zur Abarbeitung eines Programms werden die Befehle der Reihe nach aus dem Speicher in die CPU geholt. Die Adresse der Speicherzelle, aus der der Befehl kommt, steht im Befehlszähler (BZ). Die Ausführung eines Befehls geht in einzelnen Abschnitten, den

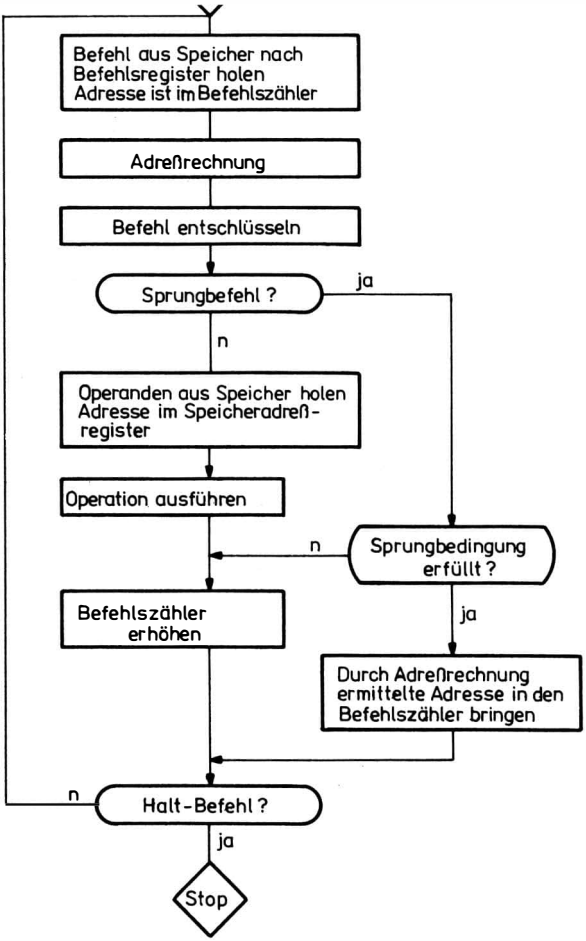


Bild 2.10 Befehlsabarbeitung

Maschinenzyklen, vor sich. Bild 2.10 zeigt, welche Funktionen bei der Ausführung eines Befehls abgearbeitet werden.

2.6. Ein- und Ausgabesteuerung (E/A-Steuerung)

2.6.1. Prinzip der E/A-Steuerung

Die Ein-/Ausgabe-Steuerung stellt die Verbindung zwischen dem Rechner und den peripheren Geräten her (Bild 2.11).
Die Schnittstellen zwischen Rechner und E/A-Steuerung und zwischen E/A-Steuerung und externem Gerät bestehen aus Daten-, Adreß- und Steuerleitungen. Man spricht vom *Datenbus, Adreßbus* und *Steuerbus.* Ist an die E/A-Steuerung nur ein externes Gerät angeschlossen, dann kann der dazugehörige Adreßbus entfallen. Entsprechen die Leitungen einer Schnittstelle einer Normung, so spricht man von einer *Standard-Interface.*

Datenbus
Über den Datenbus werden Dateninformationen übertragen. Er besteht meistens aus 8 (1 Byte) oder 16 Leitungen.

Adreßbus
Der Adreßbus enthält die Adresse der E/A-Steuerung bzw. die Adresse des externen Geräts. Das Gerät entschlüsselt die Adresse und bildet ein Adreßsignal, wenn es durch die richtige Adresse angesprochen wird.

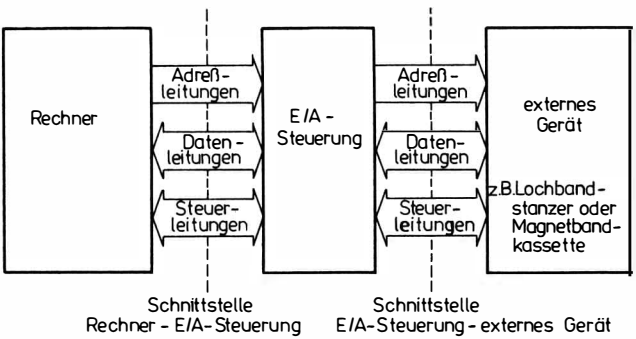


Bild 2.11 Ein-/Ausgabe-Steuerung

Steuerbus
Der Steuerbus enthält Signale, die den Zeitpunkt der Datenübertragung festlegen. Es sind Melde- und Steuersignale. Meldesignale sind Ausgangssignale eines Geräteteils. Wichtige Meldesignale sind:

- Geräteteil für Daten empfangsbereit;
- Geräteteil hat Daten übernommen;
- Geräteteil hat Daten zur Ausgabe bereitgestellt;
- Fehler im Geräteteil.

Die Meldesignale des sendenden Geräteteils sind gleichzeitig Steuersignale für den empfangenden Geräteteil.
Der Datenaustausch geschieht im allgemeinen nach dem sogenannten *Hand-shake-Prinzip.* (Die anfordernde Stelle gibt ein Anforderungssignal (Request) und wartet, bis die Gegenstelle ein Quittungssignal zurückgibt.)

Beispiel
Die E/A-Steuerung A gibt Daten an das Gerät G (Bild 2.12):

- Die E/A-Steuerung sendet die Adresse des Gerätes G und stellt die Daten auf dem Datenbus bereit.
- Die E/A-Steuerung sendet als Zeichen bereitstehender Daten das Signal „A-Bereit".
- Das Gerät G entschlüsselt die Adresse und übernimmt, wenn es bereit ist, die Daten.
- Hat G die Daten übernommen, so sendet es das Signal „Daten übernommen".
- Jetzt kann die E/A-Steuerung die Daten und die Adresse wieder abschalten.

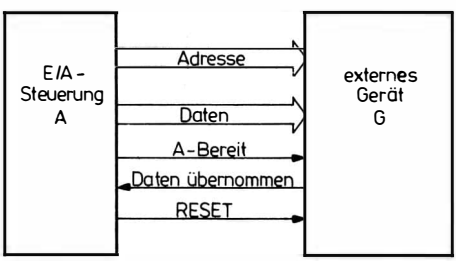


Bild 2.12 Aufbau einer Interface-Schnittstelle

Die Signale „A-Bereit" und „Daten übernommen" sind Signale des Steuerbusses. Ein wichtiges Steuersignal ist das *Löschsignal* (RESET), mit dem vom Rechner aus alle angeschlossenen Geräte in die Ausgangsstellung gesetzt werden.

2.6.2. Programmierte Ein- und Ausgabe

Wird ein Rechnerwort vom Rechner über die E/A-Steuerung zum externen Gerät mit Hilfe eines Ein- oder Ausgabebefehls ein- oder ausgegeben, so spricht man von *programmierter Ein-* bzw. *Ausgabe.* Dabei wird meistens vom Datenbus ein Byte in ein Register der CPU eingegeben oder der Inhalt eines Registers auf den Datenbus bereitgestellt. Der Ein- bzw. Ausgabebefehl enthält die Richtung (Ein- oder Ausgabe) und die Geräteadresse. Für die programmierte Ein- und Ausgabe gibt es 4 Befehlsgruppen.

E/A-Befehle Gruppe 1: *(E/A-Steuerbefehle)*	Durch den Befehl werden Funktionssignale zum externen Gerät gesandt, die dort eine bestimmte Funktion auslösen (z. B. Magnetbandkassette starten).
E/A-Befehle Gruppe 2: *(E/A-Testbefehle)*	Durch den Befehl wird der Zustand des externen Geräts abgefragt und in Abhängigkeit vom Zustand ein bedingter Sprung ausgeführt.
E/A-Befehle Gruppe 3: *(Eingabebefehle)*	Durch den Befehl wird die Eingabe eines Datenwortes vom Datenbus in ein spezielles Register der CPU realisiert.
E/A-Befehle Gruppe 4: *(Ausgabebefehle)*	Durch den Befehl wird der Inhalt eines speziellen Registers der CPU auf dem Datenbus bereitgestellt.

2.6.3. Autonome Ein- und Ausgabe

Die *autonome Ein-* und *Ausgabe* ist auch unter den Begriffen DSK (*D*irekter *S*peicher*k*anal) oder DMA (*D*irect *M*emory *A*ccess)

bekannt. Dabei geschieht die Ein- bzw. Ausgabe in Datenblöcken zwischen Speicher und externem Gerät mit Hilfe der E/A-Steuerung (*DMA-Steuerung* genannt).
Die DMA-Steuerung enthält folgende Register:

- Adreßregister zum Adressieren des Speichers;
- Datenregister (Datenpuffer) zur Zwischenspeicherung eines Wortes;
- Geräteadreßregister;
- Blocklängenregister.

Vor dem Auslösen des autonomen Datentransfer werden über die programmierte Ausgabe Adreßregister, Geräteregister und Blocklängenregister mit den entsprechenden Werten gefüllt. Nach dem Start des autonomen Datentransfer geht der weitere Datenaustausch zwischen Speicher und externem Gerät mit Hilfe der DMA-Steuerung vor sich. Zwischen DMA-Steuerung und externem Gerät vollzieht sich der Datenaustausch wie beim programmierten Kanal im Hand-shake-Verfahren. Ist die DMA-Steuerung zum Datenaustausch mit dem Speicher bereit (Datenpuffer voll bei Eingabe, Datenpuffer leer bei Ausgabe), so erfolgt ein Speicherzugriff. Dazu läuft ein Speicherzyklus ab (Speicherzyklus: Adresse an Adreßschlüsselung, Speicheranforderungssignal anlegen, Speicher auf Lesen oder Schreiben schalten, Daten auslesen oder einschreiben). Dieser Speicherzyklus wird bei Rechnern, in denen parallel zur autonomen Ein- und Ausgabe die Arbeit mit der CPU (*C*entral *P*rocessor *U*nit = Rechenwerk und Steuerwerk) möglich ist, eingeschoben. Nach der Ein- bzw. Ausgabe des Datenblocks gibt die DMA-Steuerung ein „Endsignal" ab.

Beispiel
Ausgabe eines Datenblocks vom Speicher. Dabei werden folgende Teilschritte ausgeführt:

- Einstellen des Grundzustands durch E/A-Befehle über den programmierten Kanal;
  a) Füllen des Adreßregisters mit der Anfangsadresse des Datenblocks,
  b) Füllen des Blocklängenregisters mit der Anzahl der Worte,
  c) Füllen des Geräteadreßregisters mit der Adresse des Ausgabegeräts (z. B. Drucker),
  d) Starten zur autonomen Datenausgabe.

Bild 2.13 Taktdiagramm einer Ausgabe über den DMA-Kanal

- Die DMA-Steuerung fordert einen Speicherzyklus;
  a) Adresse an Adreßentschlüsselung,
  b) Speicheranforderungssignal,
  c) Speicher lesen, Wort nach Datenpuffer,
  d) Inhalt des Adreßregisters um 1 erhöhen,
  e) Inhalt des Blocklängenregisters um 1 erniedrigen.
- Zwischen DMA-Steuerung und Ausgabegerät findet ein Hand-shake-Zyklus statt;
  a) die DMA-Steuerung sendet die Adresse des Ausgabegeräts und stellt die Daten auf dem Datenbus bereit,
  b) die DMA-Steuerung sendet das Signal „Daten vorhanden",
  c) das Ausgabegerät übernimmt das Datenwort,
  d) das Ausgabegerät sendet „Daten übernommen",
  e) die DMA-Steuerung schaltet das Signal „Daten vorhanden" ab.

Speicherzyklus und Hand-shake-Zyklus zum Ausgabegerät werden so lange wiederholt, bis das Blocklängenregister Null ist. Jetzt sendet die DMA-Steuerung das „Endesignal". Bild 2.13 zeigt das Taktdiagramm einer Ausgabe über den DMA-Kanal.

2.7. Programmunterbrechung (INTERRUPT)

Durch eine Programmunterbrechung besteht die Möglichkeit, den Programmablauf durch Signale von außen zu steuern. Ein in den Rechner gegebenes Programmunterbrechungssignal führt zum Abbrechen des gerade laufenden Programms und zur Fortsetzung der Arbeit des Rechners mit einem anderen Programm. Nach einem Programmunterbrechungssignal wird zunächst ein dem Signal zugeordnetes Bedienprogramm abgearbeitet. Erst danach setzt der Rechner das unterbrochene Programm fort. Im einzelnen erzeugt ein Programmunterbrechungssignal folgende Arbeitsgänge:

- Nach Beendigung des gerade laufenden Befehls wird ein INTERRUPT-Zyklus durchlaufen.
- Der Rechner gibt ein Signal „INTERRUPT angenommen" als Quittung nach außen ab.
- Während des INTERRUPT-Zyklus wird ein INTERRUPT-Befehl gebildet und anschließend sofort abgearbeitet.

Der INTERRUPT-Befehl kann ein Sprung in ein Unterprogramm sein. Das Unterprogramm ist das entsprechende Bedienpro-

gramm. Der Rücksprung ins unterbrochene Programm wird genauso organisiert wie der Rücksprung aus einem Unterprogramm. Am Anfang des Bedienprogramms bringt man im allgemeinen die Inhalte der Register, die nach dem Rücksprung ins unterbrochene Programm noch benötigt werden, in den *STACK* (Stapelspeicher). Vor dem Rücksprung werden die Register wieder mit den im STACK geretteten Inhalten gefüllt.
Ein Rechner kann einen oder mehrere INTERRUPT-Eingänge haben. Die zu den Eingängen gebildeten INTERRUPT-Befehle unterscheiden sich. Die einzelnen Eingänge lassen sich im allgemeinen durch spezielle Steuerbefehle sperren und öffnen (sie sind maskierbar). Bei mehreren Eingängen besteht eine Vorrangordnung. Kommen an 2 Eingängen die INTERRUPT-Signale gleichzeitig an, so hat das Signal mit der höheren Priorität den Vorrang. Das andere Signal kann vorgemerkt oder ignoriert werden.

2.8. STACK-Organisation

Der STACK (auch Stapelspeicher genannt) ist ein Teil des Arbeitsspeichers, der so organisiert wird, wie man Gegenstände in einem Keller ablagert, d. h., die Informationen werden hineingestapelt. Die Information, die zuletzt hineingebracht wurde, muß man auch als erste wieder herausnehmen. Man sagt, die Speicherung geschieht nach dem Prinzip „last in first out".
Zur Adressierung des STACK dient ein spezielles Register, der STACKPOINTER (Stackzeiger). Für die STACK-Adressierung gelten im allgemeinen folgende Regeln:

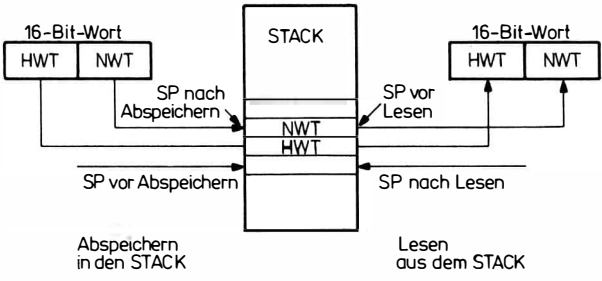


Bild 2.14 Arbeitsweise des STACK-Speichers

- Der STACKPOINTER (SP) zeigt auf das zuletzt eingeschriebene oder auf das nächste zu lesende Byte.
- Der STACK wird von höheren Adressen nach niedrigeren Adressen beschrieben.
- Die Abspeicherung (als PUSH bezeichnet) und das Lesen (als POP bezeichnet) geschieht meistens mit einem 16-Bit-Wort (2 Byte).

Aus Bild 2.14 ist zu ersehen, in welcher Weise ein 16-Bit-Wort in den STACK abgespeichert wird und wie es aus dem STACK gelesen wird, wenn eine Speicherzelle 8 Bit (1 Byte) speichern kann.

2.9. Zusammenstellung der Funktionseinheiten eines Rechners

Aus Bild 2.15 sind die wichtigsten Funktionseinheiten eines Rechners zu ersehen. Für die rein elektronischen Funktionseinheiten (Rechenwerk, Steuerwerk, Speicher, E/A-Steuerung, DMA-Steuerung, INTERRUPT-Steuerung) gibt es spezielle programmierbare Schaltkreise. Dabei werden Rechenwerk und Steuerwerk im Mikroprozessorschaltkreis vereinigt. Für die übrigen Funktionseinheiten gibt es spezielle Schaltkreise.
In Teil 2 sind Grundlagen und Arbeitsweise der programmierbaren Schaltkreise sowie einige Schaltkreise für die genannten Funktionseinheiten eines Rechners beschrieben.

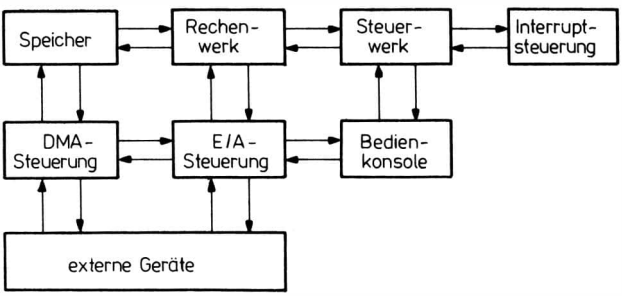


Bild 2.15 Funktionseinheiten eines Rechners

3. Aufbau und Arbeitsweise digitaler Schaltkreise für Mikrorechner

3.1. Übersicht

Die Digitaltechnik befaßt sich mit der Verarbeitung logischer Signale. Durch die Verfeinerung der Technologie war es im Laufe der Zeit möglich, immer mehr der für diese Verarbeitung notwendigen logischen Grundschaltungen auf einem Chip unterzubringen. Von den ersten SSI-Schaltkreisen, in denen einige Transistoren untergebracht sind, bis zu den LSI-Schaltkreisen mit 10^4 bis 10^5 Transistoren ging eine intensive technologische Entwicklung parallel.
Während bei den niedrigintegrierten Schaltkreisen das Spektrum von einfachen Grundschaltungen bis zu speziellen Schaltungen wie Zähler, Register, Decoder, Konverter, Multiplexer, Leitungstreiber reicht, geht die Entwicklung der hochintegrierten Schaltkreise in 2 Richtungen. Die eine Richtung ist der spezielle *Kundenschaltkreis*. Dazu gehören z. B. der Taschenrechnerschaltkreis und der Schaltkreis für Digitaluhren. Diese Schaltkreise realisieren eine spezielle Funktion. Die Schaltung ist fest vorgegeben und für die jeweilige Funktion weitestgehend optimal entwickelt.
Die zweite Richtung sind die *programmierbaren Schaltkreise*. Sie sind Bestandteile eines Rechners. Die Entwicklung verläuft zur Zeit in der Richtung, daß einmal die gesamte Zentraleinheit einschließlich der Ein- und Ausgabekanäle auf einem Chip integriert wird. So entstehen heute bereits Chips, die eine minimale Rechnerkonfiguration (CPU, Speicher, Ein-/Ausgabe-Kanäle) realisieren. Die folgende Zusammenstellung zeigt eine Übersicht existierender Schaltkreistypen.

Digitale Grundschaltkreise
- logische Grundfunktionen
- Flip-Flop-Schaltungen
- Treiber

Analoge Grundschaltkreise
- Operationsverstärker
- NF-Verstärker

- ZF-Verstärker
- Netzregelschaltkreise

MSI-Schaltkreise der Digitaltechnik
- Taktgeneratoren
- Codierer und Decodierer
- Register
- Zähler
- Multiplexer
- Rechenschaltungen
- Speicher
- Bustreiber

MSI-Schaltkreise der Analogtechnik
- Analog-Digital-Umsetzer
- Digital-Analog-Umsetzer

LSI-Kundenschaltkreise
- Taschenrechnerschaltkreise
- Schaltkreise für Digitaluhren

Programmierbare LSI-Schaltkreise
(Schaltkreise der Mikrorechentechnik)
- Mikroprozessoren
- Ein-/Ausgabe-Schaltkreise
- INTERRUPT-Schaltkreise
- Schaltkreise zum direkten Speicherzugriff (DMA-Schaltkreise)

3.2. Schaltalgebra

Eine Reihe von Grundfunktionen, z. B. die Entschlüsselung einer Adresse, müssen meistens durch logische Grundschaltkreise aufgebaut werden. Zum Aufstellen solcher Schaltungen ist die Schaltalgebra eine große Hilfe. Sie ist die Grundlage für das Fällen von Entscheidungen und für das Prüfen von logischen Bedingungen. Mit den Binärziffern 0 und 1 werden sogenannte *logische Operationen* durchgeführt. Zu den logischen Operationen gehören:
die ODER-Funktion bzw. Disjunktion (OR),
die UND-Funktion bzw. Konjunktion (AND),
das EXKLUSIV-ODER (XOR)
und die Nicht-Funktion bzw. NEGATION (NOT).

Die ODER-Funktion

Die ODER-Funktion wird für 2 Binärziffern A und B folgendermaßen definiert:

Wenn A oder B oder beide gleich 1 sind, so ist das Ergebnis gleich 1. Andernfalls ist das Ergebnis gleich 0.

Zur Darstellung der ODER-Funktion wird das Symbol „+" bzw. „$\vee$" (*vel* – lat., oder) genommen.

Es gelten die Kombinationen:
$0 \vee 0 = 0$,
$0 \vee 1 = 1$,
$1 \vee 0 = 1$,
$1 \vee 1 = 1$.

Logische Funktionen werden mit Hilfe einer Funktionstabelle definiert. Die Funktionstabelle enthält die Ausgangssignale, die sich bei den zulässigen Kombinationen der Eingangssignale ergeben. Die Funktionstabelle für die ODER-Funktion ist aus Tabelle 3.1. zu ersehen.

UND-Funktion

Die UND-Funktion wird für 2 Binärziffern A und B folgendermaßen definiert:

Wenn A und B beide gleich 1 sind, so ist das Ergebnis gleich 1. Andernfalls ist das Ergebnis 0.

Zur Darstellung der UND-Funktion wird das Symbol „$\cdot$" bzw. „$\wedge$" (et – lat., und) genommen. Es ist auch üblich, daß 2 Binärziffern ohne Operationszeichen aneinandergeschrieben werden:
$A \cdot B \triangleq A \wedge B \triangleq AB$

Tabelle 3.1 Funktionstabelle für die ODER-Funktion

Eingänge		Ausgang
A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 3.2. Funktionstabelle für die UND-Funktion

Eingänge		Ausgang
A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Es gelten die Kombinationen
$0 \wedge 0 = 0$,
$0 \wedge 1 = 0$,
$1 \wedge 0 = 0$,
$1 \wedge 1 = 1$.
Die Funktionstabelle für die UND-Funktion ist in Tabelle 3.2. dargestellt.

Das EXKLUSIV-ODER

Beim EXKLUSIV-ODER ist das Ergebnis der Verknüpfung von A und B gleich 1, wenn die Eingangssignale verschieden sind, und 0, wenn die Eingangssignale gleich sind.
Für das EXKLUSIV-ODER wird das Symbol $\oplus$ bzw. das Symbol $\veebar$ verwendet. Es gelten folgende Kombinationen:
$0 \oplus 0 = 0$,
$0 \oplus 1 = 1$,
$1 \oplus 0 = 1$,
$1 \oplus 1 = 0$.

Tabelle 3.3 Funktionstabelle für das EXKLUSIV-ODER

Eingänge		Ausgang
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Die Funktionstabelle für das EXKLUSIV-ODER ist aus Tabelle 3.3. zu ersehen.

Die Nicht-Funktion (NEGATION)

Die Nicht-Funktion bezieht sich nur auf eine Binärziffer. Sie wird durch einen Querstrich über die betreffende Ziffer dargestellt. ($\overline{A}$ heißt NEGATION von A.)

Bei der NEGATION ist das Ergebnis 1, wenn A gleich 0 ist, und 0, wenn A gleich 1 ist.

Es gilt:

$\overline{1} = 0$,

$\overline{0} = 1$.

Tabelle 3.4. zeigt die Funktionstabelle für die NEGATION.

Rechenregeln für logische Operationen

a) Die Negation einer negierten Größe ergibt die Größe selbst:

(1) $\overline{\overline{a}} = a$.

b) Für die logischen Operationen mit 2 Größen gelten folgende Vertauschungsgesetze (Kommutativgesetz):

(2) $a \vee b = b \vee a$,

(3) $a \wedge b = b \wedge a$,

(4) $a \oplus b = b \oplus a$.

Insbesondere gilt:

$1 \vee a = 1$,

$1 \wedge a = a$,

$0 \vee a = a$,

$0 \wedge a = 0$,

$a \vee \overline{a} = 1$,

$a \wedge \overline{a} = 0$.

Tabelle 3.4. Funktionstabelle für die NEGATION

Eingang	Ausgang
A	$\overline{A}$
0	1
1	0

c) Für zusammengefaßte Ausdrücke gilt das Distributivgesetz:

(5) $(a \vee b) \vee c = a \vee (b \vee c)$,

(6) $(a \wedge b) \wedge c = a \wedge (b \wedge c)$,

(7) $a \wedge b \vee c \quad = (a \vee c) \wedge (b \vee c)$.

Weiterhin gilt:

(8) $\overline{a \wedge b} = \bar{a} \vee \bar{b}$,

(9) $\overline{a \vee b} = \bar{a} \wedge \bar{b}$,

(10) $\bar{a} \vee ab = \bar{a} \vee b$,

(11) $a \vee \bar{a}b = a \vee b$.

Die Regeln (1) bis (11) lassen sich überprüfen, wenn man jeweils für die rechte und linke Seite die Funktionstabelle aufstellt und dabei die Funktionstabellen für die UND-Funktion, die ODER-Funktion, die NEGATION und für das EXKLUSIV-ODER verwendet.

Mit Hilfe der genannten Umformungsregeln kann man logische Ausdrücke wesentlich vereinfachen. Bei der technischen Realisierung lassen sich Schaltungen optimieren und damit Bausteine einsparen.

Beispiel

Der logische Ausdruck

$T = ad \vee a\bar{c} \vee bd \vee bc \vee \bar{a}b \vee \overline{ab}$

läßt sich wie folgt vereinfachen:

$T = ad \vee a\bar{c} \vee bd \vee bc \vee \bar{a}$,

$T = d \vee \bar{a} \vee bc \vee \bar{c}$,

$T = d \vee \bar{a} \vee b \vee \bar{c}$.

Ein sehr häufig vorkommendes Problem ist die Aufstellung eines

Tabelle 3.5. Funktionstabelle für die Addition von 3 Dualziffern (2 Summanden a, b und Übertrag u)

a	b	u	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

logischen Ausdrucks bei vorgegebener Funktionstabelle. Will man z. B. die Summe von 3 Dualziffern bilden (2 Operanden a und b und einen Übertrag u vom niedrigen Stellenwert), so kann die Funktionstabelle 3.5 aufgestellt werden. Die Summe s ist 1, wenn eine Ziffer 1 oder alle 3 Ziffern 1 sind. Sind alle 3 Ziffern 1, so tritt ein Übertrag in die nächste Stelle auf. Sind 2 Ziffern 1, dann ist die Summe 0 und der Übertrag in die nächste Stelle 1.
s ist hier eine logische Funktion von den 3 Größen a, b und u. Aus Tabelle 3.5. kann man ablesen, daß s = 1 ist, wenn

a = 0; b = 0; u = 1
oder
a = 0; b = 1; u = 0
oder
a = 1; b = 0; u = 0
oder
a = 1; b = 1; u = 1.

Statt a = 0 kann man auch sagen, es muß $\bar{a} = 1$ sein, d. h., es ist
s = 1, wenn
$\bar{a} = 1$; $\bar{b} = 1$; u = 1 bzw. $\bar{a}\,\bar{b}\,u = 1$
oder
$\bar{a} = 1$; b = 1; $\bar{u} = 1$ bzw. $\bar{a}\,b\,\bar{u} = 1$
oder
a = 1; $\bar{b} = 1$; $\bar{u} = 1$ bzw. $a\,\bar{b}\,\bar{u} = 1$
oder
a = 1; b = 1; u = 1 bzw. a b u = 1.

Den Ausdruck $\bar{a}\,\bar{b}\,u$ nennt man die Elemenartarkonjunktion P_1 (a, b, u). Sie ist nur für die Belegung 0 0 1 (a = 0; b = 0; u = 1) gleich 1, sonst 0. Ebenso ist die Elementarkonjunktion $P_2(a, b, u) = \overline{abu}$ nur für die Belegung 0 1 0 (a = 0; b = 1; u = 0) gleich 1, sonst 0.
Die ODER-Funktion der Elementarkonjunktion $\overline{ab}u$, $ab\bar{u}$, $a\overline{bu}$ und abu ist damit gerade dort 1, wo s nach Tabelle 3.5 gleich 1 ist. Sonst ist diese ODER-Funktion 0. Danach läßt sich schreiben:
$s = \bar{a}bu \vee \bar{a}b\bar{u} \vee a\bar{b}\bar{u} \vee abu$.
Durch Anwendung des Distributivgesetzes (7) erhält man:
$s = u(\overline{ab} \vee ab) \vee \bar{u}\,(\bar{a}b \vee a\bar{b})$.

Tabelle 3.6. Tabelle der Elementarkonjunktionen Pj

	a_n	a_{n-1}	$a_{n-2} \ldots a_2$	a_1	Elementarkonjunktionen
Belegung 0	0	0	0...0	0	$P_0 = \bar{a}_n \bar{a}_{n-1} \bar{a}_{n-2} \ldots \bar{a}_2 \bar{a}_1$
Belegung 1	0	0	0...0	1	$P_1 = \bar{a}_n \bar{a}_{n-1} \bar{a}_{n-2} \ldots \bar{a}_2 a_1$
Belegung 2	0	0	0...1	0	$P_2 = \bar{a}_n \bar{a}_{n-1} \bar{a}_{n-2} \ldots \bar{a}_2 \bar{a}_1$
.			.		.
.			.		.
.			.		.
.			.		.
Belegung 2^n-2	1	1	1...1	0	$P^n_{2-2} = a_n a_{n-1} a_{n-2} \ldots a_2 \bar{a}_1$
Belegung 2^n-1	1	1	1...1	1	$P^n_{2-1} = a_n a_{n-1} a_{n-2} \ldots a_2 a_1$

Mathematische Zusammenfassung

Liegt eine Funktionstabelle mit n Größen $a_1 a_2 \ldots a_n$ vor und ist T die Ergebnisfunktion, so bilde man zunächst alle Elementarkonjunktionen P_j, für deren Belegung j T = 1 ist (Tabelle 3.6.). Die ODER-Funktion dieser Elementarkonjunktion P_j ist die logische Funktion T:

$$T = \bigvee_{j=0}^{2^n-1} Q_j P_j.$$

Dabei ist $Q_j = \begin{cases} 0, \text{ wenn T für die Belegung } j \text{ gleich 0 ist;} \\ 1, \text{ wenn T für die Belegung } j \text{ gleich 1 ist.} \end{cases}$

Durch Vereinfachung des Ausdrucks für T mit Hilfe der Rechenregeln für logische Funktionen erhält man die gewünschte Funktion.

Beispiel

Ein Baustein soll mit der Adresse 14 ausgewählt werden. Die Gesamtadresse ist 16 Bit lang. Das Auswahlsignal CS (*c*hip-*s*elect) ist nur dann aktiv (= 1), wenn auf den Adreßleitungen eine 14 steht:

Adreß-
ltgn.: $A_{15}\,A_{14}\,A_{13}\,A_{12}\,A_{11}\,A_{10}\,A_9\,A_8\,A_7\,A_6\,A_5\,A_4\,A_3\,A_2\,A_1\,A_0$
Adres-
se 14: 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
CS =
$P_{14} = \overline{A}_{15}\,\overline{A}_{14}\,\overline{A}_{13}\,\overline{A}_{12}\,\overline{A}_{11}\,\overline{A}_{10}\,\overline{A}_9\,\overline{A}_8\,\overline{A}_7\,\overline{A}_6\,\overline{A}_5\,\overline{A}_4\,A_3\,A_2\,A_1\,\overline{A}_0.$

3.3. Logische Grundschaltkreise

Die Verarbeitung von Binärsignalen basiert auf wenigen Grundschaltungen. Die wichtigsten Grundschaltungen sind die Realisierung des logischen UND, des logischen ODER und der NEGATION. Diese Funktionen werden in den meisten Schaltkreisreihen durch NOR- oder NAND-Schaltungen gebildet. Der Leser informiere sich an dieser Stelle ausführlicher über die Schaltkreisserie *D 10*, z. B. in der Reihe *electronica* (Heft 141, 155, 156).
Eine sehr wichtige Eigenschaft in der Mikroelektronik ist die Möglichkeit, Schaltkreise ausgangsseitig zusammenzuschalten. Diese Eigenschaft haben nicht alle Schaltkreise. Eine Form der Zusammenschaltung ist der Wired-OR-verdrahtete Ausgang mit offenem Kollektor (Bild 3.1). Dabei erhalten Kollektorausgänge aller Schaltkreise einen gemeinsamen Arbeitswiderstand. Der Nachteil dieser Schaltung besteht im hohen Stromverbrauch, da der Arbeitswiderstand um so kleiner sein muß, je mehr Schaltkreise ausgangsseitig parallelgeschaltet werden.
Eine zweite Möglichkeit der ausgangsseitigen Parallelschaltung bietet die Tri-state-Schaltung. Bei dieser Schaltung kann der Ausgang neben den Pegeln H und L noch einen dritten hochohmigen Zustand einnehmen. Der hochohmige Zustand wird erreicht, wenn sowohl der obere als auch der untere Ausgangstransistor sperren (Tiefpegel am Steuereingang $\overline{X}$; Bild 3.2).

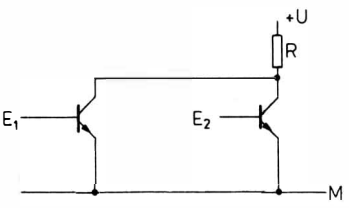


Bild 3.1
Parallelschaltung von Ausgängen nach dem Wired-OR-Prinzip

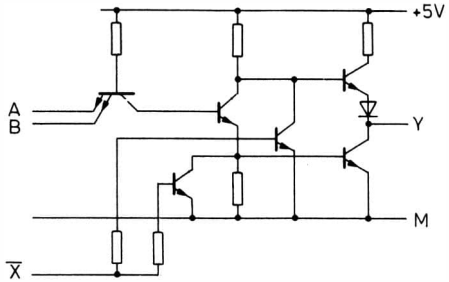


Bild 3.2 Erzeugung des hochohmigen Zustandes am Ausgang Y eines Schaltkreises durch Sperren beider Ausgangstransistoren (L-Pegel am Steuereingang $\overline{X}$)

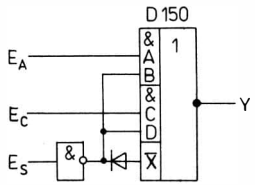


Bild 3.3
Tri-state-Steuerung am Baustein *D 150*

Zum Beispiel läßt sich der Baustein *D 150* in Tri-state-Schaltung betreiben, da über $\overline{X}$ ein Sperren des oberen Ausgangstransistors möglich ist, wenn $\overline{X}$ L-Pegel führt. Bild 3.3 zeigt die Ansteuerung des Bausteins *D 150* für den Tri-state-Betrieb. Für die Umschaltung in den hochohmigen Zustand ist der Steueranschluß E_s vorgesehen. Um den unteren Ausgangstransistor zu sperren, werden alle zweiten AND-Eingänge (B, D) auf L-Pegel gelegt. Damit ist der untere Ausgangstransistor gesperrt. Der obere Ausgangstransistor ist durch L-Pegel an $\overline{X}$ gesperrt.

3.4. Informationsspeicherung

3.4.1. Überblick

Die Grundeinheit der Information ist das *Bit*; die Grundschaltung zur Speicherung eines Bit das *Flip-Flop*. Werden mehrere Bit zusammengefaßt, so ergibt sich ein Bit-Muster. Das Bit-Muster

kann 1 Byte (8 Bit) oder Wort sein. Ein Byte oder Wort wird im *Register* gespeichert (zwischengespeichert). Zur Speicherung vieler Worte (den Daten eines Rechners) ist eine große Anzahl von Registern erforderlich. Sie bilden den *Speicher* eines Rechners.
Flip-Flop-Schaltungen sind die Grundlage aller Halbleiterspeicher. Über Aufbau und Arbeitsweise von Flip-Flop-Schaltungen informiere sich der Leser in [1]. Sie sollen an dieser Stelle nicht näher behandelt werden.

