4. Mikrorechner

4.1 Übersicht

Mit dem im vorletzten Kapitel vorgestellten Grundprinzip eines Steuerautomaten kann die Grundstruktur des programmgesteuerten Universalrechners entwickelt werden. Das grundlegende Organisationsprinzip ist das von Neumann'sche Konzept. Vom Gesichtspunkt der Rechnerorganisation her gesehen, kann die von Neumann-Struktur als eine Struktur minimalen Hardware-Aufwands bezeichnet werden. Sie besteht aus den folgenden Einheiten:

a) dem Speicher, in dem das auszuführende Maschinenprogramm und seine Rechendaten gespeichert sind;

b) der zentralen Recheneinheit (CPU), in der die Befehle eines Programms interpretiert und ausgeführt werden;

c) der Ein-/Ausgabe-Einheit, welche die Verbindung mit den peripheren Ein-/Ausgabegeräten und die Kommunikation zwischen dem Rechner und seiner Umwelt besorgt;

d) dem internen Datenbus, der den Informationsaustausch zwischen den Komponenten ermöglicht (allgem. Verbindungen, z.B. Punkt zu Punkt oder Bus).

In Bild 4-2 ist das Blockdiagramm eines Rechner nach dem von Neumann´schen Prinzip angegeben. Die zentrale Recheneinheit besteht aus dem Befehls- und Rechenwerk.

Das Befehlswerk organisiert den Speicherzugang zu Befehlen und Daten, entschlüsselt die Befehle und löst die Fortschaltung zum nächsten Befehl und die einzelnen Operationen aus. Das Rechenwerk ist eine autonome Ablaufsteuerung z.B. für die Bearbeitung arithmetischer Operationen, das mit dem Befehlsablaufnetz kommuniziert. Beide Werke sind dem Prinzip nach Schaltwerke nach Kapitel 3, die man allgemein als Mikroprogrammsteuerwerke bezeichnet.