3.4.2. Register

Ein Register dient zur Speicherung eines Bit-Musters. Es besteht deshalb aus einer entsprechenden Anzahl von Flip-Flop. Je nach der zusätzlichen Funktion, die das Register noch zu erfüllen hat, werden dazu RS-Flip-Flop, D-Flip-Flop oder JK-Flip-Flop gewählt.
Registerschaltungen können auch einfache Operationsschritte ausführen, z. B. die gespeicherte Information um eine oder mehrere Stellen nach rechts oder links verschieben (Schieberegister).
Bild 3.4 zeigt ein einfaches Speicherregister aus RS-Flip-Flop für 4 Bit. Wenn das Torungssignal STR (Strobe) den Pegel H hat, wird die Information $E_3\ E_2\ E_1\ E_0$ im Register gespeichert. H-Pegel des Signals RESET löscht den Registerinhalt. Der Registerinhalt läßt sich an $A_3\ A_2\ A_1\ A_0$ abnehmen.
Bild 3.5 zeigt ein 4-Bit-Schieberegister aus JK-Flip-Flop. Die einzelnen Stellen des Registers können von außen parallel statisch oder dynamisch mit Hilfe der Eingänge E_0 bis E_3 gesetzt werden. Das ermöglicht eine parallele Eingabe. Über die Ausgänge A_0, $\overline{A_0}$ bis A_3, $\overline{A_3}$ ist eine parallele Ausgabe der Stellen des Registers möglich. Ein 4-Bit-Muster, das im Register steht, kann seriell über den Ausgang A, $\overline{A}$ ausgegeben werden. Der Schiebeprozeß wird durch den Takt T veranlaßt. Durch die mit jedem Takt ausgelöste Rechtsverbindung können auch über den Eingängen E, $\overline{E}$ neue Bit-Muster seriell eingegeben werden. Das Register ist somit für die Umsetzung parallel → seriell und seriell → parallel geeignet.
Mit Hilfe eines Dynamikvorsatzes (Kondensatorspeicher) läßt sich ein Schieberegister auch mit einem RS-Flip-Flop aufbauen.

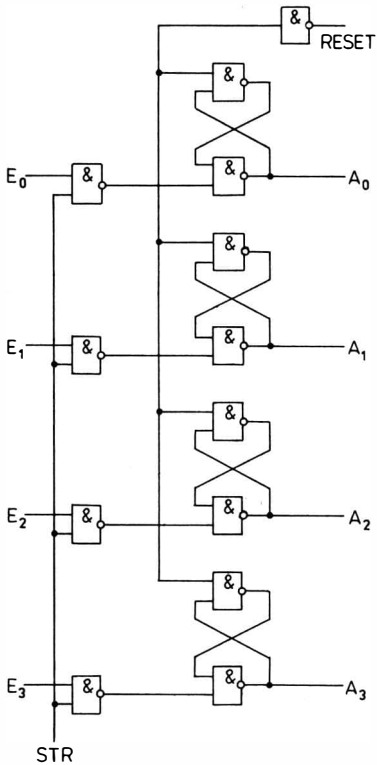


Bild 3.4
Speicherregister für
4 Bit aus RS-Flip-Flop

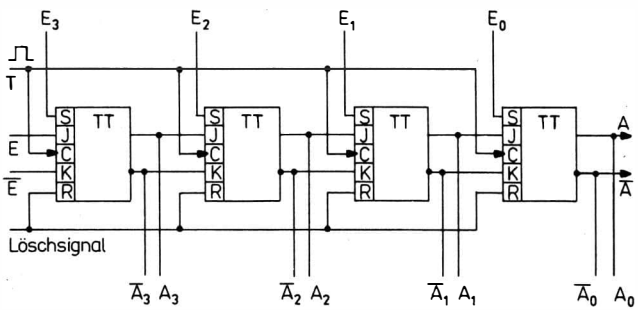


Bild 3.5 4-Bit-Schieberegister aus JK-Flip-Flop

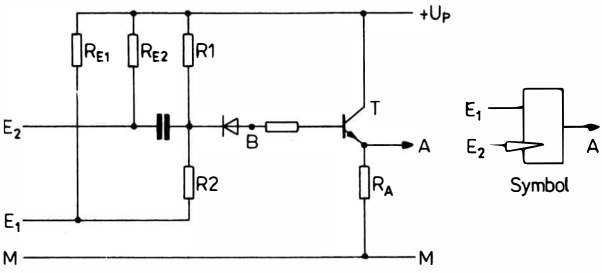


Bild 3.6 Dynamikvorsatz (Kondensatorspeicher) zur Realisierung von Schieberegistern mit RS-Flip-Flop

Bild 3.6 zeigt die Schaltung eines solchen Dynamikvorsatzes. Mit E_1 wird die Schaltung vorbereitet. Hat E_1 L-Potential, so steht an Punkt B eine kleine positive Spannung. Ein negativer Impuls bei E_2 sperrt den Transistor T kurzzeitig und erzeugt am Ausgang A einen L-Impuls. Hat E_1 H-Potential, so liegt bei B die volle Spannung U_p an. Ein negativer Impuls an E_2 bringt Transistor T nicht bis zur Sperrschwelle. Der Transistor T sperrt im Fall E_1 = L auch, wenn zum gleichen Zeitpunkt, zu dem der negative Impuls an E_2

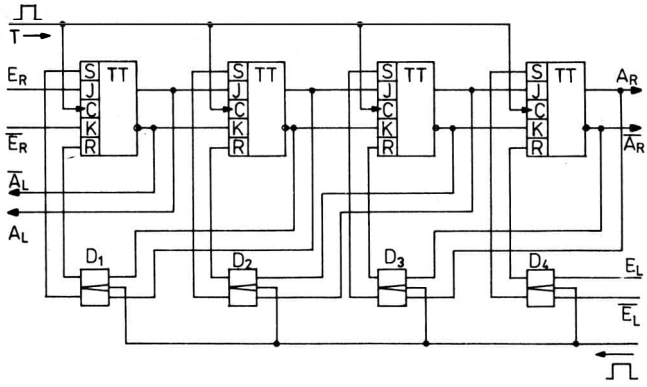


Bild 3.7 Zwei Richtungs-Schieberegister mit Dynamikvorsatz für die Linksverschiebung

erscheint, E_1 H-Pegel erhält, da das Potential bei B erst langsam über R2 umgeladen wird.
Bild 3.7 zeigt ein Zwei-Richtungs-Schieberegister für 4 Bit mit Kondensatorspeicher für die Linksverschiebung. Die Dynamikvorsätze D_1 bis D_4 ermöglichen ein kurzzeitiges Speichern der Information bei der Linksverschiebung.
Für den Aufbau von Mikrorechnern gibt es Schaltkreise, in die komplette Register integriert sind. Bild 3.8 zeigt die Schaltung des Schaltkreises *8212,* der zur Dateneingabe und -ausgabe verwendet werden kann. Er besteht aus einem 8-Bit-Register mit D-Flip-Flop und einer Steuerschaltung.

Arbeitsweise des Bausteins 8212

Der Schaltkreis *8212* arbeitet in 2 Betriebsarten, die durch MD (Mode) gesteuert werden.
Betriebsart 1: Eingang MD hat H-Pegel
Die Information der einzelnen Registerstellen kann direkt an den Ausgängen DO1 bis DO8 abgenommen werden. Wenn $\overline{DS1}$ L-Pegel und DS2 H-Pegel hat, dann wird die Information von den Eingängen DI1 bis DI8 in die D-Flip-Flop eingegeben.
Betriebsart 2: Eingang MD hat L-Pegel
Mit H-Pegel an STR (Strobe) wird die Information in die D-Flip-Flop gespeichert.
Wenn $\overline{DS1}$ L-Pegel und DS2 H-Pegel führt, so kann die Information, die im Register gespeichert ist, an den Ausgängen DO1 bis DO8 abgenommen werden.
Im Fall 2 setzt außerdem das Signal STR mit der HL-Flanke das Flip-Flop SRFF zurück und damit den $\overline{INT}$-Ausgang auf L-Pegel. Durch $\overline{DS1} \cdot DS2$ wird das Flip-Flop SRFF wieder gesetzt. Die Leitung $\overline{INT}$ dient als INTERRUPT-Anforderung für Mikroprozessorschaltkreise. Durch L-Pegel am Eingang $\overline{CLR}$ (Löschen) werden die D-Flip-Flop des Registers rückgesetzt und das Flip-Flop SRFF gesetzt ($\overline{INT}$ = H, d. h. nicht aktiv). Die Ausgänge DO1 bis DO8 sind mit einer Tri-state-Steuerung versehen, d. h., wenn die Ausgangsleitungen gesperrt sind, ist ihr Ausgang hochohmig.

Zusammenstellung der Funktionen des Bausteins 8212

MD = H: $DE \cdot \overline{DS1} \cdot DS2 \rightarrow$ Registerinhalt
DA = Registerinhalt

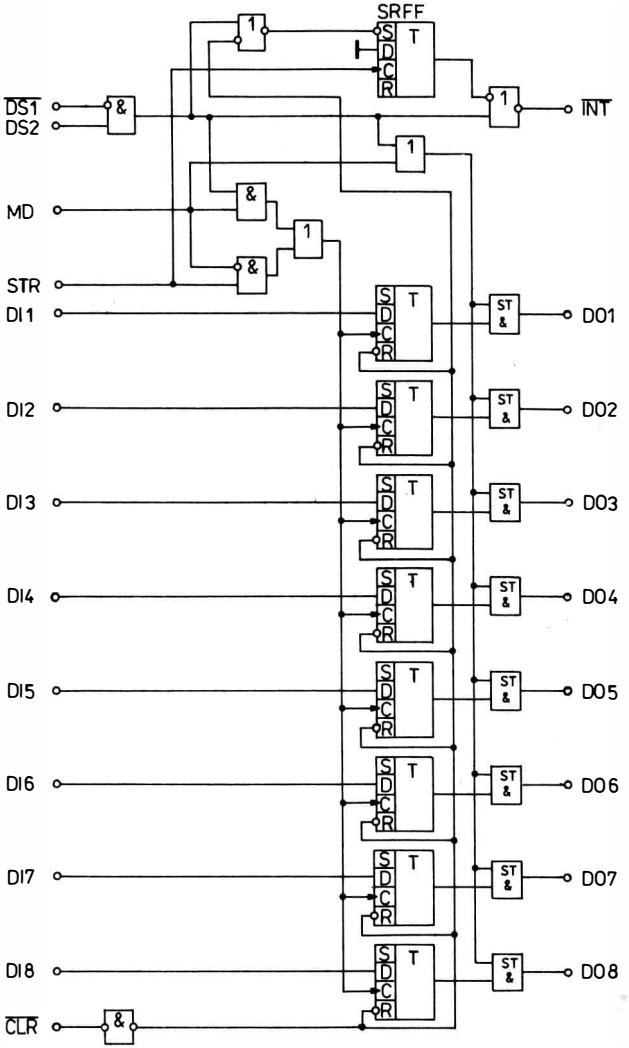


Bild 3.8 Schaltbild des Bausteins *8212*

MD = L: DE · STR → Registerinhalt, DA = Registerinhalt
· $\overline{DS1}$ · DS2
STR (H→T)→ $\overline{SRFF}$ → INTERRUPT-Anforderung
$\overline{CLR}$ = L: Löschen Register und Setzen SRFF
Das INT-Signal ist aktiv ($\overline{INT}$ = L), wenn
INT = $\overline{DS1}$ · DS2 ∨ $\overline{SRFF}$ = 1 ist.

Anwendung des Bausteins *8212*

Der Baustein *8212* läßt sich sehr universell für die Bussteuerung und Datenpufferung anwenden. Bild 3.9 zeigt den Einsatz des Bausteins als Datenpuffer. Die Eingabe in das Register geschieht mit $\overline{DS1}$ · DS2 = H. Die Ausgabe aus dem Register ist jederzeit möglich.

Bild 3.10 zeigt die Verschaltung des Bausteins *8212* als Eingabepuffer. Die Eingabe in das Register läßt sich über das Signal STR vornehmen. Das Signal STR aktiviert gleichzeitig das INT-Signal. Damit wird angezeigt, daß der Datenpuffer voll ist.

Die Ausgabe beginnt in dem Moment, wo $\overline{DS1}$ · DS2 = H ist. Mit diesem Signal wird gleichzeitig das SRFF-Flip-Flop gesetzt.

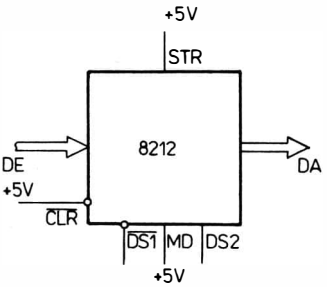


Bild 3.9
Der Baustein *8212* als Datenpuffer

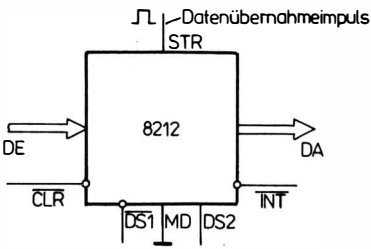


Bild 3.10
Der Baustein *8212* als Eingabetor

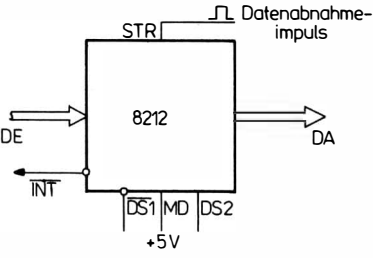


Bild 3.11
Der Baustein *8212* im Hand-shake-Prinzip

Die Ausgabe beginnt in dem Moment, wo $\overline{DS1} \cdot DS2 = H$ ist. Mit diesem Signal wird gleichzeitig das SEFF-Flip-Flop gesetzt. Das INT-Signal verschwindet jedoch erst mit der HL-Flanke von $\overline{DS1} \cdot DS2$.

Bild 3.11 zeigt die Verschaltung des Bausteins *8212* im Hand-shake-prinzip für einen Ausgabepuffer. Die Eingabe ins Register geschieht mit dem Signal $\overline{DS1} \cdot DS2 = H$. Der Inhalt des Registers liegt ständig an DA. Den zur Abnahme des Datenwortes verwendeten Strobeimpuls führt man gleichzeitig an den Strobeeingang, wodurch das INT-Signal aktiv wird (als Rückmeldung, daß die Daten abgeholt wurden).

In Tabelle 3.7 sind einige Registerschaltkreise zusammengestellt.

Tabelle 3.7. Zusammenstellung einiger Registerschaltkreise

Typ		Kurzcharakteristik
MH 7475	ČSSR	4-Bit-D-Register mit Enable
74174 PC	UVR	6-Bit-D-Flip-Flop, löschbar
74175 PC	UVR	4-Bit-D-Flip-Flop, löschbar
74259 PC	UVR	8-Bit-Register, adressierbar
K 500 TM 131		4-Bit-D-Register (ECL),
D 191 C/D	DDR	8-Bit-Schieberegister
D 195 C/D	DDR	4-Bit-rechts/links-Schieberegister
MH 7496	ČSSR	5-Bit-Schieberegister, E/A parallel mit Löschung
MH74164		8-Bit-Schieberegister Eingabe seriell; Ausgabe parallel
74165 PC 74166 PC	UVR	8-Bit-Schieberegister Eingabe seriell; Ausgabe parallel
K 155 IR 13	UdSSR	8-Bit-rechts/links-Schieberegister
74194 PC	UVR	4-Bit-rechts/links-Schieberegister
74195 PC	UVR	4-Bit-Schieberegister Eingabe parallel; Ausgabe parallel

3.4.3. Speicher

Während ein Register dazu dient, 1 Bit-Muster zu speichern, hat der Speicher die Aufgabe, eine größere Anzahl Bit-Muster (Worte genannt) aufzunehmen. Im Laufe der Entwicklung der Rechentechnik wurden die unterschiedlichsten Speicherprinzipien entwikkelt (Magnettrommelspeicher, Laufzeitspeicher, Speicherbildröhre, Ringkernspeicher, Halbleiterspeicher). In der Mikrorechentechnik wird ausschließlich der Halbleiterspeicher verwendet. Deshalb soll auch nur auf ihn eingegangen werden.
Speicherschaltkreise kann man nach der Art der Anwendung oder nach der Herstellungstechnologie einteilen. Nach der Art der Anwendung gibt es

ROM (*R*ead *O*nly *M*emory – Nur-Lesespeicher),
PROM (*P*rogrammable *R*ead *O*nly *M*emory – Programmierbarer ROM),
EPROM (*E*rasable *PROM* – Löschbarer und programmierbarer ROM),
RAM (*R*andom *A*ccess *M*emory – Schreib-Lese-Speicher).

Nach der Herstellungstechnologie unterscheidet man Speicherschaltkreise in
TTL-Technik,
p-MOS-Technik,
n-MOS-Technik,
CMOS-Technik,
I^2L-Technik,
Eine Beschreibung der unterschiedlichen Techniken findet der Leser in [2].

RAM (Schreib-Lese-Speicher)

Statische Schreib-Lese-Speicher
Bei den statischen RAM bilden 2 rückgekoppelte NICHT-Gatter die Speicherzelle, wobei den beiden Zuständen die Information 0 oder 1 zugeordnet wird.
Aus technologischer Sicht gibt es solche Speicher in TTL-Technik und MOS-Technik. Speicher in TTL-Technik sind sehr schnell, sie haben aber einen großen Leistungs- und Platzbedarf. MOS-Bausteine erfordern weniger Platz. Die Schaltkreise in p-Kanal-MOS-Technik sind sehr langsam und benötigen mehrere Betriebsspan-

nungen. Heute werden MOS-Schaltkreise fast ausschließlich in n-Kanal-Technik ausgeführt. Moderne MOS-Schaltungen sind voll TTL-kompatibel. Bei der CMOS-Technik bestehen wesentlich günstigere Betriebsspannungen (3 bis 15 V). Ihre Leistungsaufnahme liegt wesentlich unter der der TTL-Technik. Bild 3.12 zeigt den Aufbau einer Speicherzelle in TTL-Technik, Bild 3.13 den Aufbau einer Speichermatrix mit 16 Speicherzellen.

Um einen Speicherplatz innerhalb der Speichermatrix anzusteuern, müssen die X- und Y-Leitung H-Signal führen. Damit wird die Speicherzelle aktiviert, bei der X- und Y-Leitung gleichzeitig H-Signal haben. In der adressierten Speicherzelle fließt der Kollektorstrom des leitenden Transistors in die entsprechende Schreib-Lese-Leitung L oder H. An den Ausgängen der Schreib-Lese-Leitungen befinden sich Leseverstärker (LV), die aus dem Kollektorstrom ein logisches L (Q_L) oder H (Q_H) bilden.

Zum Einschreiben einer Information wird nach der Adressierung der Speicherzelle auf eine der Schreib-Lese-Leitungen ein H-Signal gegeben. Zur Speicherung einer 0 wird auf die L-Leitung ein H-Signal (Eingang E_L) und zur Speicherung einer 1 auf die H-Leitung (Eingang E_H) ein H-Signal gegeben.

Die in Bild 3.13 dargestellte Anordnung der Flip-Flop ist in den TTL-Bausteinen *7481* und *7484* realisiert. Es kann immer nur 1 Bit gelesen bzw. eingeschrieben werden. Um eine 0 einzuschreiben, muß man den E_L-Eingang ansteuern, zum Einschreiben einer 1 den E_H-Eingang. Die gelesene Information liegt an Q_L bzw. Q_H.

Bild 3.14 zeigt den Baustein *7489*. Er hat eine 16 × 4-Struktur,

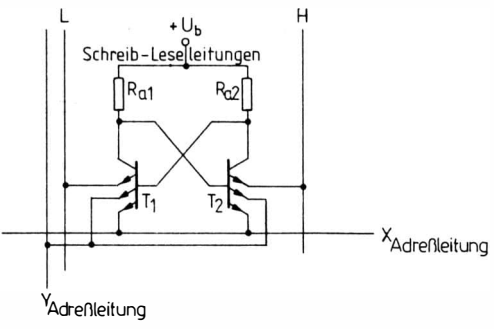


Bild 3.12 Aufbau einer TTL-Speicherzelle

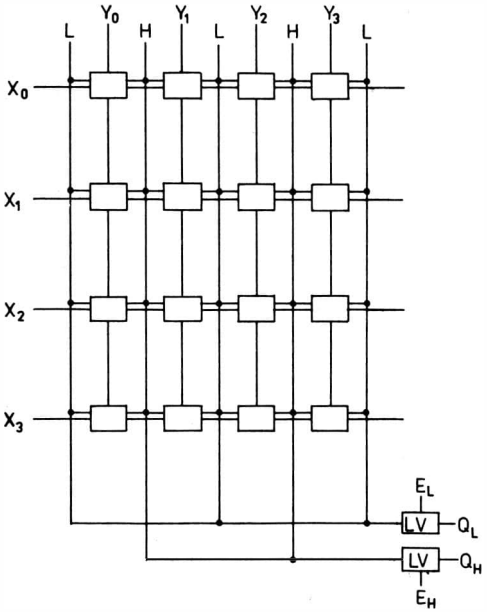


Bild 3.13 TTL-Speichermatrix mit 16 Speicherzellen

d. h. eine Speicherkapazität von 16 Wörtern zu je 4 Bit. Der Baustein *7489* ist ein voll-decodiertes RAM.

Schreibvorgang: Wenn die Freigabeeingänge $\overline{CS}$ (chip-select, Bausteinauswahl) und $\overline{WR}$ (Write) auf L-Signal liegen, wird die an den Dateneingängen D_1 bis D_4 bereitgestellte Information in die durch die Adreßeingänge A_0 bis A_3 angesteuerten Speicherzellen gebracht. Die Y-Leitungen tragen dabei die negierte Information, die auch gleichzeitig an $\overline{Q}_1$ bis $\overline{Q}_4$ anliegt. Die Flip-Flop schalten mit L-Signal auf den Y-Leitungen und mit H-Signal auf den X-Leitungen.

Lesevorgang: Wenn $\overline{CS}$ L-Signal und $\overline{WR}$ H-Signal erhält, so liegt an den Ausgängen $\overline{Q}_1$ bis $\overline{Q}_4$ der negierte Inhalt der durch A_0 bis A_3 adressierten Speicherzelle.

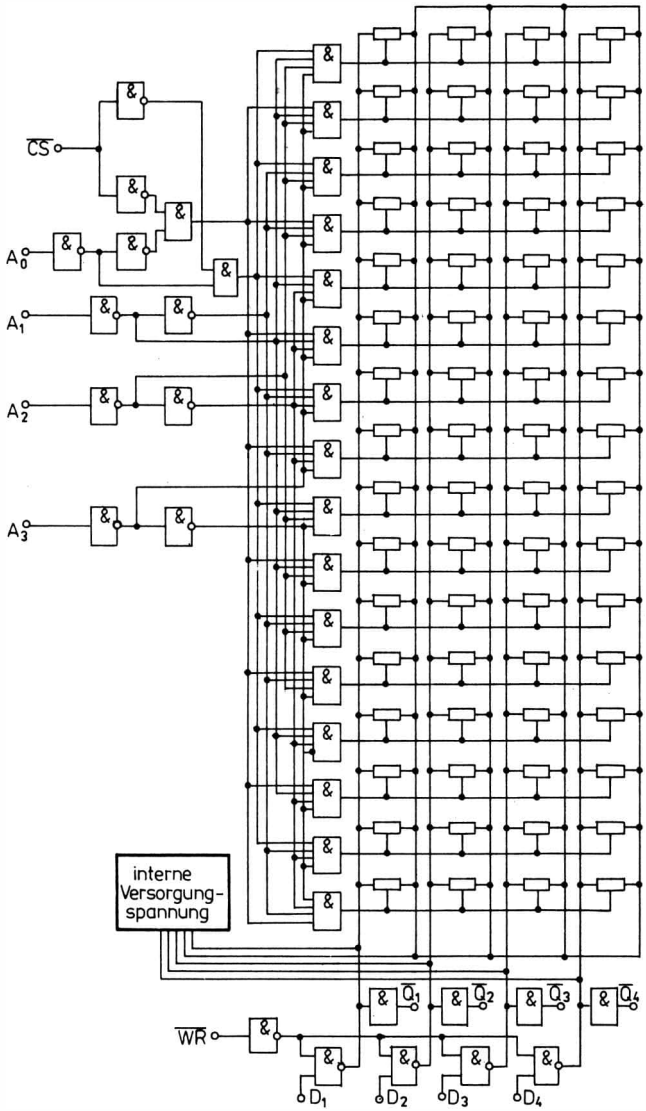


Bild 3.14 Logischer Aufbau des Speicherbausteins *7489*.

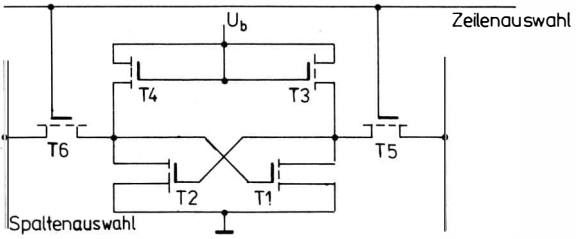


Bild 3.15 Aufbau einer MOS-Speicherzelle

Statische Speicher in MOS-Technik
Bild 3.15 zeigt die Prinzipschaltung einer MOS-Speicherzelle für statische RAM. Die Lastwiderstände sind wegen des geringen Platzbedarfs durch Transistoren T_3, T_4 ersetzt.

Lesevorgang: Wenn die Transistoren T_5 und T_6 über die Zeilenauswahl durchgesteuert sind, dann kann die Information als Signalunterschied an den Spaltenleitungen abgenommen werden.

Schreibvorgang: Leiten T_5 und T_6, so läßt sich das Flip-Flop über die Spaltenleitungen in den gewünschten Zustand setzen.
Bild 3.16 zeigt das Prinzip eines Halbleiterspeichers mit 256 Flip-Flop aus p-Kanal-MOS-Transistoren. Die Flip-Flop sind in einer Ebene so angeordnet, daß jederzeit über 16 X- und Y-Koordinaten ein direkter Zugriff zu jedem Flip-Flop besteht. Wenn an den Koordinaten X_m, Y_m das Potential $-U_{DD}$ anliegt, werden die Transistoren T_5 bis T_8 durchgesteuert.
Zur Speicherung einer „1" liegen z. B. an SL_2 18 V über T_9 (T_9 ist geöffnet) und an SL_1 0 V über R_3 und R_4 (T_{10} ist gesperrt). Dadurch leitet T_3, und T_4 sperrt. Die durch T_1 und T_2 gebildeten Lastwiderstände halten diese Stellung auch nach Abtrennen der Adresse. Zur Speicherung einer „O" haben SL_2 0 V und SL_1 18 V Potential. Beim Lesen sind T_9 und T_{10} geöffnet. Die Widerstände R_1, R_2 und R_3, R_4 liegen zu den Lastwiderständen T_1 und T_2 parallel. Durch R_2 bzw. R_4 fließt Strom entsprechend den Potentialen bei A und B. Der Spannungsabfall über R_2 und R_4 wird über einen Differenzverstärker als Informationssignal abgenommen.
In weiterentwickelten Speicherschaltkreisen sind die Adreßdecodierung sowie eine Bausteinauswahllogik enthalten.

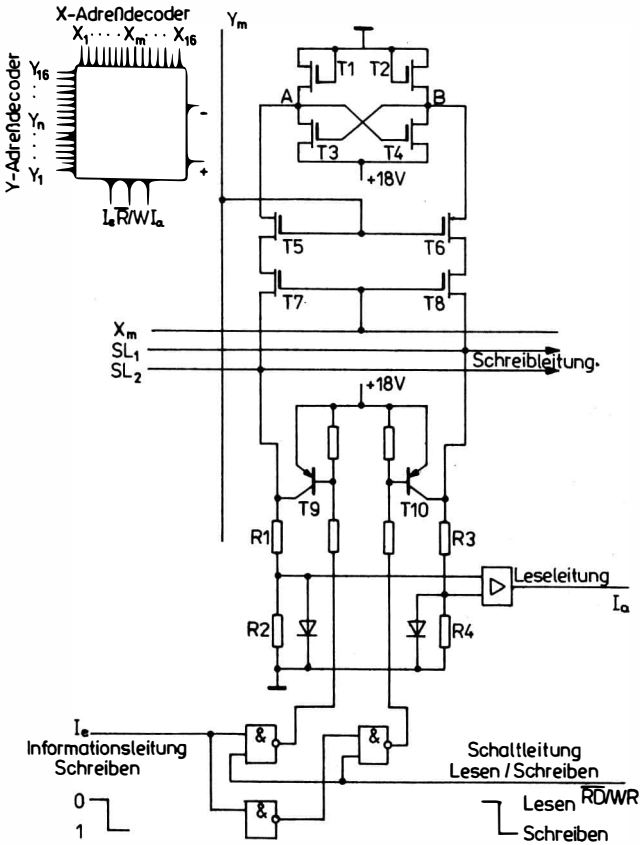


Bild 3.16 Halbleiterspeicher mit 256 Flip-Flop aus MOS-p-Kanal-Transistoren (negierte Logik)

Bild 3.17 zeigt den Aufbau eines 256 × 1 volldecodierten statischen MOS-RAM. Der Baustein wird mit $\overline{CS}$ = L angesteuert. Die Zeilenauswahl geschieht über die Adreßeingänge A_0 bis A_3, die Spaltenauswahl über A_4 und A_7. Hat R/$\overline{W}$ L-Signal, so wird die an I liegende Information eingeschrieben. Bei H-Signal an R/$\overline{W}$ erscheint an $\overline{O}$ der invertierte Inhalt der ausgewählten Speicherzelle. Mit H-Signal an $\overline{CS}$ ist der Speicher blockiert.

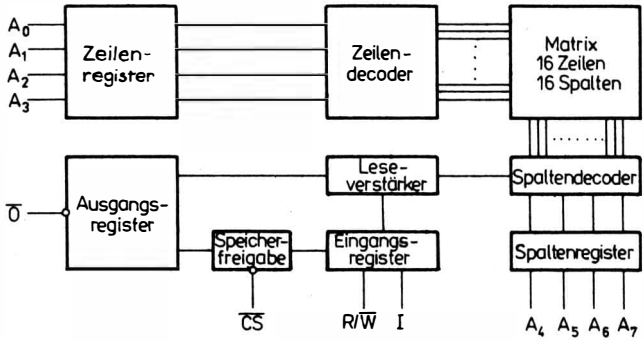


Bild 3.17 MOS-RAM mit 256 × 1 Zellen

Bild 3.18 zeigt den Aufbau eines 256 × 4-RAM. Über A_0 bis A_4 werden die Zeilen und über A_5 bis A_7 die Spalten ausgewählt. Bei L-Signal an $R/\overline{W}$ lassen sich die Daten auslesen. Die Information erscheint bei O_1 bis O_4. Bei H-Signal an $R/\overline{W}$ werden die an I_1 bis I_4 liegenden Informationen in die adressierte Speicherzelle eingeschrieben. Liegt an OD H-Signal, so sind die Ausgänge $\overline{O_1}$ bis $\overline{O_4}$ hochohmig. $\overline{CS}$ dient zur Chipwahl. Mit L-Signal an $\overline{CS}$ arbeitet der Speicherbaustein.

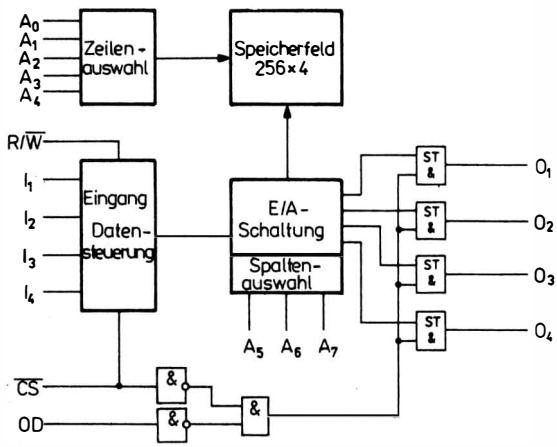


Bild 3.18 MOS-RAM mit 256 × 4 Zellen

Dynamische Schreib-Lese-Speicher

Bei den dynamischen RAM wird die Information durch Aufladung eines Kondensators gespeichert. Bild 3.19 zeigt eine Ein-Transistor-Speicherzelle. Sobald T_1 leitet, erscheint die Information auf der Datenleitung. Wegen der auftretenden Leckströme muß die Kondensatorladung periodisch (ungefähr alle 2 ms) regeneriert (wieder aufgeladen) werden.

Die ersten dynamischen RAM waren den statischen RAM angelehnt. Bild 3.20 zeigt den Aufbau einer solchen Speicherzelle. Aus Bild 3.21 ist der heutzutage meistens verwendete Aufbau einer dynamischen RAM-Zelle zu ersehen.

Zum Einschreiben muß die Schreibleitung H-Signal haben und die Datenleitung die Information (L- oder H-Signal) tragen. Über T_3 wird C_1 entsprechend aufgeladen.

Zum Lesen muß die Lese-Auswahlleitung H-Signal führen (T_2 leitet). Hat C_1 H-Signal, so wird T_1 leitend, und auf der Datenleitung entsteht L-Potential. Hat C_1 L-Potential, dann bleibt der Weg T_2-T_1-Substrat hochohmig, und die Datenleitung erhält H-Potential. Auf der Datenleitung entsteht das negierte Informationssignal der Speicherzelle. Lese-Auswahlleitung und Schreibleitung bestimmen meistens die Zeilenauswahl, die Datenleitung bestimmt die

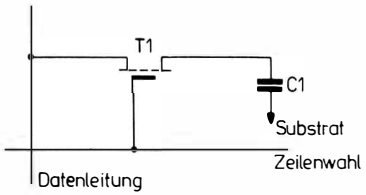


Bild 3.19
Aufbau einer dynamischer RAM-Speicherzelle mit einem Transistor

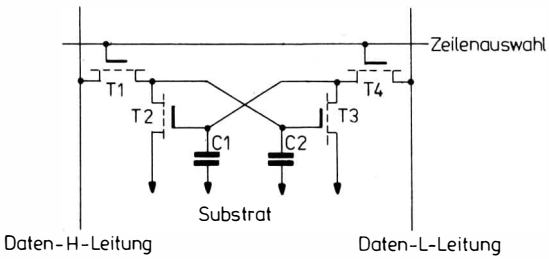


Bild 3.20 Dynamische RAM-Speicherzelle nach dem Flip-Flop-Prinzip

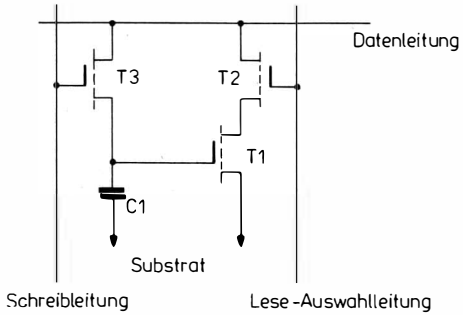


Bild 3.21 Dynamische RAM-Speicherzelle

Spaltenauswahl. Aus Bild 3.22 ist der Aufbau eines dynamischen 4 K × 1-RAM zu ersehen.

In Bild 3.22 bedeuten:

A_0 bis A_{11} Adreßeingänge;

I – Dateneingang

$R/\overline{W}$-Schreibfreigabe ($R/\overline{W} = L \triangleq$ Schreiben; $R/\overline{W} = H \triangleq$ Lesen);

$\overline{O}$ – Datenausgang;

$\overline{CS}$ – Bausteinauswahl ($\overline{CS} = L \triangleq$ Freigabe);

CE – Speicherfreigabe (chip enable)

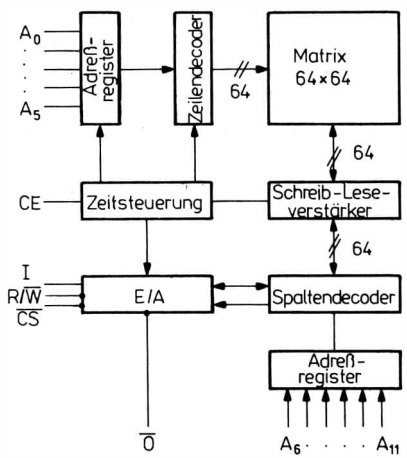


Bild 3.22 Dynamischer 4 K × 1-RAM

Jede einzelne Zeile muß innerhalb von 2 ms einen Auffrischzyklus erhalten. Während dieser Zeit darf nicht neu eingeschrieben werden. Das ist möglich bei
einem Lesezyklus: $R/\overline{W}$ = H, $\overline{CS}$ beliebig;
einem Schreibzyklus: $R/\overline{W}$ = L, dann muß $\overline{CS}$ H-Potential haben.
Der Eingang CE schaltet den Speicher im nichtaktiven Zustand (L-Pegel an CE) auf geringen Leistungsverbrauch. Er beträgt im nichtaktiven Zustand ≈ 1,3 mW, zusätzlich ≈ 7 mW für das Auffrischen.

Nur-Lese-Speicher (ROM)

Ein ROM (*R*ead *O*nly *M*emory) ist ein vom Hersteller programmierbarer Festwertspeicher. Der Inhalt eines programmierten ROM läßt sich nicht mehr ändern. Der so hergestellte Dateninhalt kann nur ausgelesen werden.
Bild 3.23 zeigt den Aufbau eines Masken-ROM. Die Basiselektroden, die die Verbindung zur Zeilenauswahl herstellen, werden durch Wegätzen einer SiO_2-Schicht, die zwischen einem Metallband (Zeilenleitungen) und dem Substrat liegt, gebildet. Dazu verwendet man eine Maske, die an jeder Programmierstelle ein Programmierungsloch hat.
Die Ausgabe erfolgt über die Spaltenleitungen. Für jeden Ausgang existiert ein Leseverstärker, der nur arbeiten kann, wenn am CE-Eingang H-Signal anliegt. Damit sperren die 4 unteren Transistoren (p-Kanal-MOS-Transistoren), und es kann kein Strom gegen Masse fließen. Soll Zeile 2 angewählt werden, dann muß die Zeilenleitung L-Signal bekommen. Die MOS-Transistoren, deren Basiselektrode vorhanden ist, bilden jetzt eine leitende Verbindung zur Masse. Der fließende Strom wird in den Leseverstärkern in ein logisches Signal umgewandelt. In Spalte 1, 3 und 4 kann über Zeile 2 ein solcher Strom fließen. Die Leseverstärker von Q_1, Q_3 und Q_4 erkennen deshalb ein L-Signal, der Leseverstärker Q_2 dagegen ein H-Signal.

Programmierbare ROM (PROM)

PROM können vom Anwender mit Hilfe eines Programmiergerätes programmiert werden. Die Programmierung geschieht mit elektrischen Impulsen.

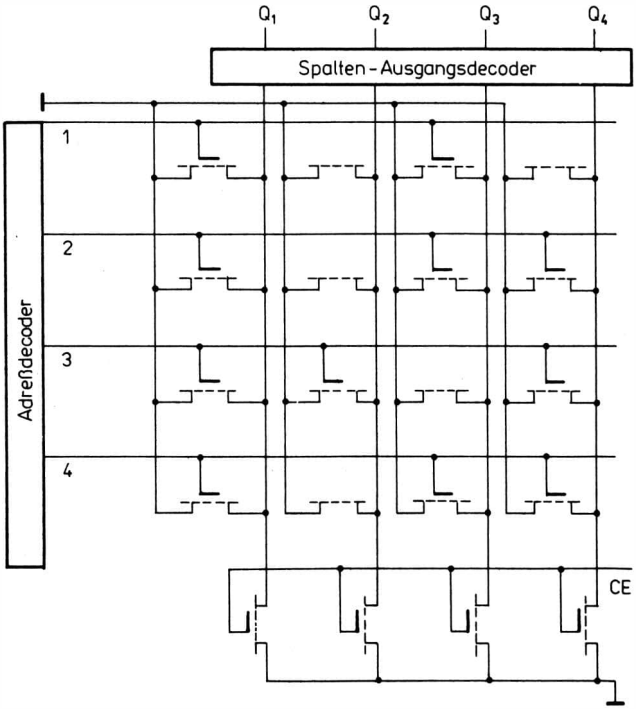


Bild 3.23 Aufbau eines Masken-ROM

MOS-PROM

Bild 3.24 zeigt den Aufbau einer MOS-PROM-Speicherzelle. In einem n-Substrat liegen die beiden p-Zonen. Bei dem Programmiervorgang wird der Drainanschluß der Speicherzelle mit Masse verbunden und der Source-Anschluß an eine Spannungsquelle angeschlossen. Bei niedriger Spannung fließt durch die Speicherzelle nur ein kleiner Strom. Bei höherer Spannung wird der Strom größer. Er verläßt die p-Zone der Source und fließt über die Metallschicht. Dabei kommt es zu einer Wanderung von Metallatomen. Durch diese Wanderung entsteht zwischen den p-Zonen und der Metallschicht eine elektrisch leitende Verbindung. Der Programmierungsstrom beträgt etwa 150 mA.

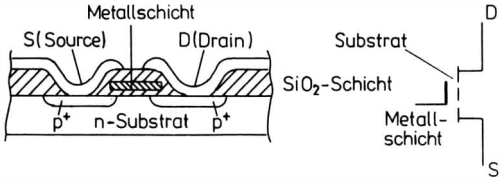


Bild 3.23 Aufbau eines Masken-ROM

TTL-Dioden-PROM

Bild 3.25 zeigt eine Diodenmatrix. Im unprogrammierten Zustand sind die Dioden bereits vorhanden, aber nicht angeschlossen. Erst durch Wegätzen einer SiO_2-Schicht werden die Katoden der Dioden mit den Y-Leitungen elektrisch verbunden.

An den Y-Achsen ist durch die Leseverstärker an jeder Leitung ein H-Signal vorhanden. Durch die Adressierung wird auf eine X-Achse ein L-Signal gelegt. Über die angeschlossenen Dioden kann ein Strom gegen Masse fließen. Die Leseverstärker wandeln diesen Strom in das logische Signal um.

Bild 3.26 zeigt den Aufbau eines 32 × 8-TTL-Dioden-PROM. Eine andere Möglichkeit zur Herstellung von TTL-Dioden-PROM ist das Fusable-link-Verfahren. Bild 3.27 zeigt den Aufbau eines FL-PROM. Im unprogrammierten Zustand sind zwischen den Dioden und den Y-Leitungen Brücken in Form eines Sicherungsdrahts (fusable-link). Die Speicherzelle wird durch Abschmelzen dieser Verbindung mit Hilfe eines Stromimpulses programmiert.

Ein anderes Verfahren zur Herstellung von programmierbaren

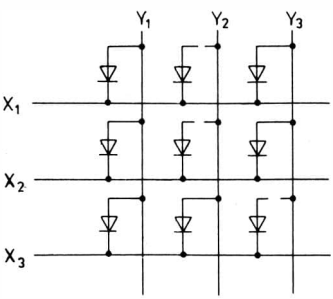


Bild 3.25
PROM-Dioden-Matrix

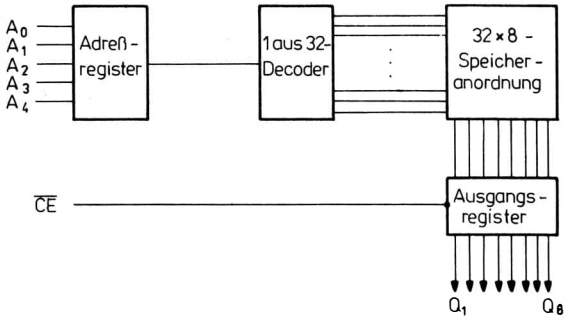


Bild 3.26 32 × 8-TTL-Dioden-PROM

Festwertspeichern ist das AIM-Verfahren (*A*valanche-*I*nduced-*M*igration). Die Matrix besteht aus einem Transistorfeld (Bild 3.28). Bei der unprogrammierten Speicherzelle ist die Basis nicht angeschlossen. Beim Programmieren wird die Basis-Emitter-Diode der zu programmierenden Speicherzelle mit Stromstößen abgebaut. Sie hat im unprogrammierten Zustand einen Widerstand von etwa 100 kΩ, im programmierten Zustand von etwa 200 Ω (Bild 3.29). Die Stromstöße zur Programmierung betragen 200 mA bei einer Spannung von 32 V und 7,5 μs Dauer. Diese Stromstöße werden so lange wiederholt, bis der Widerstand der Speicherzelle unter 200 Ω liegt.

Löschbare Festwertspeicher

(EPROM – *E*rasable PROM, REPROM – *R*eversible PROM).
Für das Löschen verwendet man ultraviolettes Licht. Dabei wird der gesamte Speicherinhalt gelöscht. Nach dem Löschen läßt sich der Speicher neu programmieren.

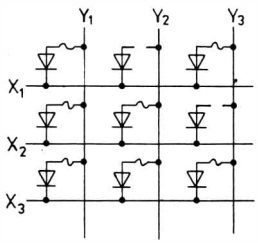


Bild 3.27
Aufbau eines PROM nach dem Fusable-link-Verfahren

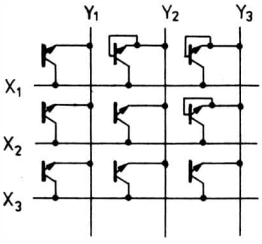


Bild 3.28
Aufbau eines PROM nach dem AIM-Verfahren

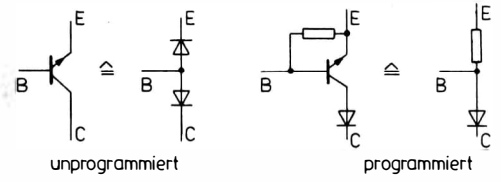


Bild 3.29 Programmierung einer PROM-AIM-Zelle

Bild 3.30 zeigt den Aufbau einer löschbaren Speicherzelle. Ansteuerungsgate und Floating-Gate sind Poly-Siliziumschichten. Das Ansteuerungsgate wird mit den Zeilenleitungen verbunden. Das Floating-Gate dient zur Ladungsspeicherung. Beim Programmieren wird über die Spaltenleitungen durch Injektion energiereicher Elektronen das Floating-Gate aufgeladen. Die gespeicherte Ladung verändert die Schwellwertspannung der Zelle. Bild 3.31 zeigt den logischen Aufbau eines EPROM.

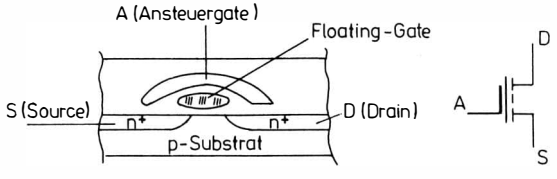


Bild 3.30 Aufbau einer mit UV-Licht löschbaren PROM-Speicherzelle

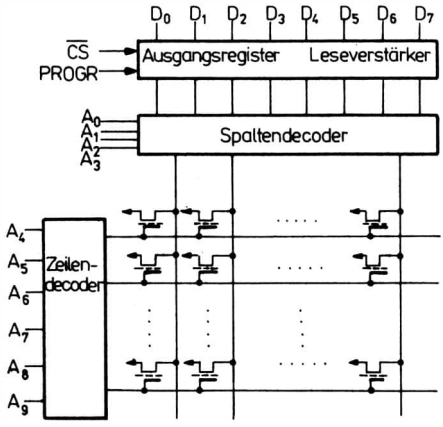


Bild 3.31 Aufbau eines EPROM

Zur Programmierung wird $\overline{CS}$ auf +12 V gelegt. An die Adreßeingänge A_0 bis A_9 wird wie beim Lesen die Wortadresse angelegt. Die Programminformation kommt an die Datenleitung D_0 bis D_7. Nachdem Adressen und Daten eingestellt sind, wird ein Programmierungsimpuls je Adresse an den PROGR-Eingang gegeben. Das einmalige Durchlaufen aller Adressen bezeichnet man als *Programmierschleife*. Die Anzahl der Programmierschleifen ist von der Impulsdauer des PROGR-Signals abhängig.

Das Löschen geschieht mit UV-Licht über vorhandene Quarzfenster. Bei einigen EPROM wird eine Wellenlänge von 253,7 nm (2537 Å) bei einer Dosis von 10 Ws/cm^{-2} und einer Löschzeit von 20 bis 30 min angegeben. Dem Strahler darf kein Filter vorgesetzt sein. Durch die UV-Strahlung werden die Elektronen vom Floating-Gate gelöst.

Außer mit UV-Licht löschbare PROM gibt es noch PROM, die sich mit einer hohen Spannung löschen lassen (VEPROM – *V*oltage *e*rasable *PROM*). Diese Bausteine haben anstelle des Quarzfensters eine Metallplatte mit einem Anschlußpin. Durch Anlegen einer hohen Spannung von etwa 10 min Dauer mit umgekehrten Vorzeichen gegenüber der Spannung beim Programmieren werden die Elektronen aus dem Floating-Gate herausgezogen.

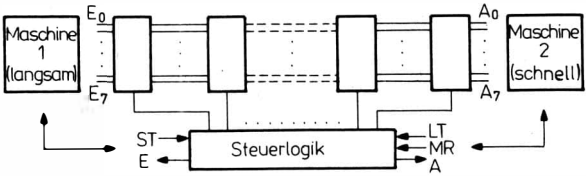


Bild 3.32 Pufferspeicherprinzip

Pufferspeicher

Eine spezielle Speicherlogik stellt der Pufferspeicher nach dem Prinzip „first in, first out" dar. Mit ihm lassen sich zwei mit unterschiedlicher Geschwindigkeit arbeitende Maschinen anpassen. Bild 3.32 zeigt die grundsätzliche Struktur eines solchen Speichers. Die Daten werden über E_0 bis E_7 mit Hilfe des Schreibtaktes ST eingeschrieben. Durch das Signal E meldet der Speicher, daß er leere Zellen hat. Wenn der Speicher gefüllt ist, wird das durch das Signal A gemeldet. Ist das Signal A aktiv, so können die Daten mit Hilfe des Lesetaktes LT ausgelesen werden. Das Signal A wird inaktiv, wenn der Speicher leer ist oder wenn Maschine 2 ein Anschlußsignal MR (*m*aster *r*eset) abgibt. Durch MR wird der Speicher gelöscht, A inaktiv und E aktiv. Dann kann ein neues Einschreiben beginnen. Mit jedem Signal ST und LT wird der Inhalt des Speichers um 1 Zelle nach rechts geschoben.

3.4.4. Zusammenstellung einiger Speicherschaltkreise

ROM-Schaltkreise

U 501 D

ROM-Speicherschaltkreis mit 256 × 8-Organisation (Bild 3.33)

Der Speicher hat einen Chip-enable-Eingang ($\overline{CE}$). Bei L-Signal an $\overline{CE}$ gibt der Baustein den Inhalt der an den Adreßleitungen A_0 bis A_7 adressierten Speicherzellen an D_0 bis D_7 ab.

8308

ROM-Schaltkreis mit 1024 × 8-Organisation (Bild 3.34)

Der Schaltkreis hat 2 Eingänge, $\overline{CS_1}$ und CS_2 zur Bausteinauswahl. Mit $\overline{CS_1} \cdot CS_2 = H$ wird die durch A_0 bis A_9 adressierte Speicherzelle gelesen. Ihr Inhalt erscheint an D_0 bis D_7.

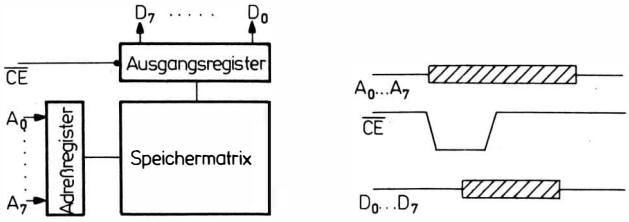


Bild 3.33 ROM-Speicherschaltkreis *U 501 D*

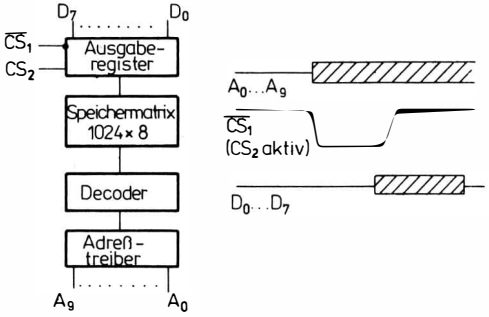


Bild 3.34 ROM-Speicherschaltkreis *8308*

8702 A

EPROM-Schaltkreis mit 256 × 8-Organisation (Bild 3.35)

Der Schaltkreis 8702 A ist ein löschbarer und elektrisch programmierbarer ROM. Der Speicher hat einen Eingang $\overline{CS}$ (chip-select). Wenn an $\overline{CS}$ L-Potential liegt, wird der Inhalt der durch die Adreßleitung A_0 bis A_7 adressierten Speicherzelle an D_0 bis D_7 bereitgestellt.

Der Schaltkreis wird durch Anlegen eines Programmimpulses am Eingang PROGR programmiert. Dabei erhalten die Adreßleitungen und Stromversorgungsleitungen impulsförmig höhere Spannungen als im Arbeitsbetrieb.

Das Löschen geschieht mit UV-Licht. Dazu muß der Baustein etwa 10 bis 20 min mit UV-Licht der Wellenlänge 253,7 nm (2537 Å) mit einer Intensität von 6 WS · cm^{-2} bestrahlt werden.

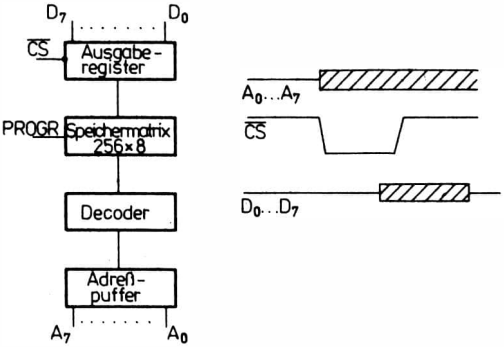


Bild 3.35
EPROM-Speicherschaltkreis *8702 A*

1702 A

EPROM-Schaltkreis mit 256 × 8-Organisation (Bild 3.36)

Der Schaltkreis *1702 A* ist ein löschbarer und elektrisch programmierbarer ROM. Der Speicher hat einen Eingang $\overline{CS}$ (chip-select). Wenn an $\overline{CS}$ L-Potential liegt, dann wird der Inhalt der durch die Adreßleitung A_0 bis A_7 adressierten Speicherzelle an D_0 bis D_7 bereitgestellt.

Tabelle 3.8. Anschlüsse des Bausteins 1702 A beim Lesen und Programmieren

Anschluß	Lesen	Programmieren
1-3, 17-21	Adreß-Bits	
4-11	Daten-Bits	
12	5 V	0 V
13	5 V	Progr.-Imp. – 48 V
14	0 V	0 V
15	5 V	12 V
16	−9 V	−35 V ... −40 V
22, 23	5 V	0 V
24	−9 V	− 9 V

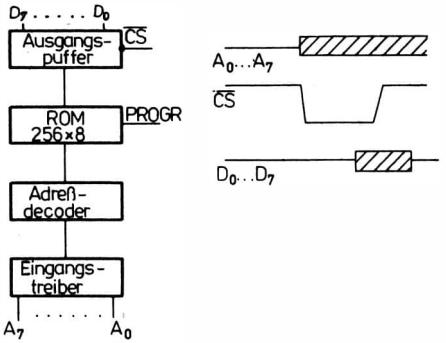


Bild 3.36 EPROM-Speicherschaltkreis *1702 A*

Zum Programmieren wird an den Eingang PROGR ein Impuls gelegt. Vor diesem Programmierimpuls muß die Adreßleitung die Adresse der zu programmierenden Speicherzelle und an die Datenleitungen das Programmwort gelegt werden.
Aus Tabelle 3.8. sind die Anschlüsse des Bausteins *1702 A* beim Lesen und Programmieren zu ersehen. Das Löschen geschieht mit UV-Licht der Wellenlänge 253,7 nm (2537 Å).

8708

EPROM-Schaltkreis mit 1024 × 8-Organisation (Bild 3.37)

Der Schaltkreis *8708* ist ein löschbarer und elektrisch programmierbarer ROM. Das Lesen geschieht in der Weise, daß der Eingang $\overline{CS}$/WE auf L-Potential gelegt und an die Adreßleitung A_0 bis

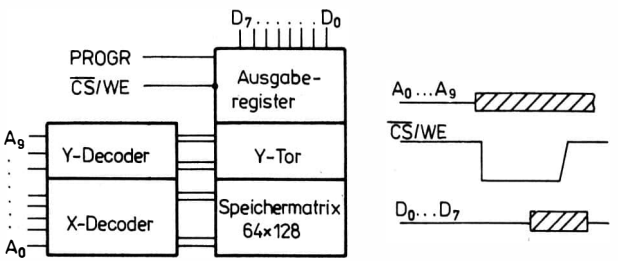


Bild 3.37 EPROM-Speicherschaltkreis *8708*

A_9 die Adresse der gewünschten Speicherzelle gegeben wird. An D_0 bis D_7 kann der Inhalt der Speicherzelle abgenommen werden. Zum Programmieren des Schaltkreises wird an den Eingang $\overline{CS}$/WE eine Spannung von 12 V und an den Eingang PROGR ein Impuls von 26 V gelegt. Den Speicherschaltkreis löscht man mit UV-Licht der Wellenlänge 253,7 nm (2537 Å) mit einer Intensität von 10 Ws · cm^{-2} in einer Zeitdauer von 20 bis 30 min.

RAM-Schaltkreise

2102

RAM-Schaltkreis mit 1024 × 1-Organisation (Bild 3.38)

Der Schaltkreis hat einen Bausteinauswahleingang $\overline{CS}$. Bei $\overline{CS}$ = L ist der Baustein angesteuert. Durch den Anschluß R/$\overline{W}$ läßt sich der Schaltkreis zwischen Lesen und Schreiben umschalten. Beim Schreiben (R/$\overline{W}$ = L) wird die an DI anliegende Information gespeichert. Bei Lesen (R/$\overline{W}$ = H) ist die gespeicherte Information an DO abnehmbar.

CM 8001

Statischer RAM-Schaltkreis mit 256 × 1-Organisation (Bild 3.39)

Das Lesen des Schaltkreises geschieht dadurch, daß an den Eingang $\overline{CS}$ L-Potential gelegt wird. Die Information erscheint am Ausgang DO und in negierter Form an $\overline{DO}$.

Zum Beschreiben des Bausteins muß $\overline{R}$/W = H sein und die Information an den Eingang DI gelegt werden.

U 253 D

Dynamischer RAM-Schaltkreis mit 1024 × 1-Organisation (Bild 3.40)

Der Schaltkreis *U 253 D* ist ein dynamischer Schreib-Lese-Speicher. Zum Lesen muß der Eingang $\overline{CS}$ L-Potential haben und die

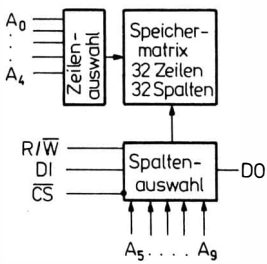


Bild 3.38
RAM-Speicherschaltkreis *2102*

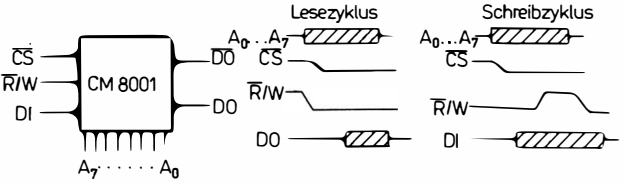


Bild 3.39 Statischer RAM-Speicherschaltkreis *CM 8001*

Adresse an A_0 bis A_9 gelegt werden. Der Eingang $\overline{WE}$ bleibt auf H-Potential. Die gelesene Information liegt negiert am Ausgang $\overline{DO}$ an. Zum Einschreiben ($\overline{WE}$ = L) legt man die Information am Eingang DI an.

Zum Auffrischen wird die ausgewählte Zelle gelesen und gleichzeitig wieder eingeschrieben. Dazu müssen alle 32 Zeilen der Speichermatrix, die von den Adressen A_0 bis A_4 angesteuert werden, innerhalb von 2 ms mindestens einmal gelesen worden sein. Die Belegung der Adressen A_5 bis A_9 (Spalten) ist dabei ohne Bedeutung.

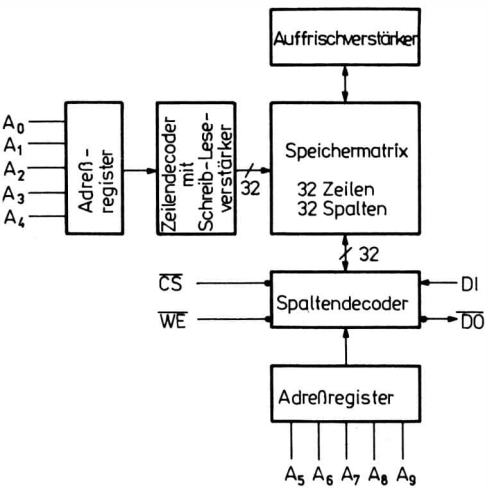


Bild3.40 Dynamischer RAM-Speicherschaltkreis *U 253 D*

Tabelle 3.9. Zusammenstellung einiger Speicherschaltkreise

Typ		Kurzcharakteristik
MSI-Schaltkreise		
D 181 C/D	DDR	16 × 1-RAM
K 155 RU 3	UdSSR	16 × 1-RAM
K 155 RU 2	UdSSR	16 × 4-RAM
MH 74188	ČSSR	32 × 8-PROM
MH 74 S 201	ČSSR	256 × 1-RAM mit Tri-state-Ausgang
74200PC	UVR	256 × 1-RAM mit Tri-state-Ausgang
K 155 RP 1	UdSSR	4 × 4-Bit-Register-Stapel
LSI-Schaltkreise		
2716		2K × 8-UV-EPROM (nur 5 V)
2758		1K × 8-UV-EPROM (nur 5 V)
2708		1K × 8-UV-EPROM (+12V, +5V, −5V)
2704		512 × 8-UV-EPROM (+12V, +5V, −5V)
8101		256 × 4-RAM (+5V)
8111		256 × 4-RAM (+5V)
2112		256 × 4-RAM (+5V)
K 565 PY 1 A		dynamischer4K × 1-RAM (−5V, +5V, +12V)

3.5. Codier- und Decodierschaltungen

Umcodierungen werden beim Aufbau von Mikrorechnern sehr häufig gebraucht. Beispiele sind die Realisierung der Anzeige durch LED-Elemente, die Zuordnung von Ziffern- und Funktionstasten zu den entsprechenden Zahlendarstellungen und die Entschlüsselung von dualen Adressen. Für viele dieser Funktionen

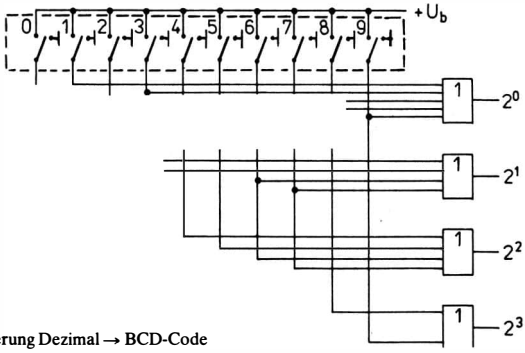


Bild 3.41 Codierung Dezimal → BCD-Code

gibt es integrierte Schaltkreise. Stehen keine speziellen Schaltkreise zur Verfügung, so lassen sich entsprechende Schaltungen auch mit logischen Grundschaltkreisen aufbauen.
Durch die Codierschaltung wird eine Darstellungsform für Zahlen oder Zeichen in eine andere Darstellungsform umgewandelt.
Bild 3.41 zeigt eine einfache Codierung Dezimalstellung → BCD-Code. Eine Dezimalziffer wird durch den ihr zugeordneten Schalter dargestellt. Beim BCD-Code bildet man duale Bit-Kombinationen mit 4 Dualstellen.
Aus Bild 3.42 ist die Schaltung des Bausteins *74147* zur Umwand-

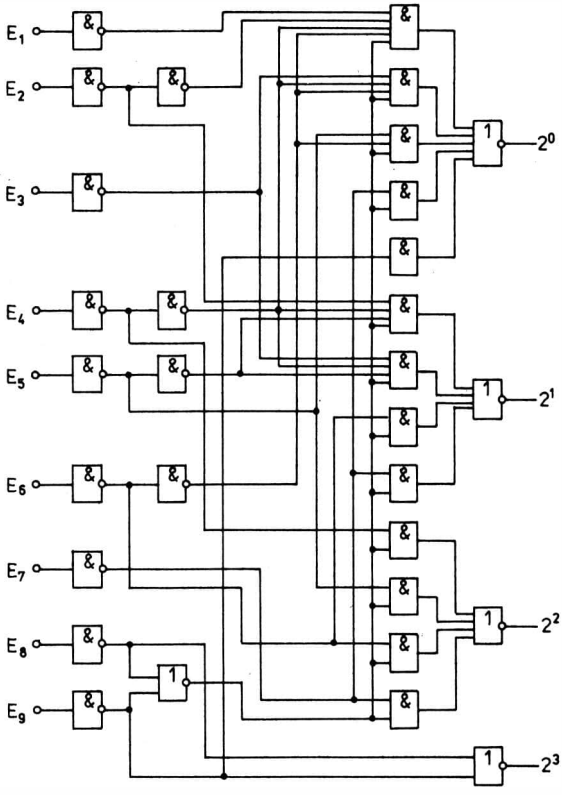


Bild 3.42 Aufbau des Bausteins *74147* zur Codierung Dezimal → BCD-Code

Tabelle 3.10. Funktionstabelle des Bausteins 74147 (Codierer: Dezimal → BCD)

E1	E2	E3	E4	E5	E6	E7	E8	E9	A3	A2	A1	A0
H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	L	L	H	H	L
H	H	H	H	H	H	H	L	H	L	H	H	H
H	H	H	H	H	H	L	H	H	H	L	L	L
H	H	H	H	H	L	H	H	H	H	L	L	H
H	H	H	H	L	H	H	H	H	H	L	H	L
H	H	H	L	H	H	H	H	H	H	L	H	H
H	H	L	H	H	H	H	H	H	H	H	L	L
H	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

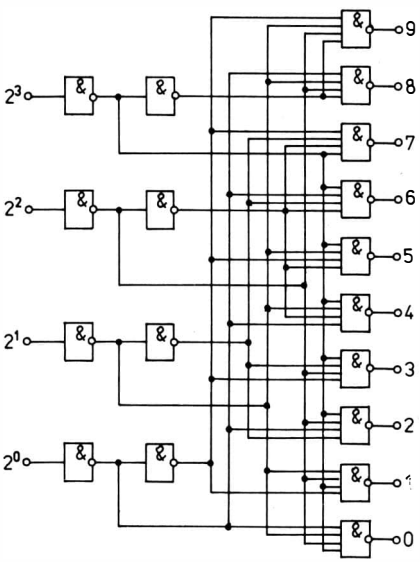


Bild 3.43
Aufbau des Bausteins *7442* zur Umwandlung BCD → Dezimal

lung einer Dezimaldarstellung 1 aus 10 in eine BCD-Darstellung mit 4 Bit und aus Tabelle 3.10. die Funktionstabelle des Bausteins *74147* zu ersehen.
Der Baustein *7442* (Bild 3.43) decodiert binäre Zahlen zu Dezimalzahlen.
In der Elektronik wird sehr häufig die 7-Segment-Anzeige verwendet. Aus Bild 3.44 ist die Zifferndarstellung bei der 7-Segment-Anzeige zu ersehen. Ein 7-Segment-Decoder muß aus dem BCD-Code die Signale zur Ansteuerung der 7 Segmente liefern.

Bild 3.44 Zifferndarstellung der Ziffern 0 bis 9 einer 7-Segment-Anzeige

Tabelle 3.11. Funktionstabelle eines 7-Segment-Decoders für die Dezimalziffern 0 bis 9

Ziffer	Eingänge				Segmente						
	D	C	B	A	a	b	c	d	e	f	g
0	L	L	L	L	H	H	H	H	H	H	L
1	L	L	L	H	L	H	H	L	L	L	L
2	L	L	H	L	H	H	L	H	H	L	H
3	L	L	H	H	H	H	H	H	L	L	H
4	L	H	L	L	L	H	H	L	L	H	H
5	L	H	L	H	H	L	H	H	L	H	H
6	L	H	H	L	H	L	H	H	H	H	H
7	L	H	H	H	H	H	H	L	L	L	L
8	H	L	L	L	H	H	H	H	H	H	H
9	H	L	L	H	H	H	H	H	L	H	H

Tabelle 3.11. zeigt die dazugehörige Funktionstabelle für die Ziffern 0 bis 9. Aus dieser Funktionstabelle ergibt sich über die Kanonische Alternative Normalform

$a = B \vee D \vee AC \vee \overline{AC}$;
$b = C \vee D \vee AB \vee \overline{AB}$;
$c = A \vee \overline{B} \vee C \vee D$;
$d = D \vee B\overline{C} \vee \overline{A}B \vee \overline{AC} \vee ABC$;
$e = \overline{AC} \vee \overline{A}B$;
$f = D \vee \overline{A}C \vee \overline{B}C \vee \overline{AB}$;
$g = D \vee B\overline{C} \vee \overline{A}B \vee \overline{B}C$.

Bild 3.45 zeigt die Schaltung eines 7-Segment-Decoders für die Ziffern 0 bis 9.

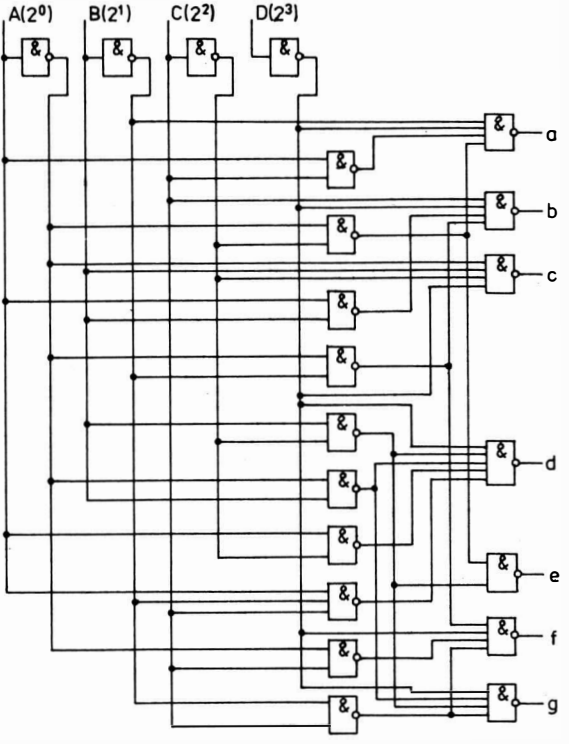


Bild 3.45 7-Segment-Decoder für die Ziffern 0 bis 9

In ähnlicher Weise läßt sich auch ein 7-Segment-Decoder für die Hexadezimalziffern 0 1 2 3 4 5 6 7 8 9 A B C D E F entwickeln. Bild 3.46 zeigt die Zeichendarstellung 0 bis F, Bild 3.47 den Aufbau des 7-Segment-Decoders, und aus Tabelle 3.12. ist die dazugehörige Funktionstabelle zu ersehen.