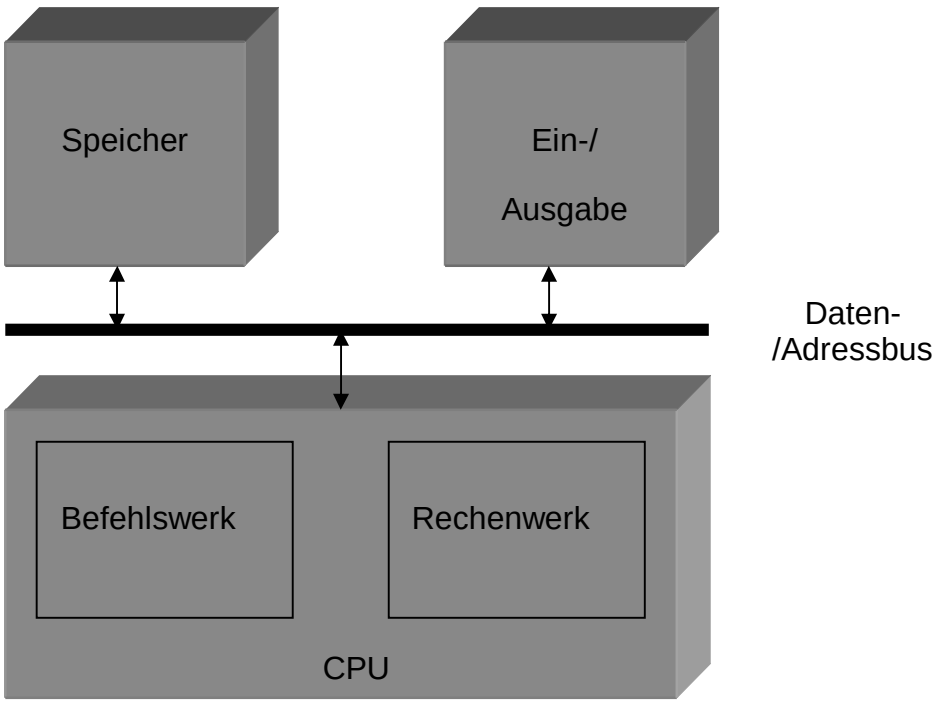


Bild 4-1: Blockdiagramm eines von Neumann-Rechners

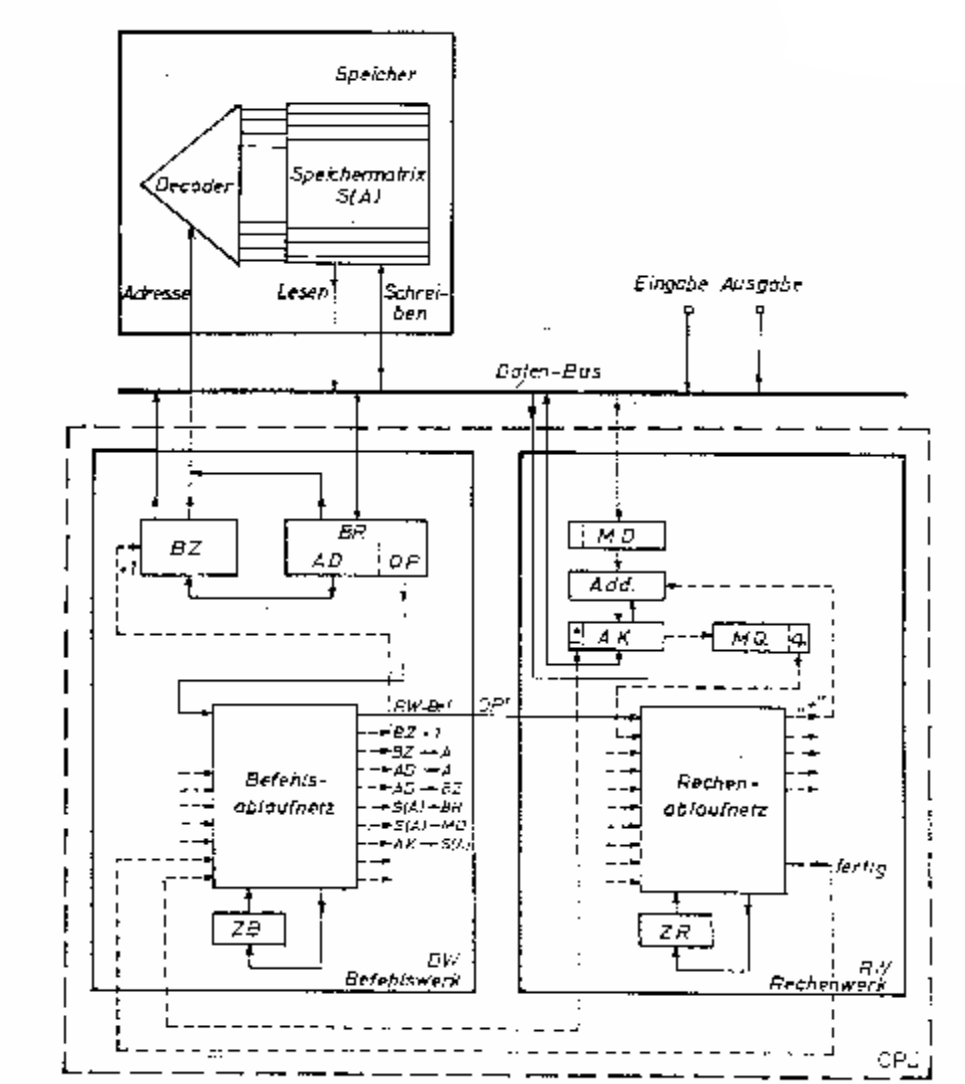


Bild 4-2: Blockschaltbild eines Universalrechners

Bild 4-2 zeigt das Blockschaltbild eines Universalrechners mit seinen Steuerwerken, der in Folgenden als Beispiel für die Architektur eines Mikrorechners dienen soll.

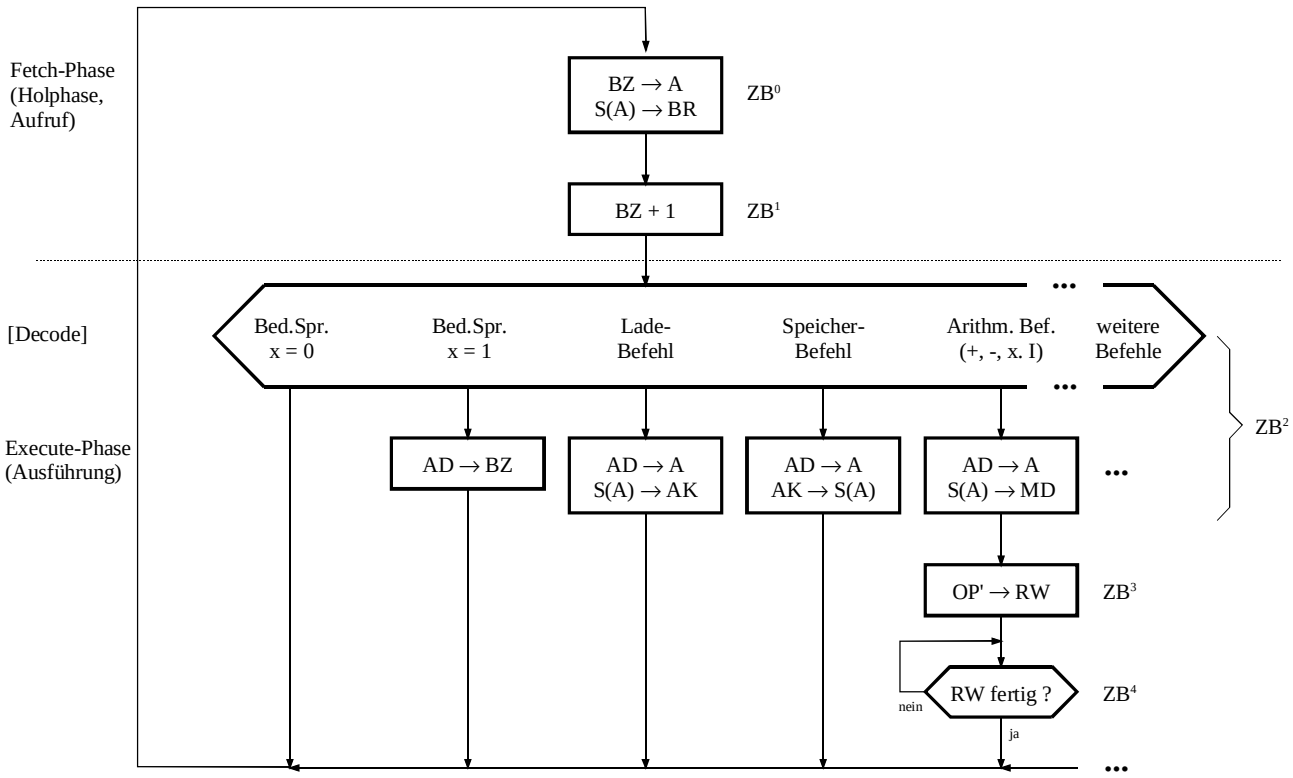
Die Mikroprogramme der Steuerwerke liegen i. a. für eine bestimmte Rechnerarchitektur fest. Allerdings gibt es auch veränderbare Mikroprogramme. Damit kann im Prinzip der Benutzer seine private Rechnerarchitektur, insbesondere seinen eigenen Maschinenbefehlscode, aufbauen.

Dem überlagert ist das eigentliche Programmlaufwerk mit dem vom Benutzer aufgestellten Programm im Arbeitsspeicher. Bild 4-3 zeigt den prinzipiellen Ablauf des Befehlsaufrufs mit Fortschaltung., Datentransport zwischen Speicher und Rechenwerk und Operationsauslösung.

Das Operationsprinzip des von Neumann-Rechners bedingt ein starres Zweiphasenschema der Programmabarbeitung. In der ersten Phase wird aufgrund der durch den Befehlszähler BZ angezeigten Adresse A ein Speicherzelleninhalt S(A) geholt und als Befehl (Operationscode OP im Befehlsregister BR) interpretiert (FETCH). In der darauf folgenden Phase wird aufgrund der im Befehl gefundenen Adresse AD ein Speicherzelleninhalt geholt und entsprechend der durch den Befehl gegebenen Vorschrift verarbeitet (EXECUTE).

Dies ist unzweifelhaft ein Prinzip maximaler Flexibilität, für das jedoch vor allem eine Leistungsminderung durch den Effekt des "von Neumann-Flaschenhalses" in Kauf genommen werden muss. Der Weg zwischen CPU und Speicher, über den ein Wort oder eine Adresse transportiert werden kann, wird der "von Neumann-Flaschenhals" genannt.