Tabelle 3.12. Funktionstabelle eines 7-Segment-Decoders für die Hexadezimalziffern 0 bis F

Ziffer	Eingänge				Segmente						
	D	C	B	A	a	b	c	d	e	f	g
0	L	L	L	L	H	H	H	H	H	H	L
1	L	L	L	H	L	H	H	L	L	L	L
2	L	L	H	L	H	H	L	H	H	L	H
3	L	L	H	H	H	H	H	H	L	L	H
4	L	H	L	L	L	H	H	L	L	H	H
5	L	H	L	H	H	L	H	H	L	H	H
6	L	H	H	L	H	L	H	H	H	H	H
7	L	H	H	H	H	H	H	L	L	L	L
8	H	L	L	L	H	H	H	H	H	H	H
9	H	L	L	H	H	H	H	H	L	H	H
A	H	L	H	L	H	H	H	L	H	H	H
B	H	L	H	H	L	L	H	H	H	H	H
C	H	H	L	L	H	L	L	H	H	H	L
D	H	H	L	H	L	H	H	H	H	L	H
E	H	H	H	L	H	L	L	H	H	H	H
F	H	H	H	H	H	L	L	L	H	H	H

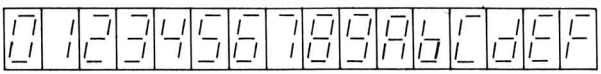
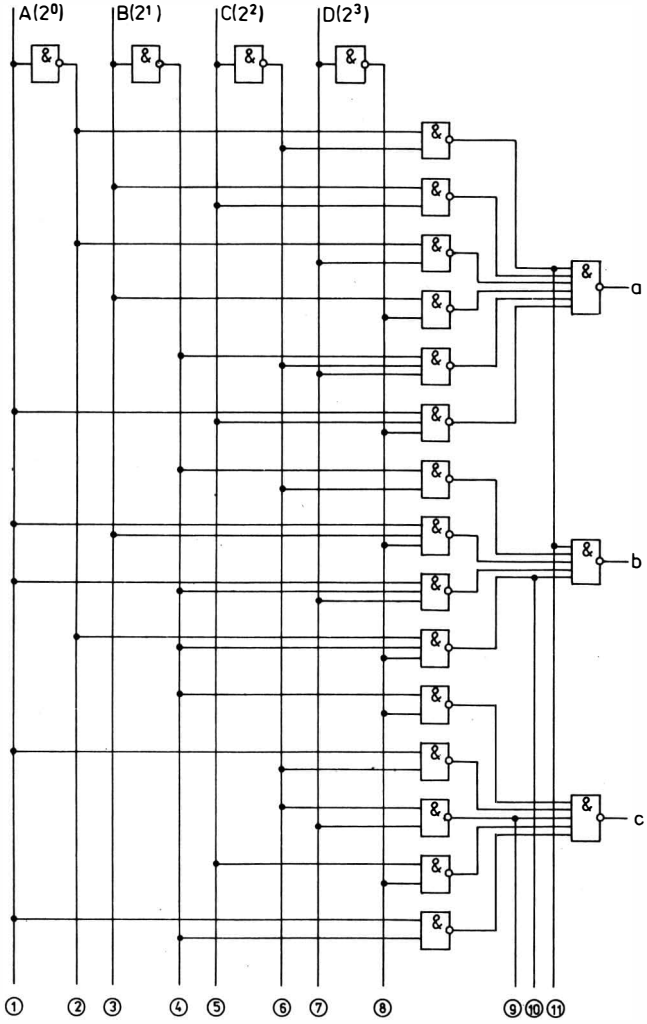


Bild 3.46 Darstellung der Hexadezimalziffern 0 bis F


A(2⁰)
B(2¹)
C(2²)
D(2³)
&
a
b
c
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩
⑪

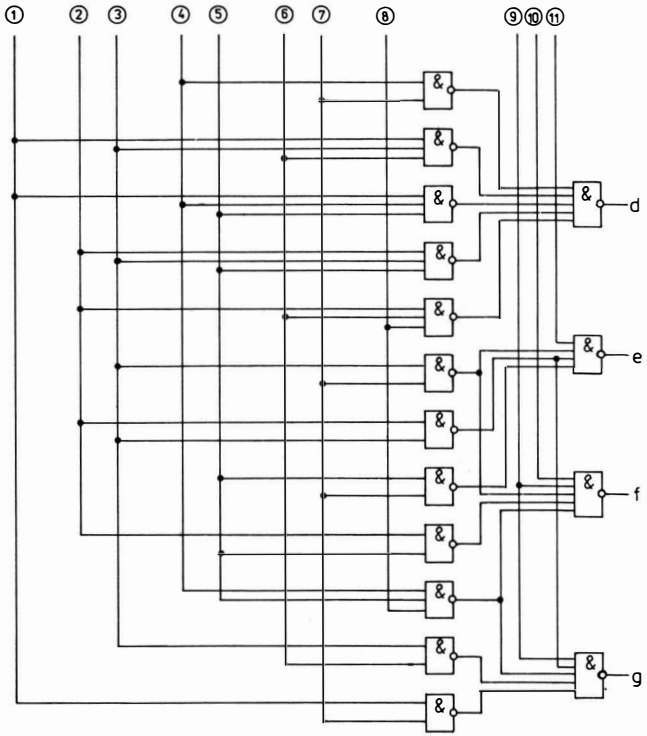


Bild 3.47 7-Segment-Decoder für die Hexadezimalziffern 0 bis F

Für die Ansteuerung der Segmente gilt:

$a = \overline{A}\overline{C} \vee B\overline{C} \vee \overline{A}D \vee B\overline{D} \vee \overline{B}\overline{C}\overline{D} \vee A\overline{C}\overline{D};$

$b = \overline{A}\overline{C} \vee \overline{B}\overline{C} \vee AB\overline{D} \vee A\overline{B}D \vee \overline{A}\overline{B}\overline{D};$

$c = \overline{B}\overline{D} \vee A\overline{C} \vee \overline{C}D \vee C\overline{D} \vee A\overline{B};$

$d = \overline{B}D \vee AB\overline{C} \vee A\overline{B}C \vee \overline{A}BC \vee \overline{A}C\overline{D};$

$e = BD \vee \overline{A}B \vee CD \vee \overline{A}\overline{C};$

$f = \overline{A}C \vee BD \vee \overline{C}D \vee \overline{B}C\overline{D} \vee \overline{A}\overline{B}\overline{D};$

$g = B\overline{C} \vee \overline{C}D \vee AD \vee \overline{A}B \vee \overline{B}C\overline{D}.$

Die Bausteine *7446, 7447, 7446 A* und *7447 A* sind integrierte 7-Segment-Decoder.

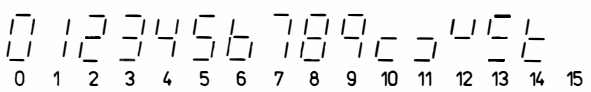


Bild 3.48 Darstellung der Hexadezimalziffern für die Bausteine *7446* und *7447*

Bild 3.48 zeigt die Hexadezimaldarstellung der Bausteine *7446, 7447, 7446 A, 7447 A,* Bild 3.49 den Aufbau der Decoder.
RBI = L ergibt eine automatische Unterdrückung der Nullanzeige. Die Segmentausgänge erhalten H-Signal, wenn die Eingänge A, B, C und D L-Signal haben (Nullbedingung).
Bei LT = L (Lampentest) führen alle Segmentausgänge L-Signal (Helltastung).
Alle 4 Bausteine unterscheiden sich bei gleicher Innenschaltung lediglich in ihren Endstufen.

7446:	offener Kollektorausgang	30 V/20 mA;
7447:	offener Kollektorausgang	15 V/20 mA;
7446 A:	offener Kollektorausgang	30 V/40 mA;
7447 A:	offener Kollektorausgang	15 V/40 mA;

Die Spannung der heute verwendeten GaAs-Leuchtdioden beträgt etwa $U_D = 1{,}6$ V. Da der vorgegebene Strom I nicht überschritten werden darf, sind zwischen Segmentanzeige und Decoder Widerstände zu schalten (Bild 3.50). Die Widerstände R können nach der Gleichung

$$R = \frac{U_b - U_D}{I};$$

$U_b = 5$ V; $U_D = 1{,}6$ V;
I = angegebener höchster Wert des Stromes;

berechnet werden.
Oft liefern bestimmte Meßgeräte Ziffern in einer anderen Darstellung, als sie benötigt werden. Dann sind sogenannte „Umcodierer" notwendig. Es gibt folgende Umcodierer:
BCD-Code → Aiken-Code;
BCD-Code → 3-Exzeß-Code;
Aiken-Code → BCD-Code;
3-Exzeß-Code → BCD-Code.

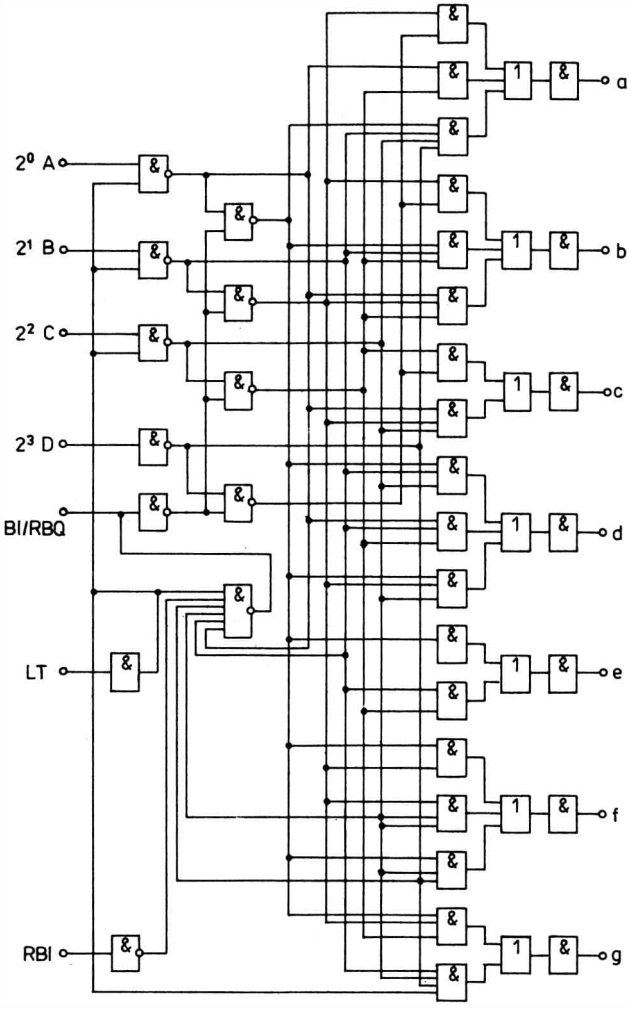


Bild 3.49 Schaltbild der 7-Segment-Decoderbausteine *7446* und *7447*

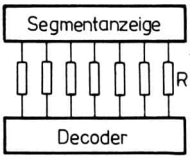


Bild 3.50
Zusammenschaltung von 7-Segment-Decoder und -Anzeige

Prüfbiterzeugung – Paritätsprüfer

Die häufigste Methode zur Feststellung von Fehlern bei einer Datenübertragung ist die Prüfbitkontrolle. Dabei wird die Anzahl der zur Übertragung verwendeten Bit durch ein Zusatzbit, das sogenannte *Prüfbit*, ergänzt. Dieses Prüfbit wird nun so gesetzt, daß die Anzahl der Einsen im Gesamtwort (Information + Prüfbit) gerade oder ungerade ist. Im Sender setzt man das Prüfbit dazu, im Empfänger läßt sich dann die gerade oder ungerade Parität überprüfen.
Bild 3.51 zeigt den Aufbau des Bausteins *74180*, der zur Überprüfung und Erzeugung der Prüfbits eines 8-Bit-Wortes (7bit + Prüfbit) verwendet werden kann. Aus Tabelle 3.13. läßt sich die zugehörige Funktionstabelle ersehen.
Soll z. B. zu einem 7-Bit-Wort ein 8. Bit so gesetzt werden, daß geradzahlige Parität entsteht, so verbindet man die 7 Bit mit E_0 bis E_6, setzt $E_7 = 0$, $S_0 = 1$ und $S_1 = 0$. Damit wird $A_1 = 1$, wenn E_0 bis

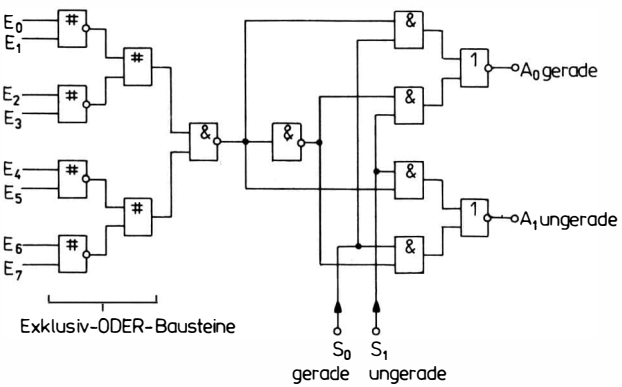
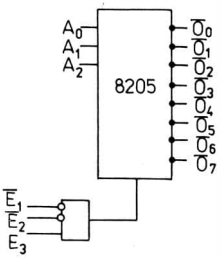


Bild 3.51 Aufbau des Bausteins *74180* zur Prüfbitbildung



$A_2A_1A_0$	$\overline{O}_0\overline{O}_1\overline{O}_2\overline{O}_3\overline{O}_4\overline{O}_5\overline{O}_6\overline{O}_7$
L L L	L H H H H H H H
L L H	H L H H H H H H
L H L	H H L H H H H H
L H H	H H H L H H H H
H L L	H H H H L H H H
H L H	H H H H H L H H
H H L	H H H H H H L H
H H H	H H H H H H H L

Bild 3.52 3-zu-8-Decoder *8205* mit Funktionstabelle

E_6 ungerade sind, und $A_1 = 0$, wenn E_0 bis E_6 gerade sind. Es gilt $A_0 = A_1$; A_1 ist unmittelbar das gesuchte Prüfbit. Zur Prüfbitkontrolle verbindet man die 8 Datenleitungen mit E_0 bis E_7, setzt $S_0 = 1$ und $S_1 = 0$. Damit ist bei geradzahliger Anzahl von Einsen $A_0 = 1$, $A_1 = 0$ und bei ungeradzahliger Anzahl $A_0 = 0$, $A_1 = 1$.
Mit 2 Bausteinen *74180* läßt sich auch eine 16-Bit-Prüfbitlogik aufbauen.
Wichtige Funktionen bei der Adreßentschlüsselung sind die Umcodierungen
Binär → Oktal;
Binär → Hexadezimal.
Im Fall „Binär → Oktal" werden aus einer Oktalziffer 8 Einzelsignale (3 zu 8), im Fall „Binär → Hexadezimal" 16 Einzelsignale (4 zu 16) gebildet. Der Baustein *8205* (Bild 3.52) realisiert die Funktion 3 zu 8.
In Tabelle 3.14. sind einige Codierschaltkreise zusammengestellt.

Tabelle 3.13. Funktionstabelle des Paritätsbausteins 74180

E_0-E_7	S_0	S_1	A_0	A_1
gerade	H	L	H	L
ungerade	H	L	L	H
gerade	L	H	L	H
ungerade	L	H	H	L
beliebig	H	H	L	L
beliebig	L	L	H	H

Tabelle 3.14. Zusammenstellung einiger Codierschaltkreise

Typ		Kurzcharakteristik
MH 74154	ČSSR	4-Bit-Binär-Decoder (4zu 16)
K 155 ID 4	UdSSR	2 × 2-Bit-Binär-Decoder (2 zu 4)
K 155 ID 7	UdSSR	3-Bit-Binär-Decoder (3 zu 8); *Schottky*-TTL
K 500 ID 161	UdSSR	3-Bit-Binär-Decoder (3 zu 8); ECL, invertierender Ausgang
K 500 ID 162	UdSSR	3-Bit-Binär-Decoder (3 zu 8); ECL
K 155 PR 6	UdSSR	BCD/Binär-Konverter; offener Kollektorausgang
K 155 PR 7	UdSSR	Binär/BCD-Konverter; offener Kollektorausgang
7443APC	UVR	3-Exzeß/Dezimal-Decoder
MH 7746	ČSSR	BCD/7-Segment-Decoder; Treiber, offener Kollektorausgang (30 V, 20 mA)
MH 7747	ČSSR	BCD/7-Segment-Decoder; Treiber, offener Kollektorausgang (15 V, 20 mA)
MH 7442	ČSSR	BCD/Dezimal-Decoder
7449 PC	UVR	BCD/7-Segment-Decoder; Treiber, offener Kollektorausgang 5,5 V
K 155 IP 2	UdSSR	9-Bit-Paritätsgenerator; 8-Bit-Paritätsprüfer
MH 74150	ČSSR	16-zu-1-Multiplexer; invertierender Ausgang
MH 74151	ČSSR	8-zu-1-Multiplexer
K 155 KP 5	UdSSR	8-zu-1-Multiplexer; invertierender Ausgang
K 155 KP 2	UdSSR	2 × 4-zu-1-Multiplexer
K 500 ID 164	UdSSR	8-zu-1-Multiplexer; ECL

3.6. Rechenschaltkreise

Der wichtigste Rechenschaltkreis ist der Mikroprozessor. Seine Teilfunktionen werden durch eine Reihe von Rechenschaltungen realisiert, die zusammen das Rechenwerk eines Rechners darstellen. Viele dieser Teilschaltungen sind auch getrennt als Schaltkreise in MSI-Technik ausgeführt. Sie können zum Aufbau kleiner Rechenschaltungen dienen.

Die Grundfunktion des Rechners ist die Addition. Bei der Addition im Dualsystem müssen die Summe und der Übertrag von 2 Summanden und einem eventuellen Übertrag aus dem niedrigeren Stellenwert gebildet werden (Bild 3.53). Tabelle 3.15. zeigt die Funktionstabelle eines solchen Volladdierers.

Über die Kanonische Alternative Normalform ergibt sich aus Tabelle 3.15. für S und $\ddot{U}_N$ die Schaltfunktion

$$S = \ddot{U}_A (\bar{A}\bar{B} \vee AB) \vee \bar{\ddot{U}}_A (\bar{A}B \vee A\bar{B});$$

$$\ddot{U}_N = \ddot{U}_A (\bar{A}B \vee A\bar{B}) \vee AB.$$

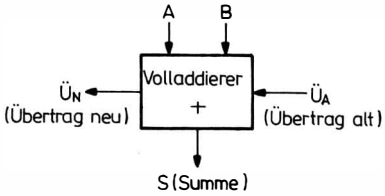


Bild 3.53.
Funktion eines Volladdierers

Tabelle 3.15. Funktionstabelle eines Volladdierers

A	B	$\ddot{U}_A$	S	$\ddot{U}_N$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Aus Bild 3.54 ist der Aufbau des dazugehörigen Volladdierers zu ersehen.
Bild 3.55 zeigt die Innenschaltung des Volladdiererbausteins *7480*. Es ist ein 1-Bit-Volladdierer mit komplementären Ein- und Ausgängen. Für die Eingänge a und b gilt:
$a^* = \overline{a_1 a_2}$ bzw. $b^* = \overline{b_1 b_2}$.
a* und b* können am Baustein abgenommen werden. Wenn a* und b* jedoch als Addiatoreingang benutzt werden, dann müssen

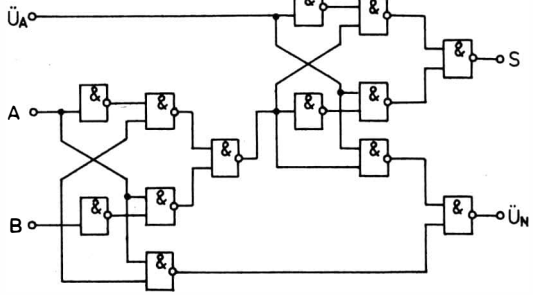


Bild 3.54 Volladdierer nach der Funktionstabelle 3. 15

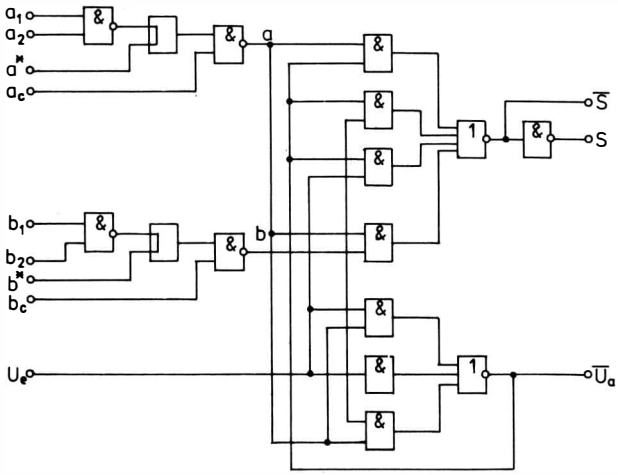


Bild 3.55 Volladdiererbaustein *7480*

a_1, a_2, b_1 und b_2 an Masse liegen. Für die Eingangssignale a und b gilt:
$a = \overline{a^* a_c}$; $b = \overline{b^* b_c}$.
Wenn a_1, a_2, b_1 und b_2 an Masse liegen, so sind $a^* = b^* = 1$ und $a = \overline{a_c}$, $b = \overline{b_c}$.
Addierbausteine gibt es in sehr unterschiedlichen Ausführungen.

Beispiele
7482: 2 Volladdierer (Bild 3.56, Funktionstabelle Tabelle 3.16).
74183: schneller Volladdierer (Bild 3.57, Funktionstabelle Tabelle 3.15).
2 Dualzahlen können mit Hilfe eines Volladdierers parallel oder seriell addiert werden. Aus Bild 3.58 ist der Aufbau des parallelen und aus Bild 3.59 der des seriellen Addierwerks zu ersehen. Die Register A und B müssen für die Paralleleingabe geeignet sein. Das Summenregister S nimmt die Daten seriell auf und gibt sie parallel ab. Mit jedem Takt werden die Register um eine Stelle nach rechts verschoben. Dabei werden die Stellen A_0, B_0 und ÜFF (Übertrags-Flip-Flop) über die Addierschaltung geführt. Die Werte der Ausgänge S und $Ü_{N+1}$ der Addierschaltung werden in S_7 und ÜFF eingetragen.

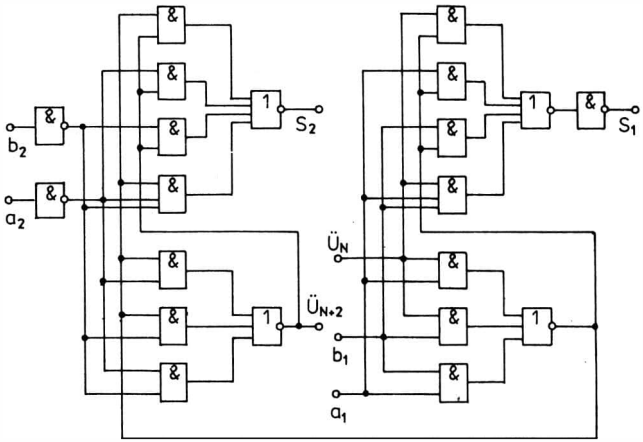


Bild 3.56 Volladdiererbaustein *7482*

Tabelle 3.16 Funktionstabelle des Volladdiererbausteins 7482

				C_n = L			C_n = H		
b_2	a_2	b_1	a_1	S_1	S_2	C_{n+2}	S_1	S_2	C_{n+2}
L	L	L	L	L	L	L	H	L	L
L	L	L	H	H	L	L	L	H	L
L	L	H	L	H	L	L	L	H	L
L	L	H	H	L	H	L	H	H	L
L	H	L	L	L	H	L	H	H	L
L	H	L	H	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
L	H	H	H	L	L	H	H	L	H
H	L	L	L	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
H	L	H	L	H	H	L	L	L	H
H	L	H	H	L	L	H	H	L	H
H	H	L	L	L	L	H	H	L	H
H	H	L	H	H	L	H	L	H	H
H	H	H	L	H	L	L	L	H	H
H	H	H	H	L	H	H	H	H	H

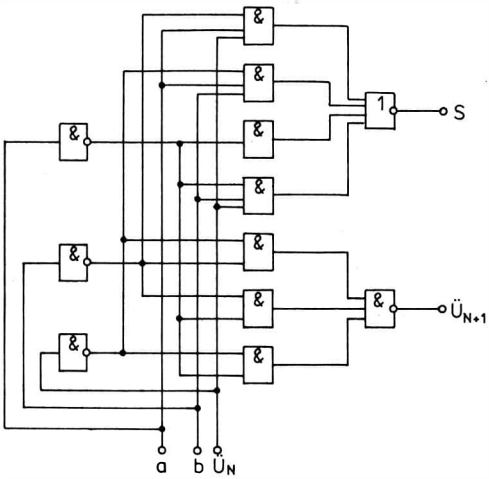


Bild 3.57 Schneller Volladdierer *74183*

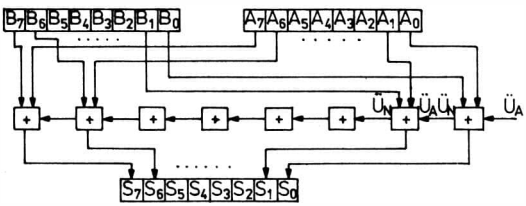


Bild 3.58 Paralleles Addierwerk

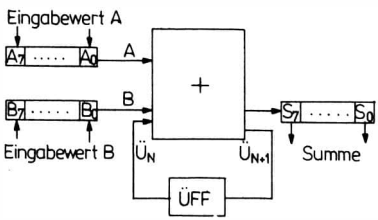


Bild 3.59
Serielles Addierwerk

Eine sehr häufige Funktion bei Mikroprozessoren ist die Erhöhung (*Inkrementierung*) oder Subtraktion (*Dekrementierung*) einer 1 vom Inhalt eines Registers. Tabelle 3.17. enthält die Funktionstabelle einer Dekrementierung. Über die Kanonische Alternative Normalform ergibt sich aus Tabelle 3.17.:

$B_0 = \bar{A}_0$;

$B_1 = A_0A_1 \vee \bar{A}_0\bar{A}_1$;

$B_2 = A_1A_2 \vee A_0A_2 \vee \bar{A}_0\bar{A}_1\bar{A}_2$;

$B_3 = A_1A_3 \vee A_0A_3 \vee A_2A_3 \vee \bar{A}_0\bar{A}_1\bar{A}_2\bar{A}_3$.

Aus Bild 3.60 ist der Aufbau eines Dekrementierers zu ersehen. In Tabelle 3.18. sind einige Rechenschaltkreise zusammengestellt.

Tabelle 3.17. Funktionstabelle eines Dekrementierers

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0

Tabelle 3.18. Zusammenstellung einiger Rechenschaltkreise

Typ		Kurzcharakteristik
K 155 IM 1	UdSSR	1-Bit-Volladdierer
K 155 IM 2	UdSSR	2-Bit-Volladdierer
K 155 IM 3	UdSSR	4-Bit-Volladdierer
K 155 IM 3	UdSSR	4-Bit-Recheneinheit
7485 PC	UVR	4-Bit-Vergleicher
K 531 IP 3	UdSSR	4-Bit-Recheneinheit; *Schottky*-TTL
K 500 IP 181	UdSSR	4-Bit-Recheneinheit; ECL

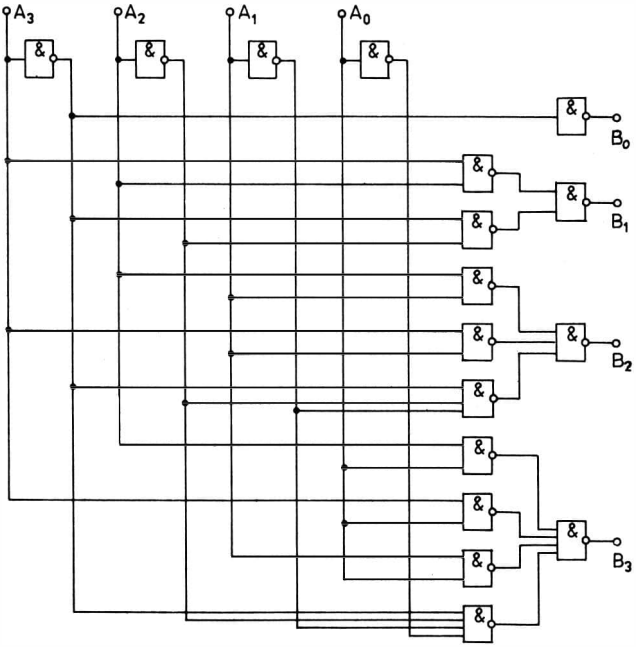


Bild 3.60 Dekrementierschaltung

3.7. Bustreiber

Die aus den Schaltkreisen kommenden Signale haben nicht immer die Leistung, die notwendig ist, um mehrere angeschlossene Bausteine zu steuern. Teilweise sind auch die Mindestpotentiale zur Steuerung eines Bausteins größer als die Potentiale, die der Geberschaltkreis liefert. Zur Anpassung der Potentiale und der Leistungsparameter im Mikrorechnerbus dienen sogenannte Bustreiber. Da der Mikrorechnerbus meistens in beiden Richtungen betrieben wird, arbeiten diese Bustreiber „bidirektional".
Bild 3.61 zeigt den Aufbau des Bustreiberbausteins *8216*. Wenn bei A H-Potential vorliegt, so arbeitet der Baustein in Richtung DI → DB und bei H-Potential an B in Richtung DB → DO. Liegt A

bzw. B auf L-Potential, so sind die Ausgänge der entsprechenden Treiber hochohmig.

Tabelle 3.19. Funktionstabelle des Bustreiberbausteins 8216

$\overline{CS}$	$\overline{DIEN}$	
L	L	DI → DB
L	H	DB → DO
H	L	hochohmig
H	H	hochohmig

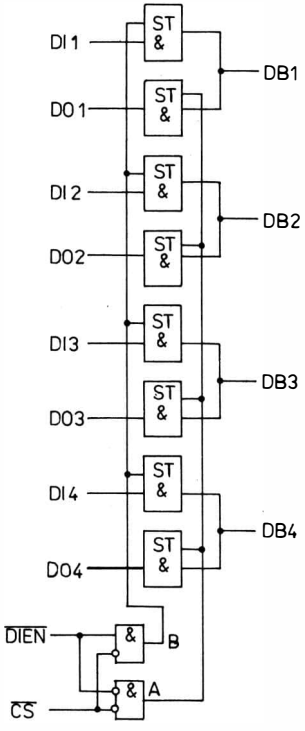


Bild 3.61
Bustreiberbaustein *8216*

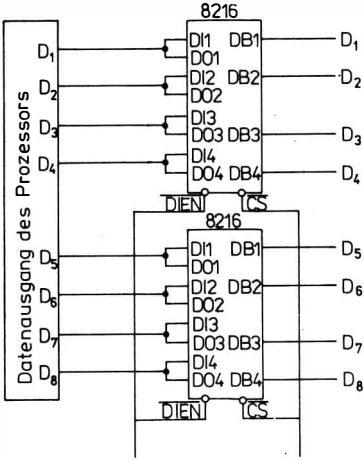


Bild 3.62
Steuerung eines 8-Bit-Datenbusses mit 2 × *8216*

Die Richtung des Informationsflusses wird durch das Signal $\overline{\text{DIEN}}$ bestimmt, während man mit $\overline{\text{CS}}$ (chip-select) den Baustein freigibt. Aus Tabelle 3.19. ist die Funktionstabelle des Bausteins *8216* zu ersehen.
Bild 3.62 zeigt die Verwendung von 2 Bausteinen *8216* zur Steuerung eines 8-Bit-Datenbusses. Während die Eingangsspannungen des Bausteins *8216* für H-Potential nur mindestens 2 V betragen müssen, liefert der Baustein H-Potential von mindestens 3,65 V.

Tabelle 3.20. Zusammenstellung einiger Treiberschaltkreise

Typ		Kurzcharakteristik
K 155 LN 3	UdSSR	6 Treiber, invertiert, offener Kollektorausgang 30 V
K 155 LN 4	UdSSR	6 Treiber, offener Kollektorausgang, 30 V
7416 PC	UVR	6 Treiber, invertiert, offener Kollektorausgang, 15 V
7417 PC	UVR	6 Treiber, offener Kollektorausgang, 15 V
74125 PC	UVR	4 Bustreiber, Tri-state
74126 PC	UVR	4 Bustreiber, Tri-state
SN 74367 AN		6 Bustreiber, 2 Enable-Eingänge, Tri-state

3.8. Zähler

Zähler lassen sich auf verschiedenste Art aufbauen. Verwendet man getriggerte Flip-Flop, die mit der HL-Flanke umkippen, und verbindet den jeweiligen Ausgang der vorherigen Zählstufe mit dem Takteingang des nächsten Flip-Flop (Bild 3.63), so spricht man von *asynchronen Zählern*. Wird der Takt an alle Flip-Flop gleichzeitig angelegt und eine logische Verbindung zwischen den Ausgängen der vorangehenden Flip-Flop und den Eingängen des nächsten Flip-Flop geschaffen, so erhält man *synchrone Zähler*. Ein Beispiel eines synchronen Vorwärtszählers zeigt Bild 3.64.

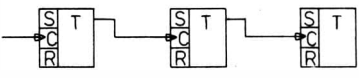


Bild 3.63
Schaltprinzip eines asynchronen Zählers

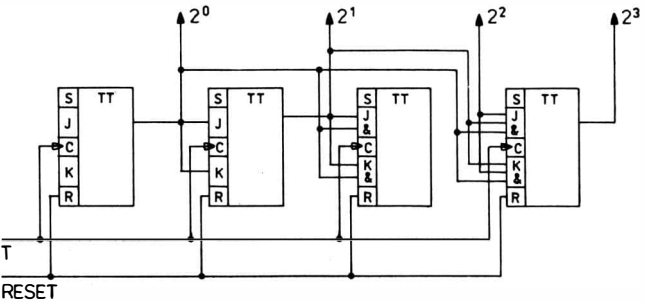


Bild 3.64 Synchroner Zähler

Zähler mit JK-Flip-Flop

Jedes als Trigger geschaltete JK-Flip-Flop teilt die Eingangsfrequenz im Verhältnis 1:2. Bild 3.65 zeigt die Schaltung für einen Vorwärtszähler im Dualcode und das zugehörige Impulsdiagramm. Soll der Zähler bereits nach Stellung 9 zurückschalten, so kann eine Schaltung nach Bild 3.66 verwendet werden. Durch eine entsprechende Zählstruktur, d. h. durch Schalten der einzelnen Setz- und Rücksetzbedingungen, lassen sich beliebige Zählfunktionen realisieren. Zum Beispiel benötigt man für einen Zähler im 3-Exzeß-Code die in Tabelle 3.21. stehende Zählfolge. Bild 3.67 zeigt das dazugehörige Schaltbild mit dem Taktdiagramm.

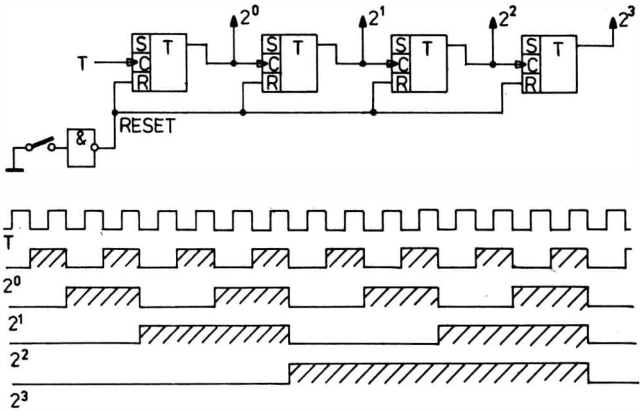


Bild 3.65 Vorwärtszähler im Dualcode mit dem dazugehörigen Impulsdiagramm

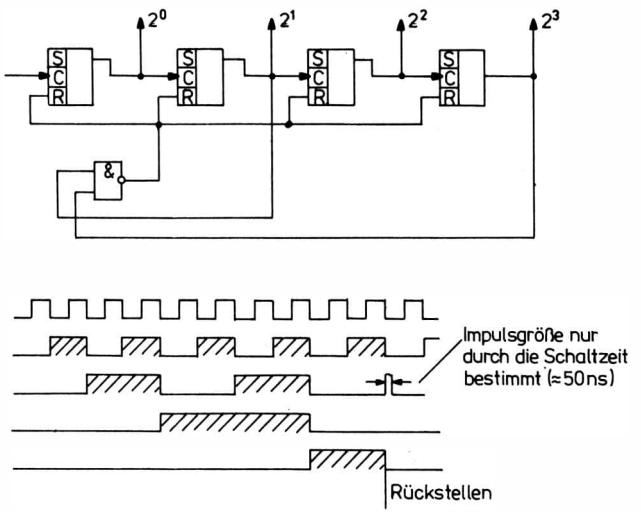


Bild 3.66 Vorwärtszähler im BCD-Code

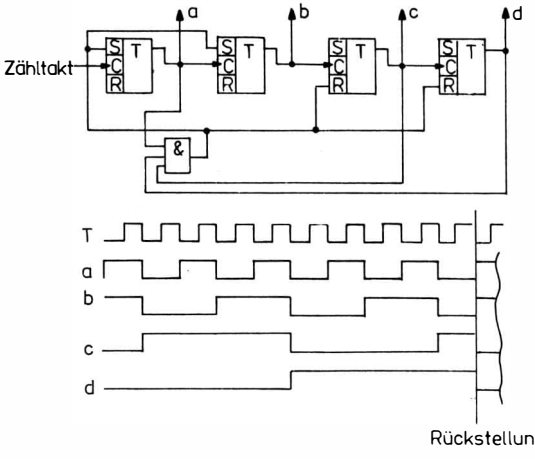


Bild 3.67 Zähler im 3-Exzeß-Code (Das gezeichnete AND-Gatter ist ein NAND!)

Tabelle 3.21 Zählfolge für einen Zähler im 3-Exzeß-Code

	d	c	b	a
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

Zähler mit RS-Flip-FLop

Schaltet man vor die Eingangsstufen eines RS-Flip-FLop Dynamikvorsätze (Kondensatorspeicher), so lassen sich sehr einfach Zählschaltungen aufbauen.

Der in Bild 3.68 dargestellte Speicher mit dynamischer Ansteuerung eignet sich als Baustein zum Aufbau von Zählstufen. Bild 3.69 zeigt einen vierstelligen Dualzähler. Die einzelnen Stufen sind als Teiler geschaltet. Jede Stufe wird von der Vorstufe angesteuert, wenn deren Ausgangssignal von H nach L springt. Mit TL werden die Zählstufen in die Ausgangsstellung 0000 gebracht.

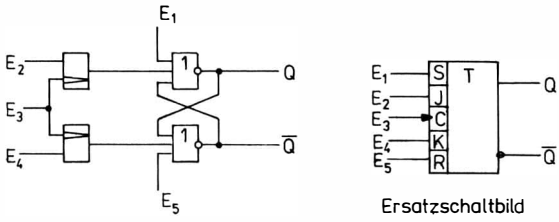


Bild 3.68 RS-Flip-Flop mit dynamischer Ansteuerung

Tabelle 3.22. Zusammenstellung einiger Zählerschaltkreise

Typ		Kurzcharakteristik
MH 7490	ČSSR	Dezimalzähler
K 155 IE 9	UdSSR	programmierbarer Dezimalzähler
74176 PC	UVR	programmierbarer Dezimalzähler
74190 PC	UVR	Synchroner programmierbarer Vorwärts-/Rückwärts-Zähler (Dezimalzähler)
D 192 C/D	DDR	Synchroner programmierbarer Vorwärts-/Rückwärts-Zähler (Dezimalzähler)
74290 PC	UVR	Dezimalzähler
MH 7493	ČSSR	4-Bit-Binärzähler
74161 PC	UVR	programmierbarer 4-Bit-Binärzähler
74177 PC	UVR	programmierbarer 4-Bit-Binärzähler
74191 PC	UVR	Synchroner programmierbarer Vorwärts-/Rückwärts-Zähler (4-Bit-Binärzähler)
D 193 C/D	DDR	Synchroner programmierbarer Vorwärts-/Rückwärts-Zähler (4-Bit-Binärzähler)
74293 PC	UVR	4-Bit-Dualzähler

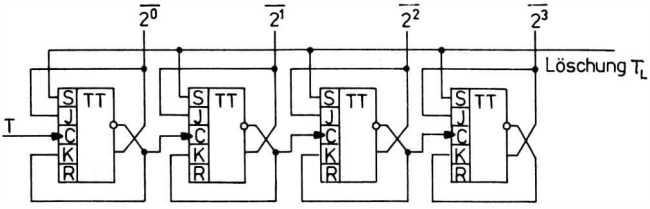


Bild 3.69 Vierstelliger Dualzähler mit Schaltelementen nach Bild 3.68

3.9. Taktgeneratoren

Die meisten Rechnerschaltkreise arbeiten taktgesteuert. Bild 3.70 zeigt eine einfache Variante zur Erzeugung einer unstabilisierten Taktserie. Liegt am Punkt A H-Pegel, so hat Punkt B L-Pegel. Der Kondensator C entlädt sich über R, bis Punkt A L-Pegel erhält. Jetzt hat B H-Pegel, und der Kondensator C lädt sich über R und den Negator 1 wieder auf. Die Umladung von C wird durch Negator 2 geringfügig unterstützt.

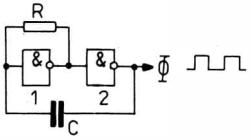


Bild 3.70
Erzeugung einer unstabilisierten Taktserie

Bild 3.71 zeigt eine Schaltung, in der die Auf- und Entladung von C durch R_1 und R_2 geschieht. Außerdem wird durch Hinzuschalten eines Quarzes die erzeugte Taktserie frequenzstabilisiert. Liegt die Quarzfrequenz zu hoch, so kann man die Taktfrequenz mit D-Flip-Flop untersetzen. Bild 3.72 zeigt eine solche Untersetzung mit 2 D-Flip-Flop.

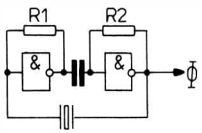


Bild 3.71
Einfacher Taktgenerator

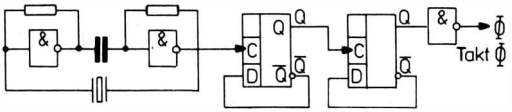


Bild 3.72 Taktgenerator mit zweifacher Untersetzung

Sollen aus dem Urtakt 2 Taktserien hergestellt werden, so läßt sich das mit einem D-Flip-Flop nach Bild 3.73 erreichen.
Der Baustein *8224* ist ein integrierter Taktgenerator für den Mikroprozessor *8080*. Er erzeugt die notwendigen Taktimpulse Φ_1 und Φ_2 und dient gleichzeitig zur Verarbeitung des Zeitsignals SYNC und zur Erzeugung der Signale RESET, READY und $\overline{\text{STSTB}}$ (Bild 3.74).

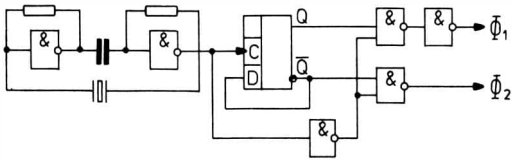


Bild 3.73 Taktgenerator zur Erzeugung von 2 Taktserien Φ_1 und Φ_2

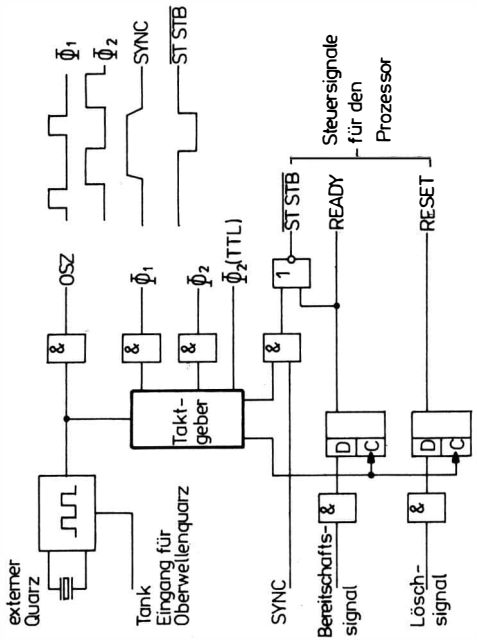


Bild 3.74 Integrierter Taktgenerator *8224* für den Prozessor *8080*

4. Mikroprozessoren

Der Mikroprozessor ist der Schaltkreis innerhalb eines Mikrorechners, der die Steuerung des Programmablaufs übernimmt. Er beinhaltet das Rechen- und Steuerwerk eines Rechners. Zum Einbau in Rechnersysteme verfügt er über Ein- und Ausgabesignale, mit deren Hilfe weitere Bausteine eines Rechners angeschlossen werden können. Die Ein- und Ausgangssignale des Mikroprozessors kann man unterteilen in

- Adreßsignale (Adreßbus)
- Datensignale (Datenbus)
- Steuer- und Meldesignale (Controlbus)
- Versorgungsspannungen (Taktsignale, Betriebsspannungen und Masseleitung).

Die Arbeitsweise des Mikroprozessors wird durch seinen Befehlsvorrat bestimmt. Den Befehlen entsprechen Signale, durch die die Inhalte der einzelnen internen Register untereinander transportiert werden. Während des Transports führen logische Schaltungen die einzelnen Operationen aus. Voraussetzung zum Verständnis des Befehlsschlüssels ist deshalb die Registerstruktur des Prozessors.

Der Befehlsschlüssel bildet die Basis für die Entwicklung der Software eines Rechners. Um bereits entwickelte Software auch auf Nachfolgerechnern zu nutzen, versucht man sie so aufzubauen, daß im allgemeinen die Befehle des Vorgängers als Teilmenge enthalten sind. Damit wird eine sogenannte Aufwärtskompatibilität erreicht.

Die Prozessoren *U 808 D, 8080, U 880 D* und *U 881* bilden eine solche Generationsreihe.

Mit dem Prozessor *U 808 D* begann die Entwicklung der Mikrorechentechnik in der DDR. Der Baustein benötigt eine umfangreiche Busanpassung und benutzt nicht den Arbeitsspeicher zur Stackorganisation. Bei der Interruptorganisation wurde nicht der Begriff ‚Interruptvektor' verwendet. Bild 4.1 zeigt die logische Struktur dieses Bausteins. Er wird für Neuentwicklungen nicht eingesetzt.

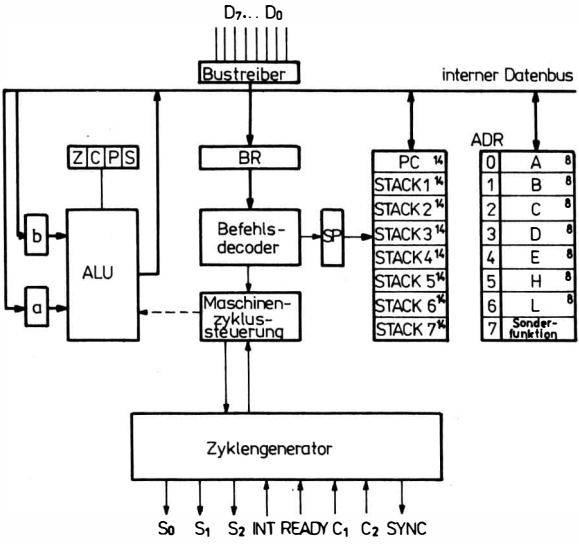


Bild 4.1 Registerstruktur des Bausteins *U 808 D* (statt C_1, C_2 lies Φ_1, Φ_2)

4.1. Der Mikroprozessorbaustein *U 880*

Der Baustein *U 880* ist gegenüber dem Baustein *U 808 D* weiter vervollkommnet. Die wesentlichen Verbesserungen sind folgende:

- Es ist nur eine Betriebsspannung $V_{cc} = 5$ V notwendig und ebenfalls nur ein Steuertakt Φ erforderlich.
- Die Steuersignale zur Auswahl und Ansteuerung der externen Bausteine werden im Prozessor schon so weit aufbereitet, daß sie direkt mit den Eingängen und Ausgängen der externen Bausteine verbunden werden können.
- Die Befehlsliste ist wesentlich erweitert; während der Baustein *U 808 D* 48 Basisbefehle verarbeitet, sind es beim *U 880* 158 Befehle. Neu sind dabei Befehle für Doppelwortoperationen, für BCD-Arithmetik, für einen zweiten Registersatz, Blocktransferbefehle und Blocksuchbefehle in Verbindung mit dem Speicher und den Ein- und Ausgabebausteinen, Bitoperationen, Indexoperationen sowie wesentlich erweiterte Verschiebebefehle.

– Die Behandlung von externen INTERRUPT ist durch einen maskierten INTERRUPT sowie durch die Möglichkeit des Aufbaus einer Adreßliste für unterschiedliche INTERRUPT-Routinen erweitert worden.

4.1.1. Registerstruktur des Mikroprozessorbausteins *U 880*

Aus Bild 4.2 ist die Registerstruktur des Bausteins *U 880* zu ersehen. Der Mikroprozessor enthält einen internen 8-Bit-Bus, von dem aus alle Register zu erreichen sind. Von diesem Bus aus werden Daten über den externen Datenbus D_0 bis D_7 ein- und ausgegeben. Er läßt sich in beiden Richtungen betreiben. Der externe bidirektionale Datenbus ist über Bustreiber mit dem internen 8-Bit-Bus verbunden. An den internen 8-Bit-Bus sind angeschlossen:

– 2 Registersätze zur Zwischenspeicherung der Zahlen im Prozessor, die aus je 8-Bit-Registern bestehen. Der 1. Registersatz beinhaltet die Register A, F[2)], B, C, D, E, H, L, der 2. Registersatz die Register A′, F′[2)], B′, C′, E′, H′, L′. Durch einen einfachen Austauschbefehl können die Inhalte der Registersätze komplett vertauscht werden. Dadurch ist es möglich, einen bestimmten Programmabschnitt einem der Registersätze zuzuordnen. Wechselt das Programm, dann können die dazugehörigen Registersätze umgetauscht werden.

– Zweckregister I, R, IX, IY, SP, PC.

Das Register I (8-Bit-INTERRUPT-Adreßregister) enthält den höherwertigen Teil einer Adresse, deren niederwertiger Teil bei einem INTERRUPT von dem entsprechenden Gerät gebildet wird. Die Adresse weist auf eine Speicherzelle, in der die Startadresse des INTERRUPT-Bedienprogramms steht.

Das Register R (Speicherauffrischregister) enthält eine 7-Bit-Adresse, die in Verbindung mit dem Auffrischsignal RFSH auf den niederwertigen 7 Bit des Adreßbus ausgesendet wird. Diese Adresse wird während der Operationscodeentschlüsselung ausgesendet. Sie dient zum Auffrischen von dynamischen Speichern. Während jedes Operationscodeholzyklus erhöht sich der Inhalt des Registers R um 1.

Die beiden Register IX und IY (Indexregister) können eine 16-

[2)] F, F′ sind Flagregister

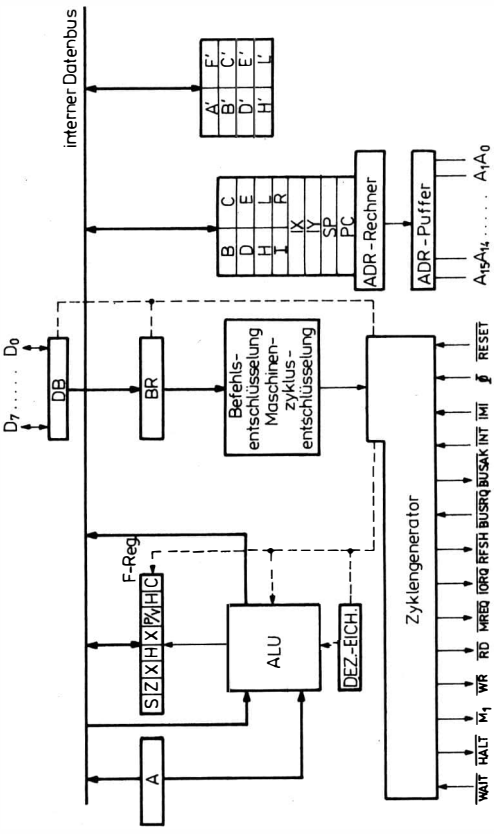


Bild 4.2 Registerstruktur des Bausteins *U 880*

Bit-Basisadresse enthalten. Während der Adressenrechnung wird aus dieser Basisadresse durch Addition einer Adreßzahl die eigentliche Operandenadresse ermittelt.

Das Register SP (Stackpointer) enthält eine 16-Bit-Adresse, die die Speicherzelle an der Spitze eines Kellerspeichers adressiert. Der Kellerspeicher ist als „last in – first out-Speicher" organisiert (das zuletzt eingeschriebene Wort wird zuerst gelesen).

Das Register PC (Program-Counter oder Befehlszähler) enthält eine 16-Bit-Adresse, die angibt, aus welcher Speicherzelle der laufende Befehl geholt wird.

- In dem Befehlsregister BR wird der Operationscode des laufenden Befehls gespeichert. Hier kommt es zur Decodierung des Befehls und zur Bildung der Steuersignale für dessen Abarbeitung. Die Steuersignale bestehen aus den Befehls- und den Zeitsignalen. Die Befehlssignale werden durch die Befehlsentschlüsselung und die dazugehörigen Zeitsignale durch die Zeitsteuerung gebildet. Die Zeitsteuerung besteht aus dem Zyklengenerator, der durch den externen Takt Φ und die Befehlssignale gesteuert wird. Im Zyklengenerator werden auch die Signale zur Steuerung der externen Bausteine gebildet sowie die von den externen Bausteinen kommenden Signale abgetastet.
- Das Flagregister enthält 6 Flip-Flop, die in Abhängigkeit von den einzelnen Befehlen und vom Ergebnis der Befehle gesetzt oder rückgesetzt werden. Die einzelnen Flags haben folgende Bedeutung:

  C: Carry-Flag

  C ist gleich 1, wenn bei der Addition ein Übertrag in die 8. Stelle auftritt, oder wenn bei der Subtraktion ein Borgen von der 8. Stelle notwendig wird.

  N: Subtraktions-Flag

  N ist gleich 1, wenn die ausgeführte Operation eine Subtraktion war.

  P/V: Parity-Überlauf-Flag (Überlauf = Overlow)

  P/V ist gleich 1, bei logischen Operationen, wenn die Anzahl der Einsen im Ergebnis geradzahlig ist, bei Rechenoperationen, wenn ein Überlauf auftritt (Ergebnis größer als die größte darstellbare Zahl).[3)]

  H: Half-Carry-Flag

  H ist gleich 1, wenn es bei der Addition zu einem Übertrag in die 4. Stelle kommt oder wenn bei der Subtraktion ein Borgen von der 4. Stelle notwendig wird.

  Z: Zero-Flag

  Z ist gleich 1, wenn das Ergebnis 0 ist.

  S: Sign-Flag

  S ist gleich 1, wenn im Ergebnis das Vorzeichen 1 (negativ) ist.

[3)] Der Prozessor *U 880* arbeitet mit einem 8-Bit-Zahlwort im Zweierkomplement. Der Stellenwert 2^7 entspricht dem Vorzeichen. Die größte positive Zahl ist $2^7 - 1$, die negative Zahl mit dem größten Betrag -2^7. Der vom Prozessor erfaßte Zahlenbereich umfaßt $-2^7 \leqq Z \leqq 2^7 - 1$.

Beispiele

```
                    1  ↖          Übertrag in 4. Stelle
 120 =    0 1 1 1      1 0 0 0    ergibt 1 → H
+105 =    0 1 1 1      1 0 0 1
-------------------------------
 225 = 0] 1 1 1 1      0 0 0 1
       ↙   ↘
0 → C      1 → P/V        wegen Überlauf
                                  kein Übertrag in 4. Stelle
                                  ergibt 0 → H
 - 5 =    1 1 1 1      1 0 1 1
-16 =    1 1 1 1      0 0 0 0
-------------------------------
-21 = 1] 1 1 1 0      1 0 1 1
       ↙
1 → C        aber 0 → P/V
```

4.1.2. Befehlsaufbau des Bausteins *U 880*

4.1.2.1. Befehlsstruktur

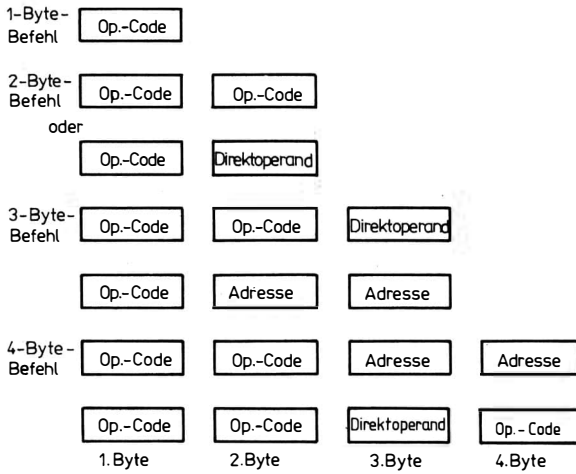


Bild 4.3. Befehlsstrukturen im Prozessor *U 880*

Ein Befehl besteht aus Operationsteil und Adreßteil. Zur Darstellung eines Befehls werden 1 bis 4 Byte benötigt. Davon kann der Operationscode 1 bis 3 Byte [4] und der Adreßteil ebenfalls 1 bis 2 Byte lang sein. Bei den meisten Befehlen ist der Operationscode 1 Byte lang. Bei 2 Byte langen Operationscodes gibt das 1. Byte die Befehlsgruppe an. Durch das 2. Byte und 3. Byte werden spezielle Befehle innerhalb der Gruppe gekennzeichnet. Bild 4.3 zeigt die im Prozessor *U 880* möglichen Befehlsstrukturen.

4.1.2.2. Adreßbildung

Direktoperand
Der zum Befehl gehörende Operand steht im Anschluß an den Operationscode:
Zelle 1 Operationscode,
Zelle 2 NWT-Operand (niederwertiger Teil des Operanden),
Zelle 3 HWT-Operand (höherwertiger Teil des Operanden).

Adressierter Operand
Im Befehl steht die Adresse der Speicherzelle, in der der Operand steht:
Zelle 1 Operationscode,
Zelle 2 NWT-ADR (niederwertiger Teil der Adresse),
Zelle 3 HWT-ADR (höherwertiger Teil der Adresse).

Relative Adressierung
Im Befehl steht eine positive oder negative Zahl N.
Der Operand steht um N Zellen nach oder vor dem Befehl.
Zelle 1 Operationscode
Zelle 2 N (positive oder negative Zahl im Zweierkomplement),
Zelle 3 nächster Befehl;
ADR als Operanden = ADR Zelle 3 + N.

Indirekte Adressierung
Die Adresse ADR des Operanden steht in einem speziellen Register:

[4] Bei einigen Befehlen mit Indexrechnung ist der Operationscode 3 Byte und der Adreßteil 1 Byte lang.

ADR = ⟨Register⟩
Als Register treten die Registerpaare BC, DE, HL sowie die Register SP, IX und IY auf.

Indexierung
Die Adresse ADR des Operanden ergibt sich aus der im Befehl angegebenen Zahl N plus dem Inhalt eines Indexregisters.
Zelle 1 Operationscode
Zelle 2 N (positive oder negative Zahl im Zweierkomplement)
ADR = N + ⟨Indexregister⟩

Registeroperand
Der Operand steht in einem im Befehl angegebenen Register.

4.1.3. Zeitverhalten

Ein Befehl wird in mehreren Maschinenzyklen abgearbeitet. Es gibt Maschinenzyklen für folgende Funktionen:
- Operationscode holen,
- Speicher lesen oder schreiben.
- Ein- und Ausgabe,
- INTERRUPT,
- DMA-Funktion,
- Ausführung einer HALT-Operation.

Ein Maschinenzyklus unterteilt sich in 3 bis 6 Zustände (T-Zyklen). Ein T-Zyklus entspricht einer Periode des Grundtaktes Φ.
Bild 4.4 zeigt ein Beispiel für den Aufbau eines Befehlszyklus (Gesamtzeitraum zur Abarbeitung eines Befehls).

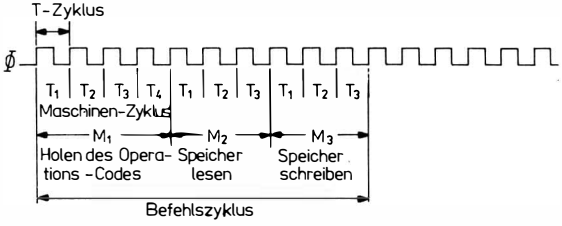


Bild 4.4 Aufbau eines Befehlszyklus im *U 880*

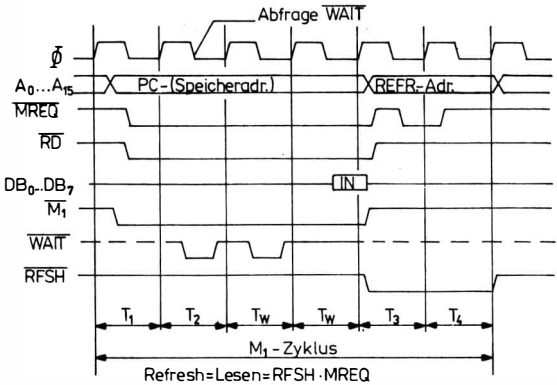


Bild 4.5 Operationscodeholzyklus im *U 880*

Operationscode-Holzyklus (Bild 4.5)
Am Anfang des Zyklus enthält der Befehlszähler die Adresse des Operationscodes. Mit der Rückflanke von T_1 wird das Signal $\overline{MREQ}$ aktiv, gleichzeitig das Signal $\overline{RD}$. $\overline{MREQ}$ bedeutet eine Anforderung zum Speicher; $\overline{RD}$ sagt aus, daß eine Lese-Operation ablaufen soll. Das Einlesen der Daten geschieht mit der Vorderflanke von Φ während T_3. In den Zyklen T_3 und T_4 wird eine Auffrischadresse für dynamische Speicher an den Adreßbus gelegt. Die Auffrischadresse liegt an den Bitstellen A_0 bis A_6, während die übrigen Bit 0 sind. Zu dem Zeitpunkt, in dem Auffrischadresse am Adreßbus liegt, ist das Signal $\overline{RFSH}$ aktiv. Ist zum Zeitpunkt der Rückflanke von Φ im Zustand von T_2 das $\overline{WAIT}$-Signal aktiv, so wird nach T_2 ein Wartezustand eingeschoben. Dieses Einschieben von Wartezuständen wiederholt sich so lange, bis das $\overline{WAIT}$-Signal inaktiv wird.

Speicher-Lese- oder -Schreib-Zyklus (Bild 4.6)
Mit Beginn des Speicher-Lesezyklus (angesteuert durch die Vorderflanke von Φ im Zustand von T_1) wird die Speicheradresse auf den Adreßbus gelegt. Mit der Rückflanke von Φ (Zustand T_1) aktivieren sich die Signale $\overline{MREQ}$ und $\overline{RD}$. Das Signal $\overline{MREQ}$ kann zur Ansteuerung des betreffenden Speichers genommen werden, während $\overline{RD}$ den Speicher auf Lesen umschaltet. Zum Zeitpunkt der Rückflanke von Φ in T_2 tastet der Prozessor das

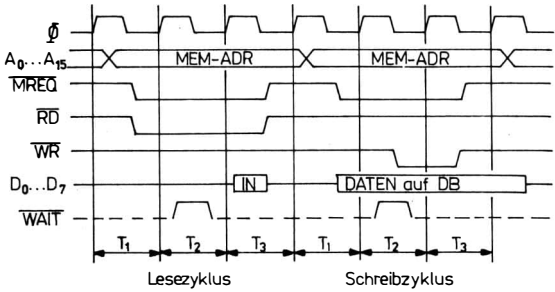


Bild 4.6 Speicher-, Lese- oder Schreibzyklus im *U 880*

$\overline{\text{WAIT}}$-Signal ab und fügt nach T_2 bei aktivem $\overline{\text{WAIT}}$-Signal einen Wartezyklus T_{wait} ein. Während T_{wait} bleiben die Adresse am Adreßbus und die Daten am Datenbus erhalten. Bei der nächsten Rückflanke von Φ wird die Abfrage von $\overline{\text{WAIT}}$ wiederholt und eventuell ein weiterer Wartezustand eingeschoben. Ist das $\overline{\text{WAIT}}$-Signal nicht mehr aktiv, dann folgt der Zustand T_3. Während des Taktes Φ im Zustand T_3 werden die Daten vom Datenbus in den Prozessor übernommen, mit der Rückflanke von Φ in T_3 werden die Signale $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ wieder abgeschaltet.
Beim Speicher-Schreib-Zyklus wird die Adresse genau wie zum Speicher-Lese-Zyklus mit der Vorderflanke von Φ in T_1 auf den Adreßbus gelegt. Mit der Rückflanke von Φ in T_1 werden die Daten an den Datenbus gelegt. Mit der Rückflanke von Φ in T_2 wird das Signal $\overline{\text{WR}}$ aktiv und gleichzeitig das $\overline{\text{WAIT}}$-Signal abgefragt. $\overline{\text{WR}}$ kann zum Umschalten des Speichers auf Schreiben benutzt werden. Während das Übernehmen der Daten in den Speicher mit Φ in T_3 erfolgen kann, wird mit der Rückflanke von Φ in T_3 $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ wieder abgeschaltet.

Ein- und Ausgabe-Zyklus (Bild 4.7)
Beim Ein- und Ausgabe-Zyklus wird nach T_2 automatisch ein Wartezyklus eingefügt, um dem Ein- und Ausgabebaustein zu ermöglichen, eine Adreßentschlüsselung durchzuführen und im Notfall das $\overline{\text{WAIT}}$-Signal zu setzen. Der Ablauf des Zyklus ähnelt dem des Speicher-Lese- oder -Schreib-Zyklus. Zum Zeitpunkt der Vorderflanke von Φ in T_2 wird das Signal $\overline{\text{IORQ}}$ aktiv. Gleichzeitig aktiviert sich entweder $\overline{\text{RD}}$ oder $\overline{\text{WR}}$, je nachdem, ob es sich um

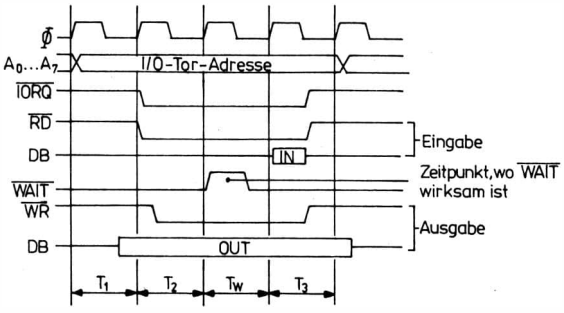


Bild 4.7 Ein- und Ausgabezyklus im *U 880*

eine Eingabe oder um eine Ausgabe handelt. Bei der Ausgabe erscheinen die Daten auf dem Datenbus bereits während T_1, so daß zum Zeitpunkt Φ in T_3 die Daten abgenommen werden können.

INTERRUPT-Zyklus

Bild 4.8 zeigt das Zeitdiagramm für den maskierten INTERRUPT-Zyklus. Das Signal INT wird im letzten Zustand eines Befehls abgetastet. Ist es aktiv, dann beginnt mit T_1 ein INTERRUPT-Zyklus. Mit der Vorderflanke von Φ in T_1 gelangt die Adresse aus dem Befehlszähler an den Adreßbus. Gleichzeitig wird M_1 eingeschaltet. In jedem INTERRUPT-Zyklus werden automatisch 2

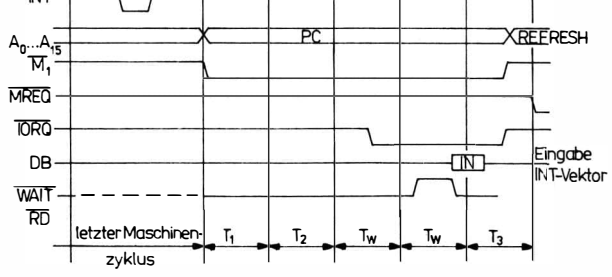


Bild 4.8 Zyklus für den maskierten INTERRUPT

WAIT-Zustände eingeschoben, damit die INTERRUPT-Logik genügend Zeit zur Entschlüsselung der Adresse und zur Bereitstellung des INTERRUPT-Vektors hat. Mit der Rückflanke von Φ im ersten Wartezustand wird zusätzlich das Signal $\overline{IORQ}$ aktiv. Das gleichzeitige Vorhandensein von $\overline{IORQ}$ und $\overline{M_1}$ besagt, daß der INTERRUPT angenommen worden ist. Nach der Rückflanke von Φ des letzten Wartezustands wird vom Prozessor der Datenbus abgetastet und als INTERRUPT-Vektor übernommen.
Während T_3 und T_4 kommt es wie beim Zyklus M_1 zur Ausgabe einer Auffrischadresse mit den dazugehörigen Signalen $\overline{MREQ}$ und $\overline{RFSH}$.
In Abhängigkeit vom INTERRUPT-MODE (0, 1, 2) wird der INTERRUPT-Vektor unterschiedlich interpretiert.

Maskierter INTERRUPT

MODE 0 Der INTERRUPT-Vektor wird als Befehlscode interpretiert.

MODE 1 Der INTERRUPT-Vektor bleibt unberücksichtigt. Es wird der Befehl CALL 38H gebildet und ausgeführt.

MODE 2 Der INTERRUPT-Vektor wird in Verbindung mit dem I-Register als Adresse interpretiert, die angibt, in welcher Zelle sich die Ansprungadresse des Bedienungsprogramms befindet. Es wird der Befehl CALL ⟨I-Register, INTERRUPT-Vektor⟩ ausgeführt.

Nichtmaskierter INTERRUPT

Es wird der Befehl CALL 66H gebildet und ausgeführt (Taktdiagramm Bild 4.9)

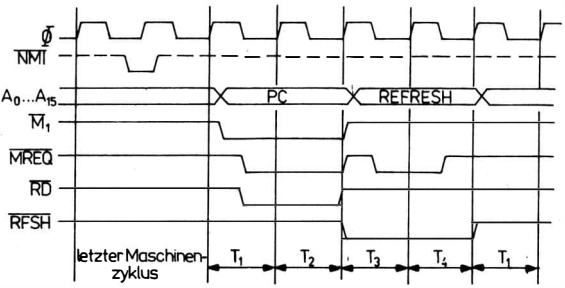


Bild 4.9 Taktdiagramm für den nichtmaskierten INTERRUPT beim *U 880*

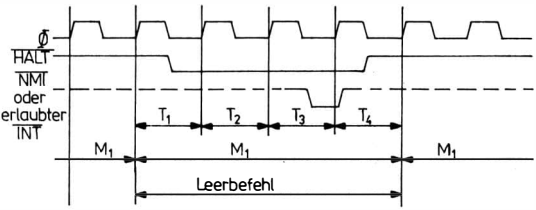


Bild 4.10 Haltezyklus im *U 880*

Haltezyklus (Bild 4.10)
Nach der Entschlüsselung eines HALT-Befehls führt der Prozessor Leerbefehle (NOP) aus, und zwar so lange, bis ein INTERRUPT erscheint (entweder ein nichtmaskierter oder ein maskierter INTERRUPT, wenn dieser erlaubt ist). Die INTERRUPT-Eingänge werden mit der Vorderflanke von Φ in T_4 abgetastet. Ist zu diesem Zeitpunkt ein INTERRUPT-Eingang aktiv, dann setzt sich mit dem nächsten Takt die Befehlsabarbeitung fort. Es wird ein Sprung an die Stelle ausgeführt, die der entsprechenden INTERRUPT-Behandlung entspricht.

DMA-Zyklus (Bild 4.11)
Mit der Vorderflanke von Φ jedes letzten Taktes eines Maschinenzyklus wird das Signal $\overline{\text{BUSRQ}}$ abgetastet. Ist es zu diesem Zeit-

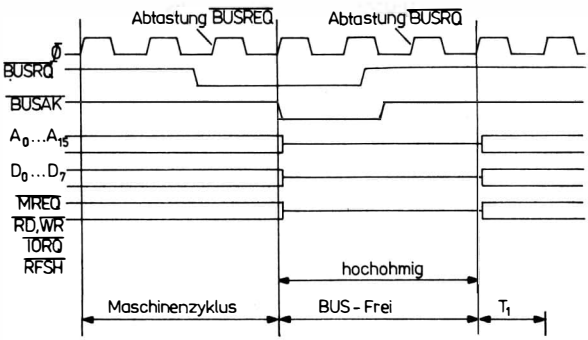


Bild 4.11 DMA-Zyklus im *U 880*

punkt aktiv, so werden mit Beginn das nächsten T_1 der Adreßbus, der Datenbus und die Steuersignale $\overline{MREQ}$, $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ und $\overline{RFSH}$ in den hochohmigen Zustand gesetzt. Gleichzeitig aktiviert sich das Signal $\overline{BUSAK}$, als Zeichen dafür, daß der hochohmige Zustand erreicht ist. Nun wird in jedem Zustand mit der Vorderflanke von Φdas Signal $\overline{BUSRQ}$ abgetastet. Ist es nicht mehr aktiv, so wird im nächsten Takt der hochohmige Zustand beendet, und es beginnt ein neuer Maschinenzyklus. Während $\overline{BUSAK}$ aktiv ist, kann kein INTERRUPT auftreten. Der REFRESH ist unterbrochen.

4.1.4. Befehlsabarbeitung

Während der Abarbeitung eines Befehls werden folgende Arbeitsgänge durchlaufen:
- Befehl holen
- Befehl entschlüsseln
- Operand holen
- Befehl ausführen.

Die einzelnen Arbeitsgänge werden in Maschinenzyklen ausgeführt. Die Art des Maschinenzyklus wird durch die Befehlssignale, die aus der Befehlsentschlüsselung hervorgehen oder von außen als Signale des Steuerbus an den Prozessor gelangen, gebildet. In jedem Maschinenzyklus entstehen durch Hinzufügen von Zeitsignalen zu den Befehlssignalen interne Steuersignale, die die Abarbeitung in Form von Registertransporten steuern.