Da es die Aufgabe des Programms ist, den Inhalt des Speichers zu verändern, werden ständig Worte hin- und her gesendet. Dabei besteht ein Großteil des Verkehrs in diesem Flaschenhals nicht aus nützlichen Daten, sondern nur aus den Namen von Daten, wie auch aus Operationen und Daten, die benötigt werden, um solche Namen zu berechnen; d.h. ein Großteil dieses Verkehrs betrifft nicht die signifikanten Daten selbst, sondern die Informationen, wo diese zu finden sind.



Spr / $\bar{X}$ * = bedingter Sprung mit nicht erfüllter Sprungbedingung
Spr / X = bedingter Sprung mit erfüllter Sprungbedingung
ZB = Befehlswerk-Zustand

Bild 4-3: Flussdiagramm des Befehlswerkes

4.2 Realisierungsaspekte

Für den praktischen Einsatz von Rechenanlagen baut man aufwendigere Maschinen, um die Effektivität zu erhöhen und um einen wirtschaftlichen Kompromiss für folgende Grundanforderungen zu erzielen:

a) Ausnutzung des Speicherraumes:
z.B. möglichst große logische Leistung pro Befehlswort (Varianten: 1 Befehl/Wort, mehrere Befehle/Wort, mehrere Worte/Befehl, feste oder variable Befehlslänge)

b) Einfache Programmierbarkeit (für Mensch und automatische Übersetzer).
Häufig vorkommenden Operationen wird ein Befehlscode zugeordnet. Seltener gebrauchte Operationen setzt man aus mehreren Grundoperationen zusammen.

c) Minimaler Aufwand

d) Maximale Geschwindigkeit

Einteilung von Digitalrechnern
Die historische Einteilung (bis ca. 1990) berücksichtigte Mikro- und Minirechner sowie Mainframes, worunter man grob folgendes verstehen kann:

Ein Mikrorechner ist auch heute in der Definition ein Kleinrechner auf Mikroprozessorbasis, der i.a. nur einen Benutzer bedienen kann. "Klein" bezieht sich primär auf Größe und Preis, wohingegen früher wesentliche Kriterien wie Wortlänge oder Hauptspeicherkapazität heute zurücktreten (32, 64 Bit, bis zu mehrere GB Hauptspeicherkapazität). Schlagworte sind PC, Workstation.

Die Grenzen zur nächsten Klasse waren fließend: Minirechner sind danach mittlere Systeme, den mehrere Benutzer gleichzeitig bedienen können (Obergrenze etwa 30-40). Diese Art der Rechner ist wegen der Leistungssteigerung der PC´s im Prinzip verschwunden und eher durch Aufgaben spezialisierte Server-Rechner erstzt worden (z.B. File-, Druckserver).

Unter Mainframe versteht man i.a. einen Großrechner mit hoher Leistung bzw. Geschwindigkeit, den viele Anwender gleichzeitig benutzen können. Typische Kenndaten für Großrechner sind der Durchsatz, meist gemessen in MIPS (Million Instructions Per Second) oder FLOPS (Floating Point Operations Per Second), die Haupt- und Cachespeicherkapazität sowie die Anzahl der Kanäle zum Anschluss von Peripherie.

Heutzutage gilt parallel dazu eine Klassifikation nach Anwendungsfeldern wie z.B.

- Workstations
- Datenbanksysteme .
- Kommunikationsrechner wie Netzwerkserver
- Eingebettete Rechner ohne direkte Bedienerschnittstellen(z.B. Steuerrechner in Fahrzeugen etc.)

4.3 Struktur von Maschinenbefehlen

Die Anweisungen einer Maschinensprache lassen sich grundsätzlich in vier Gruppen aufteilen:

A) Anweisungen, die eine von der Zentraleinheit auszuführende Datenmanipulation spezifizieren (Rechenwerksbefehle)

B) Anweisungen, die den Interpretationsablauf eines Maschinenprogramms steuern (Sprungbefehle)

C) Anweisungen, die den Datenfluss zwischen Hauptspeicher und Zentraleinheit bzw. innerhalb der Zentraleinheit steuern (Transportbefehle)

D) Anweisungen zur Steuerung der Ein-/Ausgabe

Die Struktur der Anweisungen einer konkreten Rechenanlage ist im wesentlichen geprägt durch die Struktur der Anweisungen aus Gruppe A. Deshalb sollen sie hier näher betrachtet werden.