STS	=	BSI	·	ZSI
Steuersignal		Befehlssignal		Zeitsignal

Gleichzeitig werden zur Steuerung des Datentransfers mit den angeschlossenen Bausteinen äußere Steuersignale gebildet ($\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$, $\overline{MREQ}$, $\overline{M_1}$, $\overline{HALT}$, $\overline{BUSAK}$). Die Abarbeitung eines Befehls setzt sich aus mehreren Maschinenzyklen zusammen. Die Folge dieser Maschinenzyklen ist eine Kombination der in Abschnitt 3.3 genannten Arten der Maschinenzyklen.

4.1.5. Befehlsliste des Prozessors *U 880*

4.1.5.1. Verwendete Abkürzungen bei der Befehlsbeschreibung

r – 8-Bit-Register des Registersatzes, A, B, S, D, E, H, L;
s – 8-Bit-Quellregister oder ein Speicherplatz oder eine 8-Bit-Zahl n
d – 8-Bit-Bestimmungsregister oder Speicherplatz
n – 8-Bit-Zahl
nn – 16-Bit-zahl
dd – 16-Bit-Bestimmungsregister
ss – 16-Bit-Quellregister
s_b – Bit in einem speziellen 8-Bit-Register, b ist die Bit-Nr. (Bild 4.12)

Index L – der niederwertige Teil eines 16-Bit-Registers;
Index H – der höherwertige Teil eines 16-Bit-Registers.
– Steht ein Registername allein, z. B. A, so heißt das:
Inhalt von Register A.
– Steht ⟨HL⟩; so bedeutet das:
Inhalt der Speicherzelle, deren Adresse in HL steht.
– Steht ⟨nn⟩; m so heißt das:
Inhalt der Speicherzelle, deren Adresse nn ist.
Bedeutung der Symbole für die Flagstellung (Bedeutung des Merkbits):

↕ Das Flag wird in Abhängigkeit vom Ergebnis der Operation beeinflußt.
• Das Flag bleibt unbeeinflußt.
0 Das Flag wird durch die Operation rückgesetzt.
1 Das Flag wird durch die Operation gesetzt.
v Das Flag wird in Abhängigkeit vom Überlauf des Ergebnisses beeinflußt.
P Das Flag wird in Abhängigkeit von der Parität des Ergebnisses beeinflußt.
X Das Flag ist beliebig.

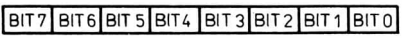


Bild 4.12 Bit-Numerierung innerhalb eines Bytes

4.1.5.2. Transportbefehle

Einzelworttransfer

LD r,s s → r C S Z P/V H N
. . . . . .

Der Inhalt des Registers s oder eine Zahl n wird in ein Register r gebracht.

r kann sein: – Register A, B, C, D, E, H, L.

s kann sein: – Direktoperand n (8-Bit-Zahl);
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird;
– Register A, B, C, D, E, H, L.

Beispiel

LD C, ⟨IX + 19H⟩

Der Inhalt von IX sei 25AFH. Durch den obigen Befehl wird der Inhalt der Zelle 25AFH + 19H = 25C8H nach Register C gebracht.

LD d,s s → d C S Z P/V H N
. . . . . .

Der Inhalt des Registers s oder eine Zahl n wird nach der Zelle d gebracht.

s kann sein: – Register A, B, C, D, E, H, L oder eine Zahl n;

d kann sein: – ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird;
– Register A, B, C, D, E, H, L.

Beispiel

LD ⟨HL⟩, 28H

Der Inhalt von HL sei 25A5H. Durch obigen Befehl wird die Zahl 28H nach der Zelle 25A5H gebracht.

LD A,s s → A C S Z P/V H N
· ↕ ↕ IFF O O bei s = I,R
· · · · · O bei s nicht I,R

Der Inhalt einer Zelle s wird nach Register A gebracht.
s kann sein: – ⟨BC⟩, ⟨DE⟩, d. h. eine Speicherzelle, deren Adresse in BC oder DE steht;
– (nn), d. h. eine Speicherzelle, deren Adresse nn ist;
– Register I oder R.

Beispiel
LD A, ⟨DE⟩
Der Inhalt von DE sei 8A25H. Durch obigen Befehl wird der Inhalt von Zelle 8A25H nach Register A gebracht.

LD d, A A → d	C	S	Z	P/V	H	N
	.	.	.	.	.	.

Der Inhalt des Registers A wird nach der Zelle d gebracht.
d kann sein: – ⟨BC⟩, ⟨DE⟩, d. h. eine Speicherzelle, deren Adresse in BC oder DE steht;
– (nn), d. h. eine Speicherzelle, deren Adresse nn ist,
– Register I oder R.

Beispiel
LD ⟨25 H⟩, A
Der Inhalt des Registers A wird nach der Zelle 25 H gebracht.

Doppelworttransfer

LD dd, nn nn → dd	C	S	Z	P/V	H	N
	.	.	.	.	.	.

Der Direktoperand nn wird in das Doppelregister dd gebracht.
dd kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD HL, 28DEH
Die Zahl 28DEH wird nach Register HL gebracht.

LD dd, ⟨nn⟩ ⟨nn⟩ → dd	C	S	Z	P/V	H	N
	.	.	.	.	.	.

Der Inhalt von Zelle nn und nn + 1 wird in das Doppelregister dd gebracht.
dd kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD IX, ⟨8AH⟩
Der Inhalt von Zelle 8AH und 8BH wird in das Indexregister IX gebracht.

LD ⟨nn⟩, ss ss → ⟨nn⟩ C S Z P/V H N
. . . . . .

Der Inhalt des Doppelregisters ss wird in die Speicherzelle nn und nn + 1 gebracht.
ss kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD ⟨20H⟩, SP
Der Inhalt des SP wird nach Zelle 20H gebracht.

LS SP, ss ss → SP C S Z P/V H N
. . . . . .

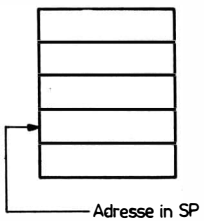
Der Inhalt des Doppelregisters ss wird in den Stackpointer SP gebracht.
ss kann sein: HL, IX, IY.

Beispiel
LD, SP, IX
Der Inhalt des Indexregisters IX wird in den Stackpointer SP gebracht.

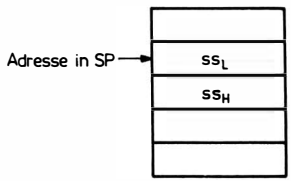
PUSH ss ss_H, ss_L → ⟨SP-1⟩, ⟨SP-2⟩ C S Z P/V H N
SP-2 → SP . . . . . .

Der Inhalt des Doppelregisters ss wird in den Kellerspeicher gebracht. Der höherwertige Teil ss_H kommt in die Zelle SP-1. Der niederwertige Teil in die Zelle SP-2. Nach Ausführung des Befehls ist der Inhalt des Stackpointers SP um 2 erniedrigt.

Kellerspeicher vor Ausführung von PUSH ss

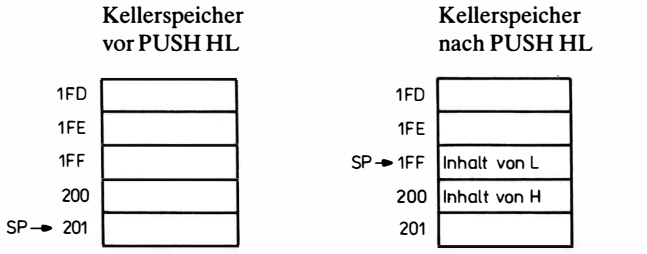


Kellerspeicher nach Ausführung von PUSH ss



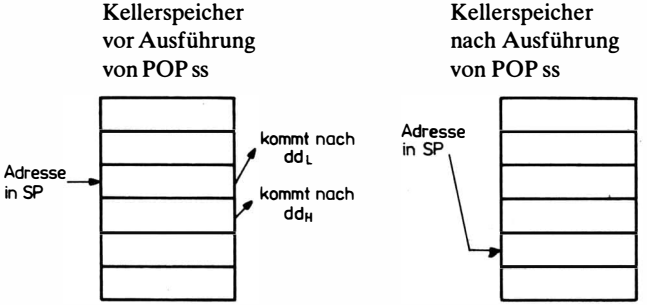
ss kan sein: BC, DE, HL, AF, IX, IY.

Beispiel
Der Inhalt von SP ist 201H



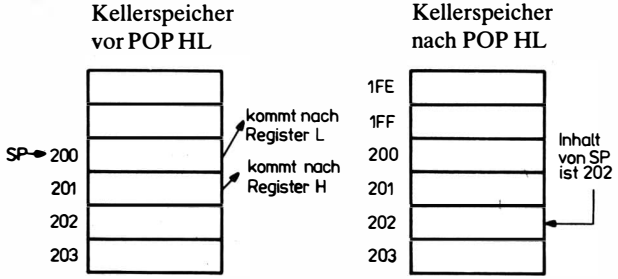
POP ss $\langle SP + 1\rangle, \langle SP\rangle \rightarrow dd_H, dd_L$ C S Z P/V H N
$SP + 2 \rightarrow SP$ . . . . . .

Aus dem Kellerspeicher werden 2 Byte in das Doppelregister dd gebracht. Das Byte aus Zelle SP kommt in den niederwertigen Teil von dd, das Byte aus Zelle SP + 1 in den höherwertigen Teil von dd. Nach Ausführung des Befehls ist der Inhalt des Stackpointers SP um 2 erhöht.



ss kann sein: BC, DE, HL, AF, IX, IY.

Beispiel
POP HL
Der Inhalt von SP sei 200H



Doppelworttransfer-Umtauschbefehle

EX, DE, HL $DE \leftrightarrow HL$ C S Z P/V H N
. . . . . .

Der Inhalt des Registerpaares DE wird mit dem Inhalt des Registerpaares HL vertauscht.

EX, AF, AF′, $AF \leftrightarrow A'F'$ C S Z P/V H N
. . . . . .

Die Inhalte der Register A und F werden mit den Inhalten der Register A′ und F′ vertauscht, und zwar A mit A′ und F mit F′.

EXX $BC \leftrightarrow B'C'$ C S Z P/V H N
$DE \leftrightarrow D'E'$ . . . . . .
$HL \leftrightarrow H'L'$

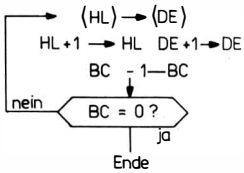
Es werden die Inhalte der Register B mit B′, C mit C′, D mit D′, E mit E′, H mit H′ und L mit L′ vertauscht.

EX $\langle SP \rangle$, ss $ss_H, ss_L \leftrightarrow \langle SP+1 \rangle, \langle SP \rangle$ C S Z P/V H N
. . . . . .

Der Inhalt des Doppelregisters ss wird mit dem Inhalt von 2 Zellen des Kellerspeichers vertauscht. Es wird dabei der niederwertige Teil ss_L des Doppelregisters mit dem Inhalt der Zelle, deren Adresse in SP steht, und der höherwertige Teil ss_H mit der nächsten Zelle (Adresse SP + 1) vertauscht. Am Ende steht in SP der gleiche Wert wie vorher.
ss kann sein: HL, IX, IY.

Blocktransfer

LDIR

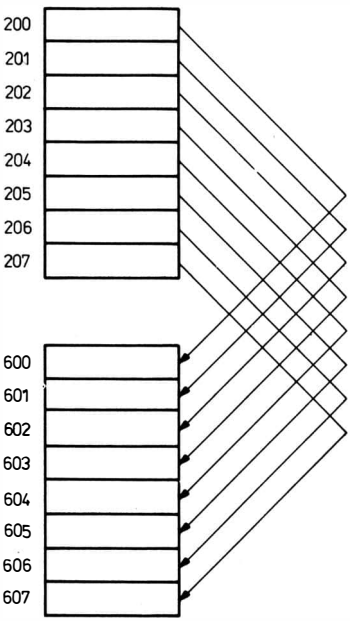


C	S	Z	P/V	H	N
·	·	·	0	0	0

Es wird der Inhalt eines Speicherbereiches, dessen Anfangsadresse in HL und dessen Blocklänge (Anzahl der Zellen des Speicherbereiches) in BC steht, in einem Speicherbereich mit der Anfangsadresse, die in DE steht, gespeichert.

Beispiel

Der Inhalt von HL sei 200, der von DE 600 und der von BC 8. Durch LDIR wird der Inhalt der Zellen 200 bis 207 in den Zellen 600 bis 607 gespeichert.



LDI $\langle HL\rangle \rightarrow \langle DE\rangle$
$HL + 1 \rightarrow HL \quad DE + 1 \rightarrow DE$
$BC - 1 \rightarrow BC$

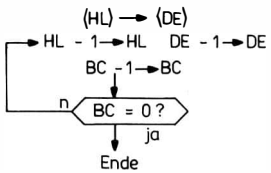
C	S	Z	P/V	H	N
·	·	·	↕	0	0

Dieser Befehl dient zur Umspeicherung eines Speicherbereiches, dessen Anfangsadresse in HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Anfangsadresse in DE steht. Durch eine einmalige Abarbeitung des Befehls wird der Inhalt der Zelle, deren Adresse in HL steht, in die Speicherzelle gebracht, deren Adresse in DE steht. Anschließend werden die Adressen in HL und DE um 1 erhöht und der Inhalt von BC um 1 erniedrigt. Ist $BC - 1 = 0$, so wird das Flag $P/V = 0$, sonst wird $P/V = 1$ gesetzt. Durch mehrmaliges Anwenden dieses Befehls läßt sich der Inhalt eines Speicherbereiches in einen anderen Speicherbereich umspeichern. Dabei kann entweder bei $BC = 0$ oder bei einer anderen Bedingung abgebrochen werden.

Beispiel
Der Inhalt von HL sei 300, der von DE 500 und der von BC sei 12. Bei der 1. Befehlsabarbeitung von LDI kommt der Inhalt von Zelle 300 nach Zelle 500. Bei der 2. Abarbeitung von LDI wird der Inhalt von Zelle 301 nach Zelle 501 gebracht usw. Nach 12 Durchläufen (Inhalt von BC) ist der Inhalt von $BC = 0$, was als Endbedingung genommen werden kann.

LDDR

C	S	Z	P/V	H	N
·	·	·	0	0	0



Es wird der Inhalt eines Speichers, dessen Endadresse in HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Endadresse in DE steht, gebracht.

Beispiel
Der Inhalt von HL sei 200H, der von DE 600H und der von BC 8H. Durch LDDR gelangt der Inhalt der Zellen 1F9-200H in die Zellen 5F9-600H.

			C	S	Z	P/V	H	N
LDD	⟨HL⟩	→ ⟨DE⟩						
	HL − 1	→ HL, DE − 1 → DE	·	·	·	↕	0	0
	BC − 1	→ BC						

Dieser Befehl dient zur Umspeicherung eines Speicherbereiches, dessen Endadresse in HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Endadresse in DE steht.
Durch eine einmalige Abarbeitung des Befehls wird der Inhalt der Zelle, deren Adresse in HL steht, in die Speicherzelle gebracht, deren Adresse in DE steht. Anschließend werden die Adressen in HL und DE um 1 erniedrigt und der Inhalt von BC um 1 erniedrigt. Ist BC − 1 = 0, so wird das Flag P/V = 0, sonst wird P/V = 1 gesetzt. Durch mehrmaliges Anwenden dieses Befehls läßt sich der Inhalt eines Speicherbereiches in einen anderen Speicherbereich umspeichern. Dabei kann entweder bei BC = 0 oder bei einer anderen Bedingung abgebrochen werden.

4.1.5.3. Rechen- und logische Operationen mit einem Operand

Akkumulator- und C-Bit-Befehle

		C	S	Z	P/V	H	N
CPL	$\bar{A} \rightarrow A$	·	·	·	·	1	1

Der Inhalt des Registers A wird bitweise negiert.

Beispiel
Der Inhalt von Register A sei 11001110.
Nach Ausführung des Befehls CPL ist der Inhalt des Registers A 00110001.

		C	S	Z	P/V	H	N
NEG	$\bar{A} + 1 \rightarrow A$ oder $0 - A \rightarrow A$	↕	↕	↕	V	↕	1

Vom Inhalt des Registers A wird das Zweierkomplement gebildet.

Beispiel
Der Inhalt von Register A sei 11001110.
Nach Ausführung des Befehls NEG ist der Inhalt des Registers A 00110010.

CCF $\bar{C} \rightarrow C$

C	S	Z	P/V	H	N
↕	·	·	·	↕	0

Der Inhalt des C-Bits wird negiert.
Im H-Bit wird der vorherige Wert des C-Bit gespeichert.

SCF $1 \rightarrow C$

C	S	Z	P/V	H	N
1	·	·	·	0	0

Der Inhalt des C-Bits wird 1 gesetzt.

DAA

C	S	Z	P/V	H	N
↕	↕	↕	P	↕	·

Der Befehl DAA dient im Zusammenhang mit der Addition und der Subtraktion von Dualzahlen zur Berechnung von Summen oder Differenzen zweier im BCD-Code dargestellten Zahlen.

Beispiel
Im Register A stehe die Zahl 28 im BCD-Code, d. h., der Inhalt von Register 00101000. Im Register B stehe die Zahl 17 im BCD-Code, d. h., B = 00010111. Nach der dualen Addition der Inhalte von Register A und B steht im Register A 00111111.
Das ist jedoch nicht die BCD-Code-Darstellung von 28 + 17 = 45. Der Befehl DAA verändert A = 00111111 in den Wert A = 01000101 (= 45 in BCD-Darstellung).

Einzelwortbefehle
INC d $d + 1 \rightarrow d$

C	S	Z	P/V	H	N
·	↕	↕	P	↕	0

Der Inhalt der Zelle oder des Registers d wird um 1 erhöht.
d kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. Inhalt einer Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Beispiel
Der Inhalt vom Indexregister IY sei 1A5H.
Durch den Befehl INC ⟨IY + 17H⟩ wird der Inhalt von Zelle 1A5H + 17H = 1BCH um 1 erhöht.

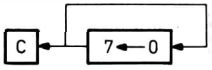
DEC d d − 1 → d

C	S	Z	P/V	H	N
·	↕	↕	P	↕	1

Der Inhalt der Zelle oder des Registers d wird um 1 erniedrigt,
d kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Beispiel
Der Inhalt des Doppelregisters HL sei 800.
Durch den Befehl DEC ⟨HL⟩ wird der Inhalt der Zelle 800 um 1 erniedrigt.

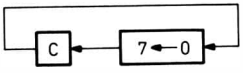
Verschiebebefehle
RLC s
RLCA[5]



	C	S	Z	P/V	H	N
RLC s	↕	↕	↕	P	0	0
RLCA	↕	·	·	·	0	0

Der Inhalt des Registers oder der Zelle s wird um eine Stelle nach links verschoben. Das aus Bit 7 (Zählung von rechts nach links) heraustretende Bit wird in das C-Bit und Bit 0 eingetragen.
s kann sein:– Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RL s
RLA[6]



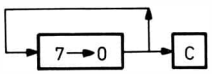
	C	S	Z	P/V	H	N
RL s	↕	↕	↕	P	0	0
RLA	↕	·	·	·	0	0

Der Inhalt des Registers s oder der Zelle s wird zusammen mit dem C-Bit um eine Stelle nach links verschoben. Das aus Bit 7 kommende Bit wird in das C-Bit und das C-Bit in Bit 0 eingetragen.

[5] RLCA führt dieselbe Funktion wie RLC A aus, ist jedoch ein 1-Byte-Befehl.
[6] RLA und RRCA führen dieselbe Verschiebung wie RL A und RRC A aus, sind aber 1-Byte-Befehle.

s kann sein:– Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RRC s
RRCA[6]

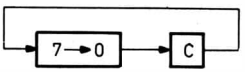


	C	S	Z	P/V	H	N
RRCs	↕	↕	↕	P	0	0
RRCA	↕	·	·	·	0	0

Der Inhalt des Registers s oder der Zelle s wird um eine Stelle nach rechts verschoben. Das aus Bit 0 heraustretende Bit wird in das C-Bit und in Bit 7 eingetragen.

s kann sein:– Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + D⟩, ⟨IY + d⟩, d. h. ein Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RR s
RRA[7]

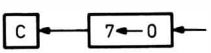


	C	S	Z	P/V	H	N
RRs	↕	↕	↕	P	0	0
RRA	↕	·	·	·	0	0

Der Inhalt des Registers s oder der Zelle s wird zusammen mit dem C-Bit um eine Stelle nach rechts verschoben. Bit 0 kommt ins C-Bit und das C-Bit nach Bit 7.

s kann sein:– Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

SLA s

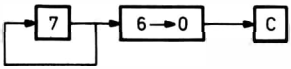


C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine arithmetische Linksverschiebung des Registers oder der Speicherzelle s. Das aus Bit 7 heraustretende Bit wird in das Register C-Bit eingetragen.
In Bit 0 wird eine 0 eingetragen. Der Befehl entspricht der Multiplikation des Registerinhalts mit 2.

[7]RRA führt dieselbe Verschiebung wie RR A aus, ist aber ein 1-Byte-Befehl.

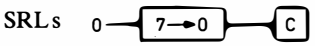
s kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

SRA s 

C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine arithmetische Rechtsverschiebung des Registers oder der Speicherzelle s. Bit 7 bleibt erhalten. In Bit 6 wird Bit 7 eingetragen, Bit 0 kommt ins C-Bit.

s kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

SRL s 

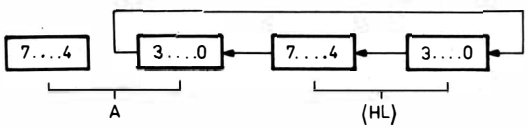
C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine Rechtsverschiebung des Registers oder der Zelle s. Dabei wird in Stelle 7 eine 0 und in das C-Bit Bit 0 eingetragen.

s kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RLD

C	S	Z	P/V	H	N
·	↕	↕	P	0	0



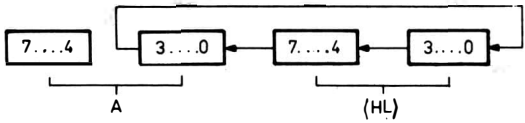
Dieser Befehl bewirkt ein Linksverschiebung um eine Tetrade (4 Bit) des Registers A und einer Speicherzelle, deren Adresse in HL steht.

Dabei werden eingetragen:
– die niederwertige Tetrade von A in die niederwertige Tetrade der Speicherzelle

- die niederwertige Tetrade der Speicherzelle in die höherwertige Tetrade der Speicherzelle und
- die höherwertige Tetrade der Speicherzelle in die niederwertige Tetrade des Registers A.

RRD

C	S	Z	P/V	H	N
·	↕	↕	P	0	0



Dieser Befehl bewirkt eine Rechtsverschiebung um eine Tetrade (4 Bit) des Registers A und einer Speicherzelle, deren Adresse in HL steht.
Dabei werden eingetragen:
- die niederwertige Tetrade von A in die höherwertige Tetrade der Speicherzelle,
- die höherwertige Tetrade der Speicherzelle in die niederwertige Tetrade der Speicherzelle und die
- niederwertige Tetrade der Speicherzelle in die niederwertige Tetrade des Registers A.

Beispiel (zu den Verschiebebefehlen)
Im Register D stehe die Bit-Folge 01110011 und im C-Bit eine 1.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	1	1	1	0	0	1	1
C-Bit	Register D							

Befehl: Information im D-Register und C-Bit nach dem Befehl:

RLC D	0	11100110
RL D	0	11100111
RRC D	1	10111001
RR D	1	10111001
SLA D	0	11100110
SRA D	1	00111001
SRL D	1	00111001
	C-Bit	Register D

Bit-Befehle

Bit b, s $s_b \rightarrow Z$

C	S	Z	P/V	H	N
·	X	↕	X	1	0

Das Bit b des Registers oder der Speicherzelle s wird in negierter Form in das Z-Flag gebracht, b ist eine Zahl zwischen 0 und 7.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Bit-Zählung

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Beispiel

Bit 3, ⟨HL⟩

Bit 3 der Speicherzelle, deren Adresse in HL steht, wird in negierter Form in das Z-Flag gebracht.
- In HL steht die Adresse 80H.
- In der Zelle 80H steht 11101110.
- Durch Bit 3, ⟨HL⟩ wird eine 0 in das Z-Flag gebracht.

SET b, s $1 \rightarrow s_b$

C	S	Z	P/V	H	N
·	·	·	·	·	·

Das Bit b des Registers oder der Speicherzelle s wird 1 gesetzt, b ist eine Zahl zwischen 0 und 7.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RES b, s $0 \rightarrow s_b$

C	S	Z	P/V	H	N
·	·	·	·	·	·

Das Bit b des Registers oder der Speicherzelle s wird 0 gesetzt.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Doppelwortbefehle

INC dd dd + 1 → dd C S Z P/V H N
. . . . . .

Der Inhalt des Doppelregisters dd wird um 1 erhöht.
dd kann sein: BC, DE, HL, SP, IX, IY

DEC dd dd − 1 → dd C S Z P/V H N
. . . . . .

Der Inhalt des Doppelregisters dd wird um 1 erniedrigt.
dd kann sein: BC, DE, HL, SP, IX, IY.

4.1.5.4. Rechen- und logische Operationen mit zwei Operanden

1-Wort-Befehle

ADD s A + s → A C S Z P/V H N
↕ ↕ ↕ V ↕ 0

Der Inhalt des Akkumulators und der Inhalt des Registers oder der Zelle s werden adddiert, und das Ergebnis wird ins Register A gebracht.

s kann sein:
- Direktoperand n (8-Bit-Zahl);
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine durch Indexierung ermittelte Speicherzelle;
- Register A, B, C, D, E, H, L.

ADC s A + s + C → A C S Z P/V H N
↕ ↕ ↕ V 0 ↕

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden addiert. Zu diesem Ergebnis wird das C-Bit in die niedrigste Stelle addiert. Das Gesamtergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

SUB s A − s → A C S Z P/V H N
↕ ↕ ↕ V ↕ 1

Der Inhalt der Zelle oder des Registers s wird vom Register A subtrahiert. Das Ergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

SBC s A − s − C → A C S Z P/V H N
↕ ↕ ↕ V ↕ 1

Der Inhalt des Registers oder der Zelle s wird vom Inhalt des A-Registers subtrahiert. Von der niedrigsten Stelle des Ergebnisses wird das C-Bit subtrahiert. Das letzte Ergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

		C	S	Z	P/V	H	N
AND s	A ∧ s → A	0	↕	↕	P	1	1

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches UND verknüpft. Das Ergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

		C	S	Z	P/V	H	N
OR s	A ∨ s → A	0	↕	↕	P	1	1

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches ODER verknüpft. Das Ergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

		C	S	Z	P/V	H	N
XOR s	A ⊕ s → A	0	↕	↕	P	1	0

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches EXKLUSIV-ODER verknüpft. Das Ergebnis kommt in das Register A.
s kann sein: siehe 1-Wort-Befehl ADD s.

	C	S	Z	P/V	H	N
CP s	↕	↕	↕	V	↕	1

Der Inhalt des Registers A wird mit dem Inhalt des Registers oder der Zelle s verglichen. Der Vergleich erfolgt durch die Bildung der Differenz A − s. Bei Gleichheit wird das Z-Bit gesetzt, bei Ungleichheit rückgesetzt.
s kann sein: siehe 1-Wort-Befehl ADD s.

Doppelwortbefehle

		C	S	Z	P/V	H	N
ADD HL, ss	HL + ss → HL	↕	·	·	·	↕	0

Der Inhalt des Doppelregisters HL wird mit dem Inhalt des Doppelregisters ss addiert. Das Ergebnis kommt nach HL. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.
ss kann sein: BC, DE, HL, SP.

ADC HL, ss HL + ss + C → HL

C	S	Z	P/V	H	N
↕	↕	↕	V	↕	0

Der Inhalt des Doppelregisters HL wird mit dem Inhalt des Doppelregisters ss addiert. Zum Ergebnis wird das C-Bit in die niedrigste Stelle addiert. Das letzte Ergebnis kommt nach HL. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.
ss kann sein: BC, DE, HL, SP.

SBC HL, ss HL − ss − C → HL

C	S	Z	P/V	H	N
↕	↕	↕	V	↕	1

Der Inhalt des Doppelregisters ss wird vom Doppelregister HL subtrahiert. Anschließend wird davon das C-Bit in der letzten Stelle subtrahiert. Das Ergebnis kommt nach HL, Bit H wird vom Übertrag von Bit 12 gesetzt.
ss kann sein: BC, DE, HL, SP.

ADD IX, ss IX + ss → IX

C	S	Z	P/V	H	N
↕	·	·	·	↕	0

Der Inhalt des Doppelregisters ss wird zum Inhalt des Indexregisters IX addiert. Das Ergebnis kommt nach IX. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.
ss kann sein: BC, DE, IX, SP.

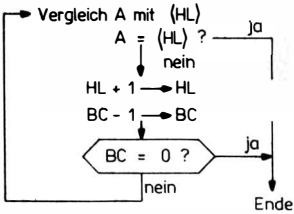
ADD IY, ss IY + ss → IY

C	S	Z	P/V	H	N
↕	·	·		↕	0

Der Inhalt des Doppelregisters ss wird zum Inhalt des Indexregisters IY addiert. Das Ergebnis kommt nach IY. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.
ss kann sein: BC, DE, IY, SP.

4.1.5.5. Rechen- und logische Operationen mit mehreren Operanden

CPIR



C	S	Z	P/V[8]	H	N
·	↕	↕	↕	↕	1

Der Inhalt eines Speicherbereiches, dessen Anfangsadresse in HL und dessen Länge in BC steht, wird nach einer Information im Register A durchsucht. Befindet sich die Information in einer Zelle des Speicherbereiches, so ist der Befehl an dieser Stelle beendet. Befindet sich die Information nicht in dem gesuchten Speicherbereich, so wird der Befehl nach der letzten Zelle des Bereiches beendet. Wird die Information im Speicherbereich gefunden, so wird trotzdem HL um 1 erhöht und BC um 1 erniedrigt. Im Flagregister steht das Ergebnis der letzten Vergleichsoperation. Das Bit P/V ist 1, wenn BC − 1 ≠ 0, sonst 0.

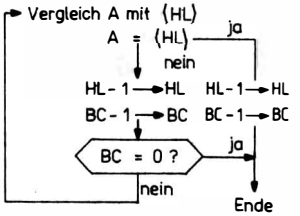
Beispiel
Die Zellen 200 H bis 205 H sollen nach der Bit-Folge 00000111 durchsucht werden.
In den Zellen 200 H bis 205 H stehen:
200 H 11111111
201 H 00000000
202 H 01000000
203 H 00000111
204 H 00000000
205 H 10000001
Dazu bringt man die Bit-Folge 00000111 in das Register A, die Adresse 200 H ins Registerpaar Hl und die Anzahl 6 ins Registerpaar BC. Nach der Ausführung des Befehls CPIR steht in HL 204 H, und das Z-Bit ist gesetzt.
Würde in Zelle 203 die Bit-Folge 01010101 stehen, so stünde nach Abarbeitung des Befehls CPIR in Hl 206H, und das Z-Bit würde rückgesetzt werden.

[8]Wenn BC zum Befehlsende 0 ist, dann wird P/V = 0; ist BC ≠ 0, so wird P/V = 1 gesetzt.

CPI	Vergleich A mit ⟨HL⟩	C	S	Z	P/V[9]	H	N
	HL + 1 → HL	·	↕	↕	↕	↕	1
	BC − 1 → BC						

Der Inhalt des Speicherplatzes, dessen Adresse in HL steht, wird mit dem Inhalt des A-Registers verglichen und das Flagregister gesetzt. Das P/V-Flag ist 1, wenn BC − 1 ≠ 0, sonst 0. Anschließend wird der Inhalt von HL um 1 erhöht und der Inhalt von BC um 1 erniedrigt. Mit Hilfe des Befehls CPI läßt sich ein Speicherbereich auf eine Bit-Folge, die im A-Register steht, durchsuchen, wobei die Abbruchbedingung frei wählbar ist.

CPDR



C	S	Z	P/V[9]	H	N
·	↕	↕	↕	↕	1

Der Inhalt eines Speicherbereiches, dessen Endadresse in HL und dessen Länge in BC steht, wird nach einer Information, die im Register A steht, durchsucht. Befindet sich die Information in einer Zelle des Speicherbereiches, so wird der Befehl an dieser Stelle beendet. Steht die Information nicht in dem gesuchten Speicherbereich, so endet der Befehl nach der ersten Zelle des Bereiches. Wird die Information im Speicherbereich gefunden, so werden trotzdem HL und BC um 1 erniedrigt.
Im Flagregister steht das Ergebnis der letzten Vergleichsoperation. Das Bit P/V ist 1, wenn BC − 1 ≠ 0, sonst 0.

Beispiel
Die Zellen 200H bis 205H sollen nach der Bit-Folge 00000111 durchsucht werden. In den Zellen 200H bis 205H stehen:

200H	11111111
201H	00000000
202H	01000000
203H	00000111
204H	00000000
205H	10000001

[9] Wenn BC nach Befehlsausführung 0 ist, dann wird P/V = 0; ist BC ≠ 0, so wird P/V = 1 gesetzt.

Dazu bringt man die obige Bit-Folge in das Register A, die Adresse 205H in das Register HL und die Anzahl 6 in das Registerpaar BC. Nach Ausführung des Befehls CPDR steht in HL 202H, und das Z-Bit ist gesetzt.
Würde sich in Zelle 203H die Bit-Folge 01010101 befinden, so stünde nach Abarbeitung des Befehls CPDR in HL 1FFH, und das Z-Bit würde rückgesetzt sein.

		C	S	Z	P/V[10]	H	N
CPD	Vergleich A mit ⟨HL⟩ HL − 1 → HL BC − 1 → BC	·	↕	↕	↕	↕	1

Der Inhalt des Speicherplatzes, dessen Adresse in HL steht, wird mit dem Inhalt des A-Registers verglichen und das Flagregister gesetzt. Das P/V-Flag ist 1, wenn BC − 1 ≠ 0, sonst 0. Anschließend werden der Inhalt von HL und der von BC um 1 erniedrigt. Mit Hilfe des Befehls CPD läßt sich ein Speicherbereich auf eine Bit-Folge, die im A-Register steht, durchsuchen, wobei die Abbruchbedingung frei wählbar ist.

4.1.5.6. Sprungbefehle

	C	S	Z	P/V	H	N
JP nn nn → PC	·	·	·	·	·	·

Die Adresse nn wird in den Befehlszähler PC gebracht. Der nächste abzuarbeitende Befehl ist damit der Befehl aus Zelle nn. Man sagt, der Rechner führt einen Sprung in die Zelle nn aus.

	C	S	Z	P/V	H	N
JP cc, nn Wenn cc = true, nn → PC	·	·	·	·	·	·

Wenn die Bedingung cc erfüllt ist, dann wird die Adresse nn in den Befehlszähler PC gebracht.
cc kann sein:
- NZ Z-Bit rückgesetzt,
- Z Z-Bit gesetzt,
- NC C-Bit rückgesetzt,
- C C-Bit gesetzt,
- PO Paritätsbit auf ungerade gesetzt,
- PE Paritätsbit auf gerade gesetzt,
- P Vorzeichenbit auf postiv gesetzt,
- M Vorzeichenbit auf negativ gesetzt.

[10]s. Fußnote S. 143

JR e PC + e → PC C S Z P/V H N
. . . . . .

Der Befehlszähler PC wird um die Zahl e verändert. e ist eine 8-Bit-Zahl im Zweierkomplement, d. h., e kann auch negativ sein. Ausgangspunkt für PC + e → PC ist die Adresse des Operationscodes des Befehls. Der Rechner führt einen Sprung um e Zellen aus. Die Zahl e steht als e − 2 in der Zelle nach dem Operationscode.

Beispiel
Von Zelle 100 soll ein Sprung nach 105 erfolgen.
Befehl: JR 5

100	18	← Operationscode des Befehls IR
101	3	← e − 2
102	——	
103	———	
104	———	
→105	———	← PC-Stand nach dem Befehl.

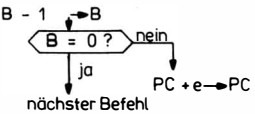
JR kk, e Wenn kk = true, PC + e → PC C S Z P/V H N
wenn kk = not true, Leerbefehl . . . . . .

Wenn die Bedingung kk erfüllt ist, dann führt der Prozessor einen Sprung um e Zellen aus.
kk kann sein: – NZ Z-Bit rückgesetzt,
– Z Z-Bit gesetzt,
– NC C-Bit rückgesetzt,
– C C-Bit gesetzt.

JP ⟨ss⟩ ss → PC C S Z P/V H N
. . . . . .

Der Inhalt des Doppelregisters ss wird nach PC gebracht. Der Rechner führt einen Sprung in die Zelle aus, deren Adresse in ss steht.
ss kann sein: HL, IX, IY.

DJNZ e

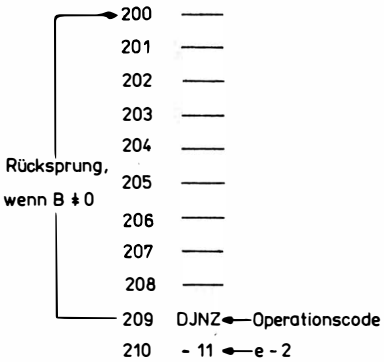


C S Z P/V H N
. . . . . .

Solange B − 1 ≠ 0 ist, führt der Rechner einen Sprung um e Zellen aus. e ist eine 8-Bit-Zahl im Zweierkomplement, d. h. e kann auch negativ sein. Ausgangspunkt für PC + e → PC ist die Adresse des

Operationscodes des Befehls. Die Zahl e steht als e − 2 in der Zelle nach dem Operationscode.

Beispiel
Eine Programmschleife von Zelle 200 bis 209 soll 10mal durchlaufen werden. Dazu bringt man eine 10 in das Register B. Der Befehl, der diese 10fache Programmschleife realisiert, lautet DJNZ − 9.
Aufbau der Progammschleife:



4.1.5.7. Unterprogrammbefehle

Unterprogrammrufe
CALL nn PC_H, PC_L → ⟨SP − 1⟩, ⟨SP − 2⟩ C S Z P/V H N
. . . . . .

SP − 2 → SP
nn → PC

Der CALL-Befehl realisiert einen Sprung in ein Unterprogramm, dessen Startadresse nn ist. Dabei wird der aktuelle Befehlszählerstand (Adresse des Operationscodes des nächsten Befehls) im STACK gespeichert. Der höherwertige Teil von PC kommt in die Zelle, deren Adresse sich aus den um 1 erniedrigten Inhalt des STACKPOINTER SP ergibt. Der niederwertige Teil von PC gelangt in die Zelle, deren Adresse sich aus dem um 2 erniedrigten Inhalt des STACKPOINTER ergibt. Der STACKPOINTER ist am Ende des Befehls um 2 erniedrigt.

CALL cc, nn wenn cc = true C S Z P/V H N

$PC_H, PC_L \rightarrow \langle SP-1\rangle, \langle SP-2\rangle$ · · · · · ·

$SP - 2 \rightarrow SP$

$nn \rightarrow PC$

Wenn die vorgegebene Bedingung cc erfüllt ist, so wird der Befehl CALL nn ausgeführt. Ist die Bedingung nicht erfüllt, dann hat der Befehl die Funktion eines Leerbefehls.

cc kann sein:
- NZ Z-Bit rückgesetzt,
- Z Z-Bit gesetzt,
- NC C-Bit rückgesetzt,
- C C-Bit gesetzt,
- PO Paritätsbit auf ungerade gesetzt,
- PE Paritätsbit auf gerade gesetzt,
- P Vorzeichenbit auf positiv gesetzt,
- M Vorzeichenbit auf negativ gesetzt.

RST z, $PC_H, PC_L \rightarrow \langle SP-1\rangle, \langle SP-2\rangle$ C S Z P/V H N

$SP - 2 \rightarrow SP$ · · · · · ·

$z \rightarrow PC$

z ist eine der Hexadezimaladressen 0H, 8H, 10H, 18H, 20H, 28H, 30H, 38H.

Der Befehl RST z (RESTART) ist ein CALL-Befehl mit der Adresse z. Er wird in Verbindung mit dem INTERRUPT in MODE 0 verwendet. Hier muß während dem INTERRUPT-Zyklus über den Datenbus ein 1-Byte-Befehl in den Prozessor gegeben werden. Der RST-Befehl stellt einen 1-Byte-CALL-Befehl dar, während der vollständige CALL-Befehl 3 Byte lang ist.

Rückkehrbefehle

RET $\langle SP+1\rangle, \langle SP\rangle \rightarrow PC_H, PC_L$ C S Z P/V H N

$SP + 2 \rightarrow SP$ · · · · · ·

Der RET-Befehl realisiert den Rücksprung aus einem Unterprogramm. Durch ihn wird die zuletzt in den STACK gespeicherte Adresse in den Befehlszähler PC gebracht. Der Inhalt der Zelle dessen Adresse im STACKPOINTER steht, kommt in den niederwertigen Teil von C, der Inhalt der Zelle, dessen Adresse sich aus dem Inhalt des STACKPOINTERS plus 1 ergibt, in den höherwertigen Teil von PC. Der STACKPOINTER ist am Ende des Befehls um 2 erhöht.

RET cc wenn cc = true; C S Z P/V H N
⟨SP + 1⟩, ⟨SP⟩ → PC_H, PC_L · · · · · ·
SP + 2 → SP

Wenn die vorgegebene Bedingung cc erfüllt ist, dann wird der Befehl RET cc ausgeführt. Ist die Bedingung nicht erfüllt, so hat der Befehl die Funktion eines Leerbefehls.

cc kann sein:
- NZ Z-Bit zurückgesetzt,
- Z Z-Bit zurückgesetzt,
- NC C-Bit zurückgesetzt,
- C C-Bit gesetzt,
- PO Paritätsbit auf ungerade gesetzt,
- PE Paritätsbit auf gerade gesetzt,
- P Vorzeichenbit auf positiv gesetzt,
- M Vorzeichenbit auf negativ gesetzt,

RETI ⟨SP + 1⟩, ⟨SP⟩ → PC_H, PC_L C S Z P/V H N
SP + 2 → SP · · · · · ·
IFF2 → IFF1

Der Befehl RETI dient als Rücksprungbefehl beim maskierten INTERRUPT.

RETN ⟨SP + 1⟩, ⟨SP⟩ → PC_H, PC_L C S Z P/V H N
SP + 2 → SP · · · · · ·
IFF2 → IFF1

Der Befehl RETN dient als Rücksprungbefehl beim nichtmaskierten INTERRUPT.

4.1.5.8. Ein- und Ausgabebefehle

Einzelwortein- und -ausgabe

IN A, ⟨n⟩ ⟨n⟩ → A C S Z P/V H N
· ↕ ↕ P 0 0

Innerhalb eines Eingabezyklus wird das auf dem Datenbus vorhandene Byte in das Register A gebracht. Während des Eingabezyklus enhält der höherwertige Teil des Adreßbus den alten Inhalt des Registers A und der niederwertige Teil die Zahl n, die als Adresse für das periphere Gerät dient.

IN r, ⟨C⟩ ⟨C⟩ → r C S Z P/V H N
· · · · · ·

Innerhalb eines Eingabezyklus wird das auf dem Datenbus vorhan-

dene Byte in das Register r gebracht. Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des Registers B und der niederwertige Teil den Inhalt des Registers C, der als Adresse für das periphere Gerät dient.
r kann sein: Register A, B, C, D, E, H, L.

OUT ⟨n⟩, A A →⟨n⟩ C S Z P/V H N
. . . . . .

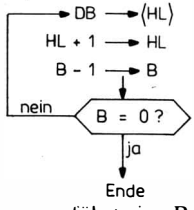
Innerhalb eines Ausgabezyklus wird der Inhalt des Registers A auf den Datenbus gebracht. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den alten Inhalt ddes Registers A und der niederwertige Teil die Zahl n, die als Adresse für das periphere Gerät dient.

OUT ⟨C⟩, r r → ⟨C⟩ C S Z P/V H N
. . . . . .

Innerhalb eines Ausgabezyklus wird der Inhalt des Registers r auf den Datenbus gebracht. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des Registers B und der niederwertige Teil den Inhalt des Registers C, der als Adresse für das periphere Gerät dient.
r kann sein: 1 Register A, B, C, D, E, H, L.

Blocktransfer

INIR C S Z P/V H N
· X1 X X 1



Adreßbus:
$A_{15}. . .A_8$ $A_7. . .A_0$
Inhalt des B-Registers
Inhalt des C-Registers

Der Prozessor führt eine Reihe Eingabebefehle durch, deren Anzahl n im Register B steht. Während jedes Eingabezyklus wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Nach jedem Eingabezyklus erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Insgesamt liest der Prozessor n Bytes ein, die in einen Speicherbereich gebracht werden, dessen Anfangsadresse in HL steht. Während eines Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des C-Registers, der als periphere Adresse dient.

INI DB $\rightarrow \langle HL \rangle$
HL + 1 $\rightarrow$ HL
B − 1 $\rightarrow$ B

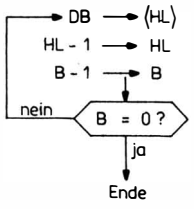
C	S	Z	P/V	H	N
·	X	↕	X	X	1

Adreßbus: $A_{15}. . .A_8$ $A_7. . .A_0$

Inhalt des B-Registers — Inhalt des C-Registers

Der Prozessor führt einen Eingabebefehl aus. Dabei wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Anschließend wird der Inhalt von HL um 1 erhöht und der Inhalt von B um 1 erniedrigt. Das Z-Bit wird 0, wenn B − 1 ≠ 0 wird, und 1, wenn B − 1 = 0 wird. Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

INDR



C	S	Z	P/V	H	N
·	X	1	X	X	1

Adreßbus: $A_{15}. . .A_8$ $A_7. . .A_0$

Inhalt des B-Registers — Inhalt des C-Registers

Der Prozessor führt eine Reihe Eingabebefehle durch, deren Anzahl n im Register B steht. Während jedes Eingabezyklus wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Nach jedem Eingabezyklus erniedrigt sich der Inhalt von HL und B um 1. Insgesamt liest der Prozessor n Bytes ein, die in einen Speicherbereich gebracht werden, dessen Endadresse in HL steht.
Während eines Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

IND DB $\rightarrow \langle HL \rangle$
HL − 1 $\rightarrow$ HL
B − 1 $\rightarrow$ B

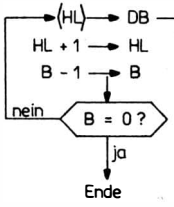
C	S	Z	P/V	H	N
·	X	↕	X	X	1

Adreßbus: $A_{15}. . .A_8$ $A_7. . .A_0$

Inhalt des B-Registers — Inhalt des C-Registers

Der Prozessor führt einen Eingabebefehl aus. Dabei wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL und B steht. Anschließend erniedrigt sich der Inhalt von HL um 1. Das Z-Bit wird 0, wenn $B - 1 \neq 0$, und 1, wenn $B - 1 = 0$ ist.
Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OTIR



C	S	Z	P/V	H	N
·	X	1	X	X	1

Adreßbus:

$A_{15} \ldots A_8$ — Inhalt des B-Registers

$A_7 \ldots A_0$ — Inhalt des C-Registers

Der Prozessor führt eine Reihe Ausgabebefehle durch, deren Anzahl n im Register B steht. Während jedes Ausgabezyklus wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus DB gebracht. Nach jedem Ausgabezyklus erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Insgesamt gibt der Prozessor n Bytes aus, die in einem Speicherbereich stehen, dessen Anfangsadresse in HL steht.
Während eines Ausbabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OUTI $\langle HL \rangle \rightarrow DB \rightarrow$ ext. Gerät
$HL + 1 \rightarrow HL$
$B - 1 \rightarrow B$

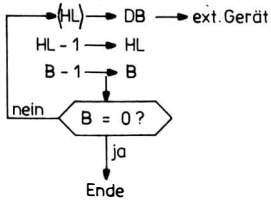
C	S	Z	P/V	H	N
·	X	↕	X	X	1

Adreßbus: $A_{15} \ldots A_8$ — Inhalt des B-Registers; $A_7 \ldots A_0$ — Inhalt des C-Registers

Der Prozessor führt einen Ausgabebefehl aus. Dabei wird der Inhalt der Speicherzelle, dessen Adresse in HL steht, auf den Datenbus DB gebracht. Anschließend erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Das Z-Bit wird 0, wenn $B - 1 \neq 0$ ist, und 1, wenn $B - 1 = 0$ ist. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den

Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OTDR



C	S	Z	P/V	H	N
·	X	1	X	X	1

Adreßbus:

$A_{15}. . .A_8$ (Inhalt des B-Registers) $A_7. . .A_0$ (Inhalt des C-Registers)

Der Prozessor führt eine Reihe Ausgabebefehle durch, deren Anzahl n im Register B steht. Während jedes Ausgabezyklus wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus gebracht. Nach jedem Ausgabezyklus erniedrigt sich der Inhalt von HL und B um 1. Insgesamt gibt der Prozessor n Bytes aus, die in einem Speicherbereich stehen, dessen Endadresse in HL steht.
Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OUTD $\langle HL \rangle \rightarrow DB \rightarrow$ ext. Gerät
$HL - 1 \rightarrow HL$
$B - 1 \rightarrow B$
Adreßbus: $A_{15}. . .A_8$ (Inhalt des B-Registers) $A_7. . .A_0$ (Inhalt des C-Registers)

C	S	Z	P/V	H	N
·	X	↕	X	X	1

Der Prozessor führt einen Ausgabebefehl aus. Dabei wird der Inhalt der Speicherzelle, dessen Adresse in HL steht, auf den Datenbus DB gebracht. Anschließend erniedrigt sich der Inhalt von HL und B um 1. Das Z-Bit wird 0, wenn $B - 1 \neq 0$ ist, und 1, wenn $B - 1 = 0$ ist.
Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

4.1.5.9. Steuerbefehle

HALT
Der Befehl HALT bringt den Prozessor in der STOP-Zustand. Während dieses Zustandes führt der Prozessor NOP-Befehle aus, wobei er über den Adreßbus die Auffrischadresse für dynamische Speicher aussendet. Den HALT-Zustand kann der Baustein durch INTERRUPT oder RESET verlassen.

NOP
Während NOP führt der Prozessor einen Leerzyklus aus.

DI
$0 \rightarrow$ IFF1, $0 \rightarrow$ IFF2
Durch den Befehl DI wird der maskierte INTERRUPT-Eingang geseperrt. Die beiden INTERRUPT-Flip-Flop IFF1 und IFF2 werden rückgesetzt.

EI
$1 \rightarrow$ IFF1, $1 \rightarrow$ IFF2
Durch den Befehl EI wird der maskierte INTERRUPT-Eingang geöffnet. Die beiden INTERRUPT-Flip-Flop IFF1 und IFF2 werden gesetzt.

IMO
IMO setzt den Prozessor in den INTERRUPT-MODE 0.

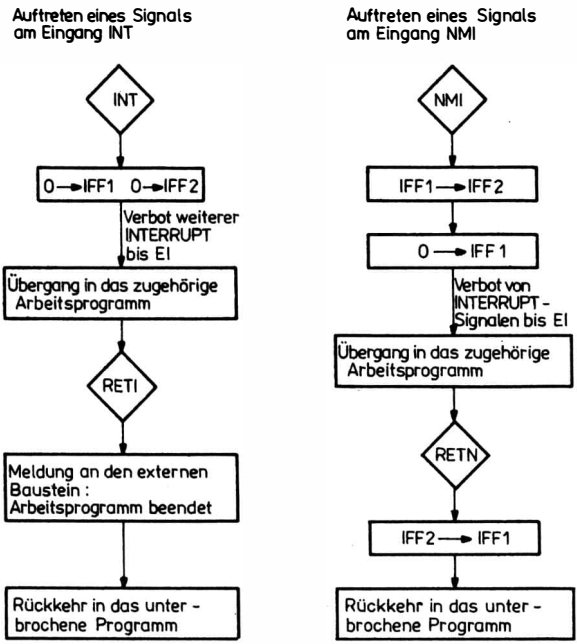
IMI
IMI setzt den Prozessor in den INTERRUPT-MODE 1.

IM2
IM2 setzt den Prozessor in den INTERRUPT-MODE 2.

4.1.6. INTERRUPT

Wie bereits beschrieben, bedeutet „INTERRUPT" Unterbrechung des gerade laufenden Programms und Übergang zu einem anderen Programm, das durch das unterbrechende Signal bestimmt wird. Durch den INTERRUPT-Eingang läßt sich also die Arbeitsweise des Prozessors steuern. Der Prozessor *U 880* hat 2 INTERRUPT-Eingänge.
- den nichtmaskierten NMI (NMI – nichtmaskierter INTERRUPT)

Ablauf bei Auftreten eines INTERRUPT-Signals



Die Rückkehr aus dem zugehörigen Arbeitsprogramm in das unterbrochene Programm geschieht über einen RETURN-Befehl.

- den maskierten Eingang INT (INT – INTERRUPT).
  Maskierbar heißt, daß das Eingangstor durch das Programm geöffnet und gesperrt werden kann. Das Öffnen geschieht durch einen speziellen Befehl EI. Für das Sperren gibt es den Befehl DI. Zur Steuerung des INTERRUPT-Eingangstors INT gibt es im Prozessor 2 Flip-Flop IFF1 und IFF2.
  IFF1 dient zur unmittelbaren Steuerung des INT-Tores. Ist es eingeschaltet, so ist das INT-Tor geöffnet; ist es ausgeschaltet, ist das INT-Tor gesperrt.
  Jedes von außen kommende INT-Signal schaltet das IFF1, nachdem es das laufende Programm unterbrochen hat, zunächst aus.

Wirkung der Befehle und Signale auf IFF1 und IFF2

	IFF1	IFF2	
RESET	0 → IFF1	0 → IFF2	INTERRUPT-
DI	0 → IFF1	0 → IFF2	MODE 0
EI	1 → IFF1	1 → IFF2	
LD A, I	bleibt	bleibt	IFF2 → Parity-Flag
LD A, R	bleibt	bleibt	IFF2 → Parity-Flag
INT	0 → IFF1	0 → IFF2	
NMI	0 → IFF1	IFF1 → IFF2	
RETN	IFF2 → IFF1	bleibt	
RETI	IFF2 → IFF1	bleibt	

Damit erreicht man, daß das durch das INT-Signal aufgerufene Programm nicht wieder unterbrochen werden kann. Das IFF1 wird auch ausgeschaltet, wenn ein Unterbrechungssignal über den NMI-Eingang kommt. In diesem Fall wird aber vorher der Zustand des IFF1 auf das IFF2 übertragen.

IFF2 dient als Zwischenspeicher für das IFF1. Wenn ein Signal über den NMI-Eingang kommt, wird der Zustand von IFF1 und IFF2 gespeichert.

Ist das zu NMI gehörige Programm abgearbeitet, wird durch den Rückkehrbefehl für den nichtmaskierten INTERRUPT RETN der Inhalt von IFF2 wieder nach IFF1 gebracht.

Der Übergang in das zum INTERRUPT gehörende Arbeitsprogramm ist bei den Eingängen NMI und INT unterschiedlich.

- Bei dem nichtmaskierten INTERRUPT (NMI) geschieht das durch den Befehl CALL 66H.
  Dieser Befehl wird im Prozessor automatisch gebildet und abgearbeitet.
- Bei dem maskierten INTERRRUPT (INT) gibt es 2 Möglichkeiten des Übergangs in das dazugehörige Arbeitsprogramm. Die 3 Möglichkeiten werden mit MODE 0, MODE 1 und MODE 2 bezeichnet. Diese MODE lassen sich vorher durch die Befehle IMO, IM1 und IM2 einstellen.
- Im MODE 0 wird ein während des INTERRUPT-Zyklus eingelesenes 8-Bit-Wort als Befehl interpretiert und sofort abgearbeitet. Dafür verwendet man meistens den Befehl RSTZ. Er ist ein CALL-Befehl zu einer festen Adresse.
- Im MODE 1 wird ähnlich wie beim nichtmaskierten INTERRUPT ein Befehl CALL 38H gebildet und ausgeführt.

- Im MODE 2 wird aus dem I-Register als höherwertiger Teil und aus einem eingelesenen 8-Bit-Wort als niederwertiger Teil eine Adresse ADR gebildet. Das niederwertigste Bit des eingelesenen Bytes muß 0 sein. Das eingelesene 8-Bit-Wort nennt man Interrruptvektor.

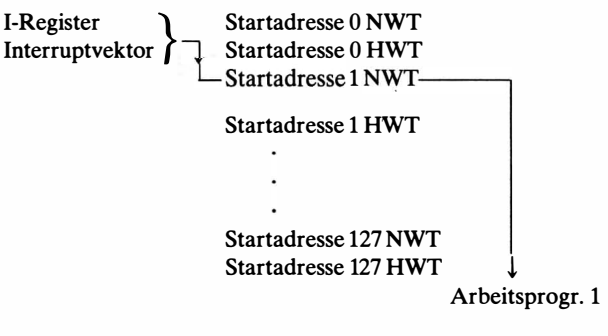
  ADR = | I-Register | Interruptvektor |

  Diese Adresse zeigt auf eine Zelle, deren Inhalt als Adresse eines CALL-Befehls verwendet wird. In MODE 2 wird also bei einem auftretenden INT-Signal der Befehl
  CALL ⟨I-Register, Interruptvektor⟩
  gebildet und ausgeführt.
  Bild 4.9 und 4.10 zeigen den zeitlichen Ablauf beim Erscheinen eines INTERRUPT-Signals.

Handhabung der INTERRUPT-Eingänge

- Der nichtmaskierte INTERRUPT wird für sehr wichtige Ereignisse verwendet. Er hat die höchste Priorität und unterbricht in jedem Fall das gerade laufende Programm. Zur Bedienung dieses INTERRUPT muß man ab Zelle 66 H ein dazugehöriges Arbeitsprogramm speichern.
- Der maskierte INTERRUPT-Eingang dient zum Aufbau umfangreicher Unterbrechungsschaltungen. An diese lassen sich. Sammelschaltungen für eine große Anzahl Unterbrechungsquellen anschließen. Die Unterbrechungsquellen können eine Mehrebenenstruktur haben und nach Prioritäten gestaffelt sein.

Zum Beispiel läßt sich MODE 2 im Speicher eine Tabelle der Startadresse aufbauen:



Der Inhalt des I-Registers gibt an, auf welcher Seite die Startadressen liegen (1 Seite = 256 Zellen). Durch das vom Datenbus kommende Byte lassen sich damit 128 INTERRUPT-Quellen bedienen.

4.1.7. Starten des Prozessors *U 880*

Die Programmabarbeitung des Prozessors läßt sich über die INTERRRUPT-Möglichkeiten oder über RESET starten. Dabei gibt es folgende Startvarianten:

1. Ein Startimpuls kommt über die NMI-Leitung. In diesem Fall muß ab Zelle 66H das Startprogramm stehen.
2. Der Startimpuls kommt über die INT-Leitung. Dazu müssen der INT-Eingang vorher mit EI freigemacht, der notwendige INTERRUPT MODE eingestellt und das INTERRUPT-Wort auf dem Datenbus bereitgestellt werden.
3. Nach RESET beginnt die Abarbeitung im Prozessor mit Zelle 0. In Zelle 0 kann damit der 1. Befehl des Startprogramms stehen.

4.1.8. Anschlüsse an den Baustein *U 880*

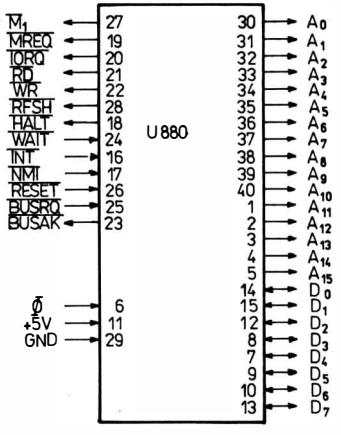
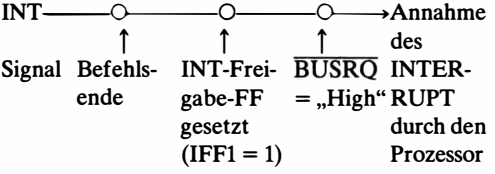


Bild 4.13 Anschlußbild des Bausteins *U 880*

A_0 bis A_{15} Adreß-Bus	Adreßbus, Tristate-Ausgang, aktiv „High"
D_0 bis D_7 Data-Bus	Datenbus, Tristate-Ein- und -Ausgang, aktiv „High"
M1	Der laufende Zyklus ist ein Operationscodehol-Zyklus Ausgangssignal, aktiv „Low"
$\overline{\text{MREQ}}$ Memory Request	Am Adreßbus ist eine Speicheradresse vorhanden. Tristate-Ausgang, aktiv „Low"
$\overline{\text{IORQ}}$ Input-Output Request	Der niederwertige Teil des Adreßbus enthält eine Ein- oder Ausgabeadresse. Ein gleichzeitiges Erscheinen von $\overline{M_1}$ und $\overline{\text{IORQ}}$ kennzeichnet einen INTERRUPT-Zyklus Tristate-Ausgang, aktiv „Low"
$\overline{\text{WR}}$ WRITE	Der Datenbus des Prozessors enthält Daten zur Ausgabe. Tristate-Ausgabe, aktiv „Low"
$\overline{\text{RD}}$ READ	Vom Prozessor werden Daten über den Datenbus eingelesen, Tristate-Ausgang, aktiv „Low"
$\overline{\text{RFSH}}$ Refresh	Die 7 niederwertigen Bits des Adreßbus enthalten eine Auffrischadresse für angeschlossene dynamische Speicher. Sie kann in Verbindung mit Memory Request zum Auffrischen dieses Speichers verwendet werden. Ausgabesignal, aktiv „Low"
$\overline{\text{HALT}}$ Halt state	Der Prozessor hat einen HALT-Befehl ausgeführt und befindet sich im HALT-Status. Während des HALT-Status führt der Prozessor NOP-Operationen aus, während dieser NOP-Operationen finden Auffrischzyklen statt. Ausgabesignal, aktiv „Low"
$\overline{\text{WAIT}}$ Wait	Das WAIT-Signal veranlaßt den Prozessor, nach dem nächsten T2-Zustand in den Wartezustand zu gehen; solange WAIT aktiv ist, führt der Prozessor Wartezyklen durch. Eingabesignal, aktiv „Low"

$\overline{\text{INT}}$
Interrupt Request

Anforderungssignal zu einer Programmunterbrechung im Prozessor. Es wird vom Prozessor am Ende eines Befehlszyklus angenommen, wenn das Flip-Flop IFF1 gesetzt (INTERRUPT-Erlaube-FF) und das Signal $\overline{\text{BUSRQ}}$ nicht aktiv ist.
Der Ablauf des INTERRUPT-Vorgangs ist im Abschnitt 4.1.6. beschrieben.
Eingabesignal, aktiv „Low"



$\overline{\text{NMI}}$
Non Maskable Interrupt

Das Signal NMI verhält sich ähnlich wie das Signal INT. Im Unterschied zu INT ist es jedoch nicht maskierbar und hat eine höhere Priorität. Es wird am Ende eines Befehls angenommen, wenn das Signal $\overline{\text{BUSRQ}}$ nicht aktiv ist.
Eingabe mit negativer Flanke getriggert.

$\overline{\text{RESET}}$

Dieses Signal setzt den Prozessor in den Grundzustand. Der Grundzustand ist gekennzeichnet durch:
$0 \rightarrow PC$ $0 \rightarrow I$
$0 \rightarrow IFF1;$ $0 \rightarrow R$
$0 \rightarrow IFF2;$ INTERRUPT MODE 0
Während das Signal RESET anliegt, sind Adreß- und Datenbus im hochohmigem Zustand und die Steuersignale inaktiv. Nach RESET beginnt die Abarbeitung mit Zelle 0.
Eingabesignal, aktiv „Low"

$\overline{\text{BUSRQ}}$
Bus Request

Dieses Signal bringt den Adreßbus, den Datenbus und die Steuersignale in den hochohmigen Zustand. Es wird am Ende jedes Maschinenzyklus abgefragt.
Eingabesignal, aktiv „Low"

$\overline{\text{BUSAK}}$
Bus

Datenbus, Adreßbus und Steuerbus befinden sich im hochohmigen Zustand. Die CPU arbeitet nicht.

Acknowledge	Es erfolgt kein REFRESH. Ausgabesignal, aktiv „Low“
Φ Steuertakt	Der Steuertakt kann über einen 330-Ω-Widerstand nach 5 V mit dem Ausgang eines TTL-Schaltkreises verbunden sein (Bild 4.14).

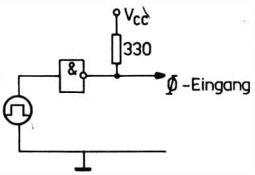


Bild 4.14
Takteingang in den Prozessor *U 880*

4.2. Der Mikroprozessorbaustein *8080*

Der Mikroprozessor *8080*, ein stark verbreiteter Baustein, wird in sehr vielen Mikrorechner-Konfigurationen verwendet. Alle in ihm vorhandenen Möglichkeiten sind auch im Mikroprozessor *U 880* enthalten.

4.2.1. Registerstruktur

Bild 4.15 zeigt die Registerstruktur des Bausteins *8080*. Die Register A, B, C, D, E, H, L, SP, PC, BR haben die gleiche Funktion wie beim Prozessor *U 880*. Das gleiche gilt für die Flags, C, S, Z, P, H. Das P-Flag wird jedoch nur in Abhängigkeit von der Parität des Ergebnisses gesetzt.

4.2.2. Befehlsaufbau

Ein *8080*-Befehl besteht wie beim *U 880* aus Operationscode und Adreßteil. Es gibt nur Befehle mit 1-Byte-Operationscode. Die Indexrechnung gibt es nicht.

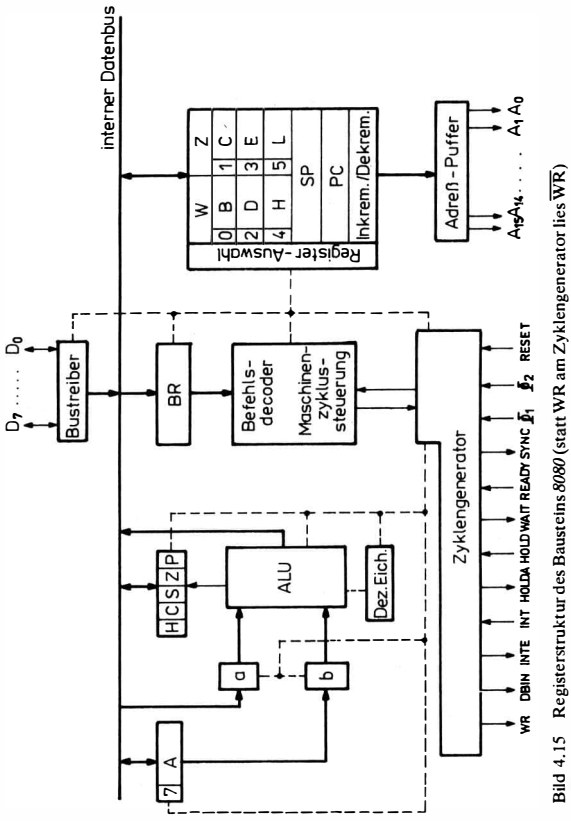


Bild 4.15 Registerstruktur des Bausteins *8080* (statt WR am Zyklengenerator lies $\overline{WR}$)

4.2.3. Zeitverhalten

Der Mikroprozessor *8080* wird durch 2 gegeneinander versetzte Impulsfolgen Φ_1 und Φ_2 gesteuert. Die Abarbeitung eines Befehls erfolgt in mehreren Maschinenzyklen. Ein Maschinenzyklus ist wiederum in 3 bis 5 Zeitzustände (T_1 bis T_5) unterteilt, wobei ein Zeitzustand die Länge von einer Periode Φ_1 hat.

Die wichtigsten Funktionen in den einzelnen Zeitzuständen innerhalb eines Maschinenzyklus sind (s. Taktdiagramm Bild 4.16)

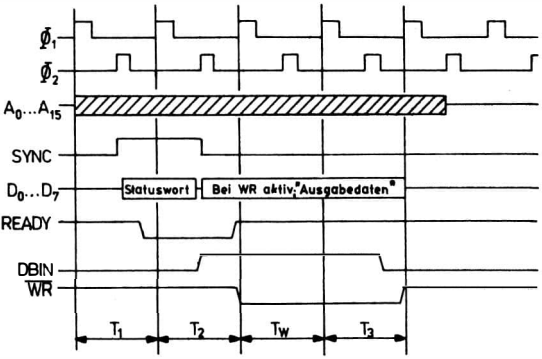


Bild 4.16 Taktdiagramm eines Maschinenzyklus des Prozessors *8080*

T_1 Ausgabe der Adresse auf den Adreßbus,
T_1/T_2 Ausgabe des Statusworts auf den Datenbus,
T_2 Abtasten des Signals „READY",
T_3 Ausgabe oder Eingabe über den Datenbus,
T_4/T_5 Ausführung des Befehls.

Nach T_2 können beliebig viele Wartezustände T_W eingefügt werden. Zu Beginn jedes Maschinenzyklus sendet der Prozessor über den Datenbus ein Statuswort aus. Das Signal SYNC zeigt an, daß das Statuswort auf dem Datenbus D_0 bis D_7 vorhanden ist. Ist zum Zeitpunkt T_2/Φ_2 READY = „Low", so kommt nach T_2 ein Wartezustand T_W. Ein weiterer Wartezustand folgt, wenn zum Zeitpunkt T_W/Φ_2 READY noch „Low" ist. Bei DBIN = „High" (aktiv) wird zum Zeitpunkt T_3/Φ_1 die Information vom Datenbus in den Prozessor geholt. Ist $\overline{WR}$ = „Low" (aktiv), so enthält der Datenbus eine vom Prozessor ausgegebene Information (DBIN und $\overline{WR}$ sind nie gleichzeitig aktiv; Bild 4.2 hat für diesen Fall nur symbolischen Charakter). Zur Ausführung eines Befehls werden mehrere Maschinenzyklen durchlaufen. Im 1. Maschinenzyklus M1 wird der Befehl geholt (Befehlsholzyklus), entschlüsselt und, wenn keine Daten von außerhalb des Prozessors benötigt werden, während T_4 und T_5 ausgeführt. Benötigt man Daten von außerhalb eines Prozessors, so folgen weitere Maschinenzyklen.

Beispiel
Eingabe eines Bytes nach dem A-Register.
Befehl: IN ADR Codierung im Speicher: 11011011 / ADR-Byte

Maschinenzyklus	Zustand	Funktion
M1	T_1	Ausgabe PC Ausgabe Status
	T_2	PC + 1 → PC Abfrage „READY"
	T_3	Befehl → BR
	T_4	interne Entschlüsselung
M2	T_1	Ausgabe PC Ausgabe Status
	T_2	PC + 1 → PC Abfrage „READY"
	T_3	ADR → Register Z und W
M3	T_1	Ausgabe Register Z und W (ist Geräteadresse)
	T_2	Abfrage READY
	T_3	Eingabe Byte vom DB → Register A

4.2.4. Anschlußsignale

D_0 bis D_7	Bidirektionaler Datenbus (aktiv „High"), wird für die Ein- und Ausgabe von Datensignalen verwendet.
A_0 bis A_{15}	Adreßbus (aktiv „High")
SYNC	Ausgabesignal (aktiv „High") sagt aus, daß der Datenbus das Statuswort enthält.
DBIN	Ausgabesignal (aktiv „High") sagt aus, daß zum Zustand T_3 eine Eingabe vom Datenbus erfolgt.
READY	Eingabesignal (aktiv „High") Ist zum Zeitpunkt T_2/Φ_2 oder T_W/Φ_2 READY = „High", so folgt der Zustand T_3, bei READY = „Low" folgt ein Wartezustand T_W.
WAIT	Dieses Ausgabesignal (aktiv „High") sagt aus, daß sich der Prozessor im Wartezustand T_W befindet.
$\overline{WR}$	Das Ausgabesignal (aktiv „Low") sagt aus, daß sich auf dem Datenbus ein ausgegebenes Byte befindet.

HOLD Eingabesignal (aktiv „High")
DMA-Anforderungssignal. Durch HOLD geht der Daten- und Adreßbus in den hochohmigen Zustand (HOLD-Zustand). Das Signal HOLD wird angenommen:
- im Zustand T_2 oder T_W;
- im HALT-Zustand.

Der HOLD-Zustand bleibt so lange bestehen, wie HOLD = „High" ist. Nachdem HOLD = „Low" wird, beginnt ein neuer Maschinenzyklus mit T_1.