Jede dieser Anweisungen hat fünf Funktionen:

1. Spezifizierung des ersten Operanden (d.h. seiner Adresse)
2. Spezifizierung des zweiten Operanden(entfällt bei monadischen Operationen)
3. Spezifizierung der Adresse, an die das Ergebnis gebracht werden soll
4. Spezifizierung der Operation, die auf den beiden Operanden ausgeführt werden soll
5. Spezifizierungen der Adresse derjenigen Anweisung, die als nächste ausgeführt werden soll

Ein Teil dieser Spezifizierungen kann implizit erfolgen. So entfällt bei fast allen Maschinensprachen die Angabe darüber, welche Anweisung als nächste ausgeführt werden soll: Man nimmt einfach diejenige, die auf die augenblicklich ausgeführte im Speicher folgt (BZi-l-Mechanismus).

allgemein:

Information über die Operanden (Daten)	Information über die Operation

AD OP

Befehlstyp	OP	AD	Bedeutung
Rechenwerks-befehle (Zu A1)	+	n	<AK> := <AK> + <n>
	-	n	<AK> := <AK> - <n>
	*	n	<AK> := <AK> * <n>
	/	n	<AK> := <AK> / <n>
Sprung-befehle (Zu A2)	C	n	Nächster Befehl in n (statt <BZ> + 1)
	Cc	n	Nächster Befehl in n, wenn <AK> ≤ 0
	Ci	n	Nächster Befehl in n, wenn <i> ≥ 0
	Cu	n	Nächster Befehl in n, <BZ> + 1 --> n
Transport-befehle (Zu A3)	i	n	<i> := <n> (LOAD I)
	A	n	<AK>:= <n> (LOAD AK)
	S	n	<n> := <AK> (STORE AK)
Betriebs- u. E/A-Befehle (Zu A4)	stop	-	Anhalten
	Dr	-	<AK> drucken
	Z	-	Wagenrücklauf und Zeilen-sprung
	-	-	Leerbefehl (gehe weiter nach <BZ> + 1)

Bild 4-4: Beispiel einer Befehlsliste (Mikrorechner nach Bild 4-2)

Programmbeispiel 1 für den Mikrorechner:

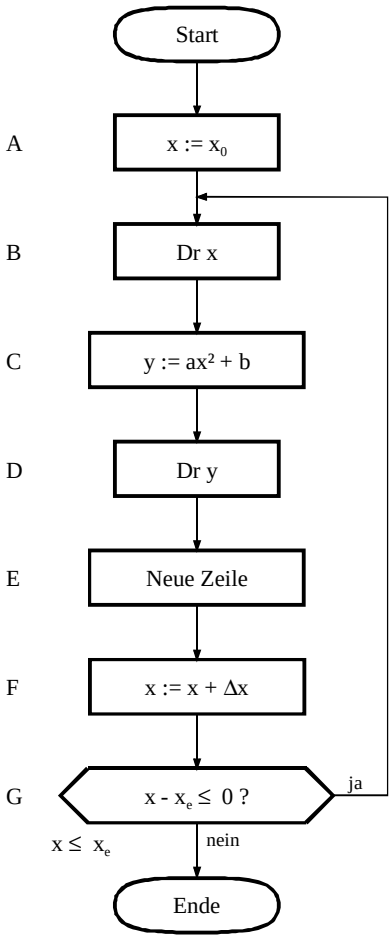
Folgende Berechnung soll durchgeführt werden:

Tabelle der Funktion $y = ax^2 + b$
Wertebereich $x_0 = 0$ bis $x_e = 100$
Schrittweite $\Delta x = 0{,}1$

Wertepaare von x und y nebeneinander drucken

Das Flussdiagrammzeigt den Ablauf der Berechnung und das Maschinensprachenprogramm (Assemblerprogramm) die Umsetzung der Berechnung für den Mikrorechner nach Bild 4-2.