HLDA Ausgabesignal (aktiv „High") sagt aus, daß sich der Prozessor im HOLD-Zustand befindet, d. h. Daten- und Adreßbus hochohmig sind.

INTE Ausgabesignal (aktiv „High"); sagt aus, daß der INT-Eingang frei ist. Es wird rückgesetzt durch den Befehl DI oder bei Annahme eines INT-Signals.

INT Eingabesignal (aktiv „High")
Programmunterbrechungsanforderung
INT wird angenommen:
- am Ende eines Befehls;
- im HALT-Zustand;
- bei INTE = „High".

Die Programmunterbrechung entspricht dem INTERRUPT-MODE 0 des Bausteins *U 880*. Bei Annahme der Programmunterbrechung wird ein INT-Zyklus durchlaufen. Im INT-Zyklus wird
- im Statuswort das Signal INTA ausgesandt;
- im Zustand T_3 ein 1-Byte-Befehl vom Datenbus eingelesen und sofort ausgeführt. Dazu verwendet man in der Regel den Befehl RST.

RESET Eingabesignal (aktiv „High")
Durch RESET wird der Baustein *8080* in den Grundzustand versetzt. Der Grundzustand ist gekennzeichnet durch:
- PC gelöscht;
- BR gelöscht;
- INTE = „Low";
- HLTA = „Low";

Die Register A, B, C, D, E, H, L bleiben erhalten.

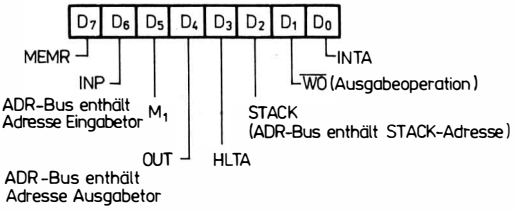


Bild 4.17 Statuswort des Bausteins *8080*

Statusinformationen

Zusätzlich zu den Steuersignalen, die direkt vom Prozessor *8080* ausgehen, gibt der Baustein *8080* am Anfang jedes Maschinenzyklus gleichzeitig mit dem Signal SYNC ein Statuswort über den Datenbus aus. Bild 4.17 zeigt den Aufbau des Statuswortes.

4.2.5. Anschluß von Schaltkreisen an den Prozessor *8080*

Mit Hilfe der Status- und Prozessor-Ein- und -Ausgabesignale werden die am Prozessor angeschlossenen Bausteine gesteuert. Bild 4.18 zeigt die Steuerung des Bausteins *8080* und die Bildung von Adreßbus, Datenbus und Steuerbus zur Realisierung von Mikrorechneraufbauten.

Darin bedeuten (s. Taktdiagramm Bild 4.19):

A_0 bis A_{15} Adreßbus
D_0 bis D_7 Datenbus
DBIN, WR *8080*-Signale

INA	DBIN · INTA	Eingabesignal für INTERRUPT-Befehl
IOR	= DBIN · INP	Eingabesignal für Eingabekanal
IOW	= WR · OUT	Ausgabesignal für Ausgangskanal
MER	= DBIN · MEMR	Speicherlesesignal
STR	= Φ_{1A} · SYNC	Statusübernahmesignal Φ_{1A} wird aus Φ_2 gebildet (etwas verzögertes Φ_2)

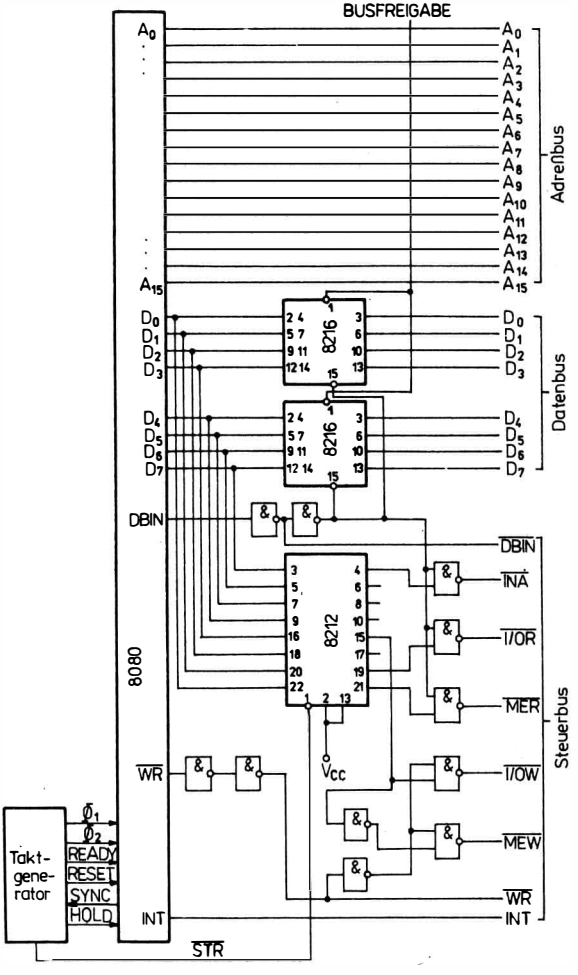


Bild 4.18 Bildung des Mikrorechnerbus mit dem Baustein *8080*

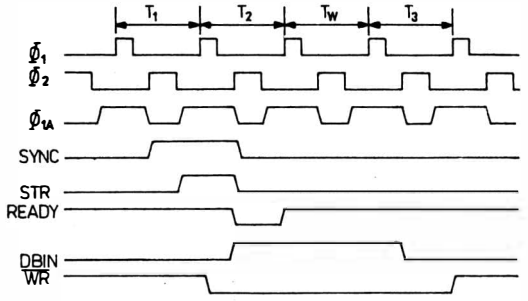


Bild 4.19 Taktdiagramm zur Bildung des Mikrorechnerbus mit dem Baustein *8080*

4.2.6. Befehlsschlüssel des Prozessors *8080*

Alle Befehle des *8080* sind auch im *U 880* enthalten. Aus den Tabellen im Anhang ist zu ersehen, welche Befehle des *U 880* im *8080* vorkommen. Es sind auch die Befehle des *U 808 D* aufgeführt, die sowohl im *8080* als auch im *U 880* enthalten sind. Die Befehle des *U 808 D* haben jedoch gegenüber dem *U 880* einen anderen Befehlscode.

4.2.7. Programmunterbrechung

Der Baustein *8080* hat einen Programmunterbrechungseingang. Der Ablauf der Programmunterbrechung ist identisch mit dem Ablauf der Programmunterbrechung im Prozessor *U 880* im MODE 0 (s. Abschnitt 4.1.6.).
Der einzige Unterschied besteht nach dem Einschalten der Spannungen bzw. nach RESET. Im Baustein *U 880* ist nach Einschalten der Spannungen und nach RESET der maskierte Eingang gesperrt.
Ein INTERRUPT kann nur über den nichtmaskierten Eingang kommen. Nach RESET startet der Prozessor *U 880* automatisch bei Zelle 0. Im Baustein *8080* ist nach Einschalten der Spannungen und nach RESET der INTERRUPT-Eingang offen. Der Prozessor *8080* wird über INTERRUPT gestartet.

Anhang

Befehlstabellen der Prozessoren *U 880* und *8080*

Abkürzungen

n	8-Bit-Zahl,
nn	16-Bit-Zahl,
N	Register A, B, C, D, E, H, L,
M	Speicherzelle ADR = ⟨HL⟩, ⟨IX + d,⟩, ⟨IY + d⟩.

Befehlsgruppen (Übersicht)

0 Adreßoperationen (steht in Verbindung mit Befehlen, die zum Speicher zugreifen)

1 Transportoperationen
- 1 Wort
- Doppelwort
- Blocktransfer

2 Rechen- und logische Operationen mit 1 Operand
- Akkumulatorbefehle
- 1-Wort-Befehle
- Doppelwortbefehle

  → logische Operationen, Verschiebeoperationen, Bit-Operation, Rechenoperationen

3 Rechen- und logische Operationen mit 2 Operanden
- 1-Wort-Befehle
- Doppelwortbefehle

4 Rechen- und logische Operationen mit Feldern

5 Sprungbefehle

6 Unterprogramm-Befehle
- Aufruf des Unterprogramms
- Rückkehr zum Hauptprogramm

7 Ein-/Ausgabebefehle
- 1-Wort-Befehle
- Blocktransfer

8 Steuerbefehle

Tabelle der U 880-Befehle

Einzelworttransfer

LD r, s s → r
LD d, r r → d
LD A, s s → A
LD d, A A → d

		C Z P/V S N H	
s = n, N, M[1]	r = N		Bei allen
r = n, N	d = N, M	. . . . .	Transportbef.
s = ⟨BC⟩, ⟨DE⟩, ⟨nn⟩, I, R		↕ IFF ↕ 0 0	LD A, I
d = ⟨BC⟩, ⟨DE⟩, ⟨nn⟩, I, R			LD A, R

Doppelworttransfer

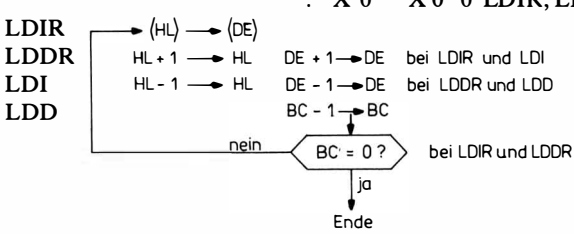
			C Z P/V S N H
LD dd, nn	nn → dd	dd = BC, DE, HL,	
LD dd, ⟨nn⟩	⟨nn⟩ → dd	SP, IX, IY	. . . . .
LD ⟨nn⟩, ss	ss → ⟨nn⟩,	ss = BC, DE, HL, SP, IX, IY	
LD SP, ss	ss → SP	ss = HL, IX, IY	
PUSH ss	ss → STACK	ss = BC, DE, HL, AF, IX, IY	
POP ss	STACK → ss		

Registeraustausch

			C Z P/V S N H
EX DE,	HL	DE ↔ HL	
EX AF,	AF'[2]	AF ↔ A'F'	. . . . .
EXX		BC ↔ B'C' DE ↔ D'E' HK ↔ H'L'	
EX ⟨SP⟩, ss		ss ↔ ⟨SP + 1⟩, ⟨SP⟩ ss = HL, IX, IY	

Blocktransfer

		C Z P/V S N H	
		. X ↕ X 0 0	LDI, LDD
		. X 0 X 0 0	LDIR, LDDR



Rechenoperationen und logische Operationen mit 1 Operand

		C Z P/V S N H
CPL	$\overline{A} \rightarrow A$	. . . . 1 1
NEG	$\overline{A} + 1 \rightarrow A$	↕ ↕ v ↕ 1 ↕

[1] N = Register A, B, C, D, E, H, L
M = Speicherzelle ADR = ⟨HL⟩, ⟨IX + d⟩, ⟨IY + d⟩

[2] In der Assemblersprache *K 1520* wird statt EX AF, AF' EXAF geschrieben.

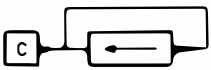
			C	Z	P/V	S	N	H
CCF	$\bar{C} \rightarrow C$		↕	.	.	.	0	X
SCF	$1 \rightarrow C$		1	.	.	.	0	0
DAA BCD	$\langle A \rangle \rightarrow A$		↕	↕	P	↕	.	↕
DEC d	$d - 1 \rightarrow d$	d = N, M	.	↕	v	↕	1	↕
INC d	$d + 1 \rightarrow d$		.	↕	v	↕	0	↕
BIT b, s	$\bar{s}_b \rightarrow Z$		.	↕	X	X	0	1
SET b, s	$1 \rightarrow s_b$	s = N, M	.	.	.	.	.	.
RES b, s	$0 \rightarrow s_b$		.	.	.	.	.	.
INC dd	$dd + 1 \rightarrow dd$	dd = BC, DE, HL, SP, IX, IY						
DEC dd	$dd - 1 \rightarrow dd$							

Verschiebebefehle

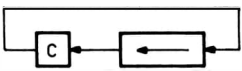
RLCA, RLA, RRCA, RRA

C	Z	P/V	S	N	H
↕	.	.	.	0	0

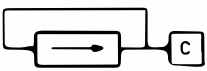
RCL s
RLCA



RL s
RLA



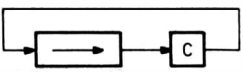
RRC s
RRCA



RLCs, RLs, RRCs, RRs, SLAs, SRAs, SRL s

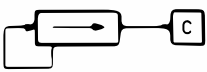
C	Z	P/V	S	N	H
↕	↕	P	↕	0	0

RR s
RRA



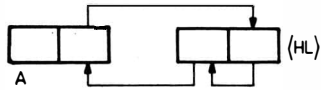
SLA s

SRA s

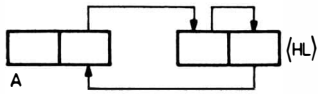


SRL s

RLD



RRD



RLD, RRD

C	Z	P/V	S	N	H
.	↕	P	↕	0	0

Rechenoperationen mit 2 Operanden

8-Bit-Operationen

ADD s	A + s → A	s = N, n, M[1]
ADC s	A + s + C → A	
SUB s	A − s → A	
SBC s	A − s − C → A	
AND s	A ∧ s → A	
OR s	A ∨ s → A	
XOR s	A ⊕ s → A	
CP s	Vergleich A, s	A − s stellt Flags

ADD, ADC

C	Z	P/V	S	N	H
↕	↕	v	↕	0	↕

SUB, SBC, CP

C	Z	P/V	S	N	H
↕	↕	v	↕	1	↕

bei AND

C	Z	P/V	S	N	H
0	↕	P	↕	0	1

0 – bei OR, XOR

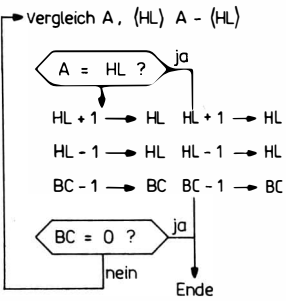
16-Bit-Operationen

ADD HL, ss	HL + ss → HL	ss = BC, DE, HL, SP
ADC HL, ss	HL + ss + C → HL	
SBC HL, ss	HL − ss − C → HL	
ADD IX, ss	IX + ss → IX	ss = BC, DE, IX, SP
ADD IY, ss	IY + ss → IY	ss = BC, DE, IY, SP

C	Z	P/V	S	N	H
↕	.	.	.	0	X
↕	↕	v	↕	0	↕
↕	↕	v	↕	1	↕
↕	.	.	.	.	↕
↕	.	.	.	.	↕

[1]) N = Register A, B, C, D, E, H, L
M = Speicherzelle ADR = ⟨HL⟩, ⟨IX + d⟩, ⟩IY + d⟩

Rechenoperationen auf Feldern C Z P/V S N H
Suchoperationen 0 ↕ ↕ X 1 X



CPIR
CPDR
CPI
CPD

A − ⟨HL⟩ stellt Flags

bei CPIR und CPI
bei CPDR und CPD

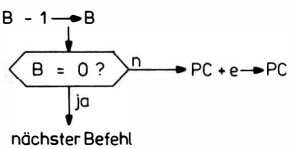
bei CPIR und CPDR

Sprungbefehle

JP nn	nn → PC	
JP cc, nn[1])	nn → PC, wenn cc erfüllt cc = NZ, Z, NC, C, PO, PE, P, M	
JR e	PC + e → PC	
JR kk, e[2])	PC + e → PC, wenn kk erfüllt kk = NZ, Z, NC, C	
JP ⟨ss⟩	ss → PC ss = HL, IX, IY	

C Z P/V S N H
. . . . . .

DJNZ e



Unterprogrammbefehle C Z P/V S N H
. . . . . .

CALL nn	PC → STACK nn → PC
CALL cc, nn[3])	CALL nn, wenn erfüllt cc = NZ, Z, NC, C, PO PE, P, M

[1]) In der Assemblersprache K 1520 wird statt JP nn JPM nn, JP (ss) JMP (ss) und JP cc, nn JPcc nn geschrieben.
[2]) In der Assemblersprache K 1520 wird statt JR kk, nn JRkk nn geschrieben.
[3]) In der Assemblersprache K 1520 wird statt CALL cc, nn CAcc nn geschrieben.

RST Z	CALL Z mit Z = OH, 8H, 10H, 18H, . . . , 38H
RET	STACK → PC
RET cc[4])	STACK → PC, wenn cc erfüllt
RETN	Rücksprung aus nichtmaskiertem Interrupt
RETI	Rücksprung aus maskiertem Interrupt

Ein-/Ausgabebefehle

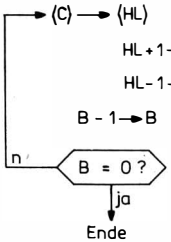
	Adreßbus $A_{15}...A_8$	$A_7...A_0$
IN A, ⟨n⟩[6]) ⟨n⟩ → A	A	n
IN r, ⟨C⟩ ⟨C⟩ → r r = N[5])	B	C
OUT ⟨n⟩, A A → ⟨n⟩	A	n
OUT ⟨C⟩, r r → ⟨C⟩ r = N[5])	B	C

C Z P/V S N H

. . . . . bei Adresse n

. ↕ P ↕ 0 0 bei Adresse C

Blockeingabe



Adreßbus

$A_{15}...A_8$ $A_7...A_0$

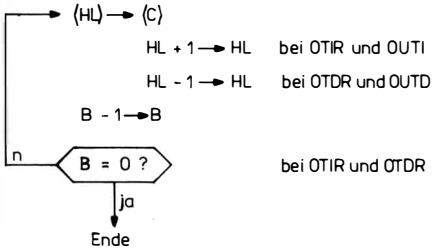
B | C

INI, IND, OUTI, OUTD

C Z P/V S N H

. ↕ X X 1 X

Blockausgabe



INIR, INDR, OTIR, OTDR

C Z P/V S N H

. 1 X X 1 X

[4]) In der Assemblersprache K 1520 wird statt RET cc Rcc geschrieben, cc und kk wie oben.

[5]) N ist Register A, B, C, D, E, H, L.

[6]) In der Assemblersprache K 1520 lauten die 4 E/A-Befehle IN n, IN r, OUT n, OUT r.

Steuerbefehle

			C	Z	P/V	S	N	H
NOP	keine Operation							
HALT	Haltbefehl							
DI	$0 \rightarrow$ IFF	INTERRUPT gesperrt	.	.	.	.	.	.
EI	$1 \rightarrow$ IFF	INTERRUPT frei						
IM0	MOD 0	Befehl vom Bus						
IM1	MOD 1	CALL 38H						
IM2	MOD 2	CALL $\langle$I, IV$\rangle$						

Tabelle der 8080-Befehle

Einzelworttransfer

				C	Z	P	S	H
MOV	r, s	$s \rightarrow r$	s, r = N, M	.	.	.	.	.
MVI	r, n	$n \rightarrow r$	[N ist Register A, B, C, C, B, H, L, M ist Speicherzelle, ADR = (HL)]					
STAX	ss	$A \rightarrow \langle ss \rangle$	ss = BC, DE					
LDAX	ss	$\langle ss \rangle \rightarrow A$	ss = BC, DE					
STA	nn	$A \rightarrow \langle nn \rangle$						
LDA	nn	$\langle nn \rangle \rightarrow A$						
SPHL		$HL \rightarrow SP$						

Doppelworttransfer

			C	Z	P	S	H
LXI	dd, nn	nn $\rightarrow$ dd dd = BC, DE, HL	.	.	.	.	.
SHLD	nn	$HL \rightarrow \langle nn \rangle$					
LHLD	nn	$\langle nn \rangle \rightarrow HL$					
PUSH	ss	ss $\rightarrow$ STACK ss = BC, DE, HL, AF					
POP	ss	STACK $\rightarrow$ ss ss = BC, DE, HL, AF					

Registeraustausch

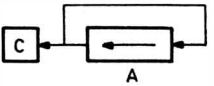
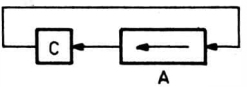
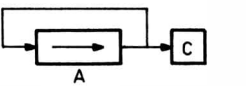
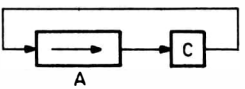
XCHG	$HL \leftrightarrow DE$	
XTHL	$HL \leftrightarrow \langle SP+1 \rangle, \langle SP \rangle$	

Rechenoperationen und logische Operationen mit 1 Operand

			C	Z	P	S	H
CMC	$\overline{C} \rightarrow C$		↕	.	.	.	X
STC	$1 \rightarrow C$		1	.	.	.	0
CMA	$\overline{A} \rightarrow A$		.	.	.	.	1
DAA	BCD $\langle A \rangle \rightarrow A$		↕	↕	↕	↕	↕
INR d	$d+1 \rightarrow d$	d = N, M	.	↕	↕	↕	↕
DCR d	$d-1 \rightarrow d$	d = N, M	.	↕	↕	↕	↕

INX dd	dd + 1 → dd	dd = BC, DE, HL, SP	C Z P S H
DCX dd	dd − 1 → dd		. . . . .

Verschiebebefehle

RLC		C Z P S H ↕ . . . 0
RAL		
RRC		
RAR		

Rechenoperationen mit 2 Operanden

8-Bit-Operationen				C Z P S H
ADD	r	A + r → A	r = N, M	↕ ↕ ↕ ↕ ↕
ADC	r	A + r + C → A		
SUB	r	A − r → A		
SBB	r	A − r − C → A		
ANA	r	A ∧ r → A		
ORA	r	A ∨ r → A		
XRA	r	A ⊕ r → A		
CMP	r	Vergleich A, r A − r setzt die Flags		
ADI	n	A + n → A		C Z P S H
ACI	n	A + n + C → A		↕ ↕ ↕ ↕ ↕
SUI	n	A − n → A		
SBI	n	A − n − C → A		
ANI	n	A ∧ n → A		
ORI	n	A ∨ n → A		
XRI	n	A ⊕ n → A		
CPI	n	Vergleich A, n A − n setzt die Flags		

16-Bit-Operationen C Z P S H

DAD ss HL + ss → HL ss = BC, DE, HL, SP ↕ . . . X

Sprungbefehle C Z P S H

PCHL HL → PC . . . . .

JMP nn nn → PC

Jcc nn nn → PC, wenn cc erfüllt cc = NZ, Z, NC, C, PO, PE, P, M

Unterprogrammbefehle C Z P S H

. . . . .

CALL nn PC → STACK, nn → PC

Ccc nn CALL nn, wenn cc erfüllt, sonst Leerbefehl
cc = NZ, Z, NC, C, PO, PE, P, M

RST Z wie CALL Z Z = OH, 8H, 10H, . . ., 38H

RET STACK → PC

Rcc RET, wenn cc erfüllt, sonst Leerbefehl
cc = NZ, Z, NC, C, PO, PE, P, M

Ein-/Ausgabebefehle C Z P S H

. . . . .

			Adreßbus $A_{15} \ldots A_8$	$A_7 \ldots A_0$
IN	n	⟨n⟩ → A	A	n
OUT	n	A → ⟨n⟩	A	n

Steuerbefehle C Z P S H

. . . . .

EI INTERRUPT frei

DI INTERRUPT gesperrt

HLT HALT

NOP keine Operation

Tabelle der U 808 D-Befehle

Einzelworttransfer C S Z P

. . . .

MOV r, ss → r
MVI r, nn → r } s, r = N, M

Rechenoperationen und logische Operationen mit 1 Operand

C S Z P

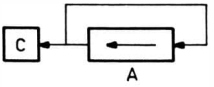
INR d d + 1 → d . ↕ ↕ ↕

DCR d d → 1 → d . ↕ ↕ ↕

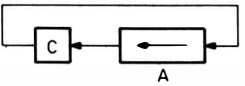
Verschiebebefehle

C S Z P
↕ . . .

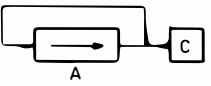
RLC



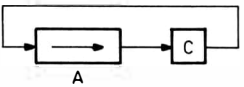
RAL



RRC



RAR



Rechenoperationen mit 2 Operanden

8-Bit-Operationen

C Z P S
↕ ↕ ↕ ↕

ADD r	A + r → A	r = N, M
ADC r	A + r + C → A	
SUB r	A − r → A	
SBB r	A − r − C → A	
ANA r	A ∧ r → A	
ORA r	A ∨ r → A	
XRA r	A ⊕ r → A	
CMP r	Vergleich A, r A − r setzt die Flags	

C Z P S
↕ ↕ ↕ ↕

ADI n	A + n → A
ACI n	A + n + C → A
SUI n	A − n → A
SBI n	A − n − C → A
ANI n	A ∧ n → A
ORI n	A ∨ n → A
XRI n	A ⊕ n → A
CPI n	Vergleich A, n A − n setzt die Flags

Sprungbefehle C Z P S

JMP nn nn → PC . . . .

Jcc nn nn → PC, wenn cc erfüllt, sonst Leerbefehl
cc = NZ, Z, NC, C, PO, PE, P, M

Unterprogrammbefehle C Z P S

. . . .

CALLnn PC → STACK nn → PC

Ccc nn CALL nn, wenn cc erfüllt, sonst Leerbefehl
cc = NZ, Z, NC, C, PO, PE, PM

RST Z Wie CALL Z Z = 0H, 8H, 10H, . . ., 38H

RET STACK → PC

Rcc RET, wenn cc erfüllt, sonst Leerbefehl
cc = NZ, Z, NC, C, PO, PE, P, M

Ein-/Ausgabebefehle C Z P S

. . . .

IN n ⟨n⟩ → A n = 0 bis 7 (Geräteadresse)

OUT n A → ⟨n⟩ n = 8 bis 31 (Geräteadresse)

Steuerbefehle C Z P S

HLT HALT . . . .

NOP keine Operation

Codierungstabelle U 880

1. Byte Opcode	U 880	Opcode	U 880
00	NOP	10	DJNZ e
01	LD BC,nn	11	LD DE,nn
02	LD (BC),A	12	LD (DE),A
03	INC BC	13	INC DE
04	INC B	14	INC D
05	DEC B	15	DEC D
06	LD B,n	16	LD D,n
07	RLCA	17	RLA
08	EXAF	18	JR e
09	ADD HL,BC	19	ADD HL,DE
0A	LD A,(BC)	1A	LD A,(DE)
0B	DEC BC	1B	DEC DE
0C	INC C	1C	INC E
0D	DEC C	1D	DEC E
0E	LD C,n	1E	LD E,n
0F	RRCA	1F	RPA

Opcode	U 880	Opcode	U 880
20	JRNZ e	30	JRNC e
21	LD HL,nn	31	LD SP,nn
22	LD (adr),BC	32	LD (adr),A
23	INC HL	33	INC SP
24	INC H	34	INC M
25	DEC H	35	DEC M
26	LD H,n	36	LD M,n
27	DDA	37	SCF
28	JRZ e	38	JRC e
29	ADD HL,HL	39	ADD HL,SP
2A	LD HL,(adr)	3A	LD A,(adr)
2B	DEC HL	3B	DEC SP
2C	INC L	3C	INC A
2D	DEC L	3D	DEC A
2E	LD L,n	3E	LD A,n
2F	CPL	3F	CCF
40	LD B,B	50	LD D,B
41	LD B,C	51	LD D,C
42	LD B,D	52	LD D,D
43	LD B,E	53	LD D,E
44	LD B,H	54	LD D,H
45	LD B,L	55	LD D,L
46	LD B,M	56	LD D,M
47	LD B,A	57	LD D,A
48	LD C,B	58	LD D,B
49	LD C,C	59	LD D,C
4A	LD C,D	5A	LD D,D
4B	LD C,E	5B	LD D,E
4C	LD C,H	5C	LD D,H
4D	LD C,L	5D	LD D,L
4E	LD C,M	5E	LD D,M
4F	LD C,A	5F	LD D,A
60	LD H,B	70	LD M,B
61	LD H,C	71	LD M,C
62	LD H,D	72	LD M,D
63	LD H,E	73	LD M,E
64	LD H,H	74	LD M,H
65	LD H,L	75	LD M,L
66	LD H,M	76	HALT
67	LD H,A	77	LD M,A
68	LD L,B	78	LD A,B
69	LD L,C	79	LD A,C
6A	LD L,D	7A	LD A,D
6B	LD L,E	7B	LD A,E
6C	LD L,H	7C	LD A,H
6D	LD L,L	7D	LD A,L
6E	LD L,M	7E	LD A,M
6F	LD L,A	7F	LD A,A

Opcode	U 880	Opcode	U 880
80	ADD B	90	SUB B
81	ADD C	91	SUB C
82	ADD D	92	SUB C
83	ADD E	93	SUB E
84	ADD H	94	SUB H
85	ADD L	95	SUB L
86	ADD M	96	SUB M
87	ADD A	97	SUB A
88	ADC B	98	SBC B
89	ADC C	99	SBC C
8A	ADC D	9A	SBC D
8B	ADC E	9B	SBC E
8C	ADC H	9C	SBC H
8D	ADC L	9D	SBC L
8E	ADC M	9E	SBC M
8F	ADC A	9F	SBC A
A0	AND B	B0	OR B
A1	AND C	B1	OR C
A2	AND C	B2	OR D
A3	AND E	B3	OR E
A4	AND H	B4	OR H
A5	AND L	B5	OR L
A6	AND M	B6	OR M
A7	AND A	B7	OR A
A8	XOR B	B8	CMP B
A9	XOR C	B9	CMP C
AA	XOR D	BA	CMP D
AB	XOR E	BB	CMP E
AC	XOR H	BC	CMP H
AD	XOR L	BD	CMP L
AE	XOR M	BE	CMP M
AF	XOR A	BF	CMP A
C0	RNZ	D0	REC
C1	POP BC	D1	POP DE
C2	JPNZ nn	D2	JPNC nn
C3	JMP nn	D3	OUT n
C4	CANZ nn	D4	CANC nn
C5	PUSH BC	D5	PUSH DE
C6	ADD n	D6	SUB n
C7	RST OH	D7	RST 1OH
C8	RZ	D8	RC
C9	RET	D9	EXX
CA	JPZ nn	DA	JPC nn
CB	Verschiebe- und Bitbefehle	DB	IN n
CC	CAZ nn	DC	CAC nn

Opcode	U 880	Opcode	U 880
CD	CALL adr	DD	IX-Befehle
CE	ADC n	DE	SBC n
CF	RST 8H	DF	RST 18H
E0	RPO	F0	RP
E1	POP HL	F1	POP AF
E2	JPPO nn	F2	JPP nn
E3	EX (SP),HL	F3	DI
E4	CAPO nn	F4	CAP nn
E5	PUSH HL	F5	PUSH AF
E6	AND n	F6	OR n
E7	RST POH	F7	RST 3OH
E8	RPE	F8	RM
E9	JMP M	F9	LD SP,HL
EA	JPPE nn	FA	JPM nn
EB	EX DE,HL	FB	EI
EC	CAPE nn	FC	CAM nn
ED	Sondertransport-befehle	FD	IY-Befehle
EE	XOR n	FE	CMP n
EF	RST 28H	FF	RST 38H

Verschiebe- und Bitbefehle

1. Byte CB

2. Byte 7 6 5 4 3 2 1 0

		REG	
0 0	RLC	0	B
0 1	RRC	1	C
0 2	RL	2	D
0 3	RR	3	E
0 4	SLA	4	H
0 5	SRA	5	L
0 6	–	6	M
0 7	SRL	7	A
1 0	BIT 0,r		
1 1	BIT 1,r		
1 2	BIT 2,r		
1 3	BIT 3,r		
1 4	BIT 4,r		
1 5	BIT 5,r		
1 6	BIT 6,r		
1 7	BIT 7,r		
2 0	RES 0,r		
2 1	RES 1,r		

```
2 2 RES 2,r
2 3 RES 3,r
2 4 RES 4,r
2 5 RES 5,r
2 6 RES 6,r
2 7 RES 7,r
3 0 SET 0,r
3 1 SET 1,r
3 2 SET 2,r
3 3 SET 3,r
3 4 SET 4,r
3 5 SET 5,r
3 6 SET 6,r
3 7 SET 7,r
```

Indexbefehle mit IX
(bei Indexbefehlen mit IY ist das 1. Byte FD)

DD09	ADD	IX, BC	DD86	ADD	A, (IX+offset)
DD19	ADD	IX, DE	DD8E	ADC	A, (IX+offset)
DD21	LD	IX, dddd	DD96	SUB	(IX+offset)
DD22	LD	(adr), IX	DD9E	SBC	A, (IX+offset)
DD23	INC	IX	DDA6	AND	(IX+offset)
DD29	ADD		DDAE	XOR	(IX+offset)
DD2A	LD	IX, (adr)	DDB6	OR	(IX+offset)
DD2B	DEC	IX	DDBE	CP	(IX+offset)
DD34	INC	(IX+offset)	DDCB of 06	RLC	(IX+offset)
DD35	DEC	(IX+offset)	DDCBof0E	RRC	(IX+offset)
DD36	LD	(IX+offset), dd	DDCB of 16	RL	(IX+offset)
DD39	ADD	IX, SP	DDCB of 1E	RR	(IX+offset)
DD46	LD	B, (IX+offset)	DDCB of 26	SIA	(IX+offset)
DD4E	LD	C, (IX+offset)	DDCB of 2E	SRA	(IX+offset)
DD56	LD	D, (IX+offset)	DDCB of 3E	SRL	(IX+offset)
DD5E	LD	E, (IX+offset)	DDCBof 46	BIT	0, (IX+offset)
DD66	LD	H, (IX+offset)	DDCBof 4E	BIT	1, (IX+offset)
DD6E	LD	L, (IX+offset)	DDCB of 56	BIT	2, (IX+offset)
DD70	LD	(IX+offset), B	DDCB of 5E	BIT	3, (IX+offset)
DD71	LD	(IX+offset), C	DDCB of 66	BIT	4, (IX+offset)
DD72	LD	(IX+offset), D	DDCB of 6E	BIT	5, (IX+offset)
DD73	LD	(IX+offset), E	DDCB of 76	BIT	6, (IX+offset)
DD74	LD	(IX+offset), H	DDCB of 7E	BIT	7, (IX+offset)
DD75	LD	(IX+offset), L			
DD77	LD	(IX+offset), A			
DD7E	LD	A, (IX+offset)			
DCBof86	RES	0, (IX+offset)			
DDCB of 8E	RES	1, (IX+offset)			
DDCB of 96	RES	2, (IX+offset)			
DDCB of 9E	RES	3, (IX+offset)			
DDCB of A6	RES	4, (IX+offset)			

DDCB of AE	RES	5, (IX+offset)
DDCB of B6	RES	6, (IX+offset)
DDCB of BE	RES	7, (IX+offset)
DDCB of C6	SET	0, (IX+offset)
DDCB of CE	SET	1, (IX+offset)
DDCB of D6	SET	2, (IX+offset)
DDCB of DE	SET	3, (IX+offset)
DDCB of E6	SET	4, (IX+offset)
DDCB of EE	SET	5, (IX+offset)
DDCB of F6	SET	6, (IX+offset)
DDCB of FE	SET	7, (IX+offset)
DDE1	POP	IX
DDE3	EX	(SP), IX
DDE5	PUSH	IX
DDE9	JP	(IX)
DDF9	LD	SP, IX

Sondertransportbefehle

ED40	IN	B	ED68	IN	L
ED41	OUT	B	ED69	OUT	L
ED42	SBO	HL, BC	ED6A	ADC	HL, HL
ED43	LD	(adr), BC	ED6F	RLD	
ED44	NEG		ED72	SBC	HL, SP
ED45	RETN		ED73	LD	(adr), SP
ED46	IM	0	ED78	IN	A
ED47	LD	I, A	ED79	OUT	A
ED48	IN	C	ED7A	ADC	HL, SP
ED49	OUT	C	ED7B	LD	SP, (adr.)
ED4A	ADC	HL, BC	EDAO	LDI	
ED4B	LD	BC, (adr)	EDA1	CPI	
ED4D	RETI		EDA2	INI	
ED50	IN	D	EDA3	OUTI	
ED51	OUT	D	EDA8	LDD	
ED52	SBC	HL, DE	EDA9	CPD	
ED53	LD	(adr), DE	EDAA	IND	
ED56	IM	1	EDAB	OUTD	
ED57	LD	A, I	EDBO	LDIR	
ED58	IN	E	EDB1	OPIR	
ED59	OUT	E	EDB2	INIR	
ED5A	ADC	HL, DE	EDB3	OTIR	
ED5B	LD	DE, (adr)	EDB6	LDDR	
ED5E	IM	2	EDB9	CPDR	
ED60	IN	H	EDBA	INDR	
ED61	OUT	H	EDBB	OTDR	
ED62	SBC	HL, HL			
ED67	RRD				