Flussdiagramm



Assemblerprogramm

		OP	AD	<Ak>[1]
A	00	A	22	x_0
	01	S	25	x
B	02	A	25	x
	03	Dr		x
C	04	x	25	x^2
	05	x	20	ax^2
	06	+	21	$ax^2 + b$
D	07	Dr		y [2]
E	08	Z		y
F	09	A	25	x
	10	+	24	$x + \Delta x$
	11	S	25	x_{neu}
G	12	-	23	$x - x_e$
	13	CC	02	$x - x_e$
	14	Stop		Δx
⋮		⋮		
	20	(a)		
	21	(b)		
	22	$(x_0) = 0$		
	23	$(x_e) = 100$		
	24	$(\Delta x) = 0{,}1$		
	25	(x)		

[1] zur Programmprüfung
[2] y wird nur gedruckt, nicht gespeichert

4.3.1 Adressen

Eine Adresse kennzeichnet einen Platz im Speicher, an dem ein Operand oder ein Befehl gefunden werden kann.

(a) Mehrere Adressen pro Befehl

Für die Operation A + B = C müssten drei Operandenadressen angegeben werden Bild 4-5.

Adresse von A	Adresse von B	Adresse von C	Op-Code (Addition)

Bild 4-5: Befehlswortstruktur nach (a)

Bei Dreiadressbefehlen werden alle notwendigen Angaben explizit gemacht, so dass jede Anweisung selbsterklärend ist.

Bei Zweiadressbefehlen entfällt die Angabe von C. Da das Ergebnis nach wie vor festgehalten werden muss, ist eine implizite Spezifizierung notwendig. Dazu gibt es zwei Möglichkeiten:

- Das Ergebnis wird immer im Akkumulator, d.h. in einem einmal festgelegten Register, abgelegt. Damit dessen Inhalt anderweitig verwendet werden kann, muss eine Transportanweisung vom Akkumulator zum Hauptspeicher und umgekehrt existieren.

- Die Adresse von B ist gleichzeitig Adresse des Ergebnisses. Damit der ursprüngliche Operand über die Dauer dieser Anweisung hinaus erhalten bleiben kann, ist ebenfalls eine Transportanweisung notwendig, die allerdings auf mindestens zwei beliebige Adressen anwendbar sein muss.

damit stellt die zweite Möglichkeit eine Verallgemeinerung der ersten dar.

(b) Eine Adresse pro Befehl

Unter Zuhilfenahme des Akkumulators und der Auflösung eines Befehls in mehrere Befehle kann man mit einer Adresse pro Befehl auskommen (Bild 4-6).

Das Beispiel A + B = C muss dann z. B. in die folgende Sequenz von Ein-Adress-Befehlen aufgelöst werden:

1. Hole A nach AK
2. Addiere B zu AK
3. Speichere AK nach C

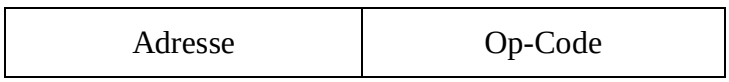


Bild 4-6: Befehlsstruktur nach (b)

Eine andere Möglichkeit ergibt sich durch die Festlegung, dass die Adresse des zweiten Operanden und des Ergebnisses der Akkumulator ist.

(c) Nulladressbefehle

Bei Nulladressbefehlen müssen alle drei Adressspezifikationen implizit erfolgen. Man legt dabei einen Kellerspeicher (Stack, LIFO-Prinzip) zugrunde, dessen zwei oberste Speicherplätze Akkumulatorfunktionen haben. Eine Operation läuft hierbei so ab, dass die beiden obersten Elemente des Kellerspeichers miteinander verknüpft und anschließend gelöscht werden. Das Ergebnis wird wieder in den obersten Speicherplatz des Kellerspeichers gebracht. Rechner mit einer solchen Nulladressstruktur nennt man auch Stackrechner..

typ. Rechenanweisungen: MOP (monadischer Operator für oberstes Speicherwort)
DOP (dyadischer Operator für die beiden obersten Speicherworte)

typ. Transportanweisungen: PUSH (auf den Stapel geben)
POP (vom Stapel abziehen)

Kurzadressen für Registerbefehle

Hat man z.B. eine Wortbreite von 22 Bit zur Verfügung, können nach (b) bei einer Op-Code-Breite von 7 Bit für die Adresse 15 Bit reserviert werden. Mit dieser Adresse kann dann direkt auf einen Speicher zugegriffen werden (Bild 4-7).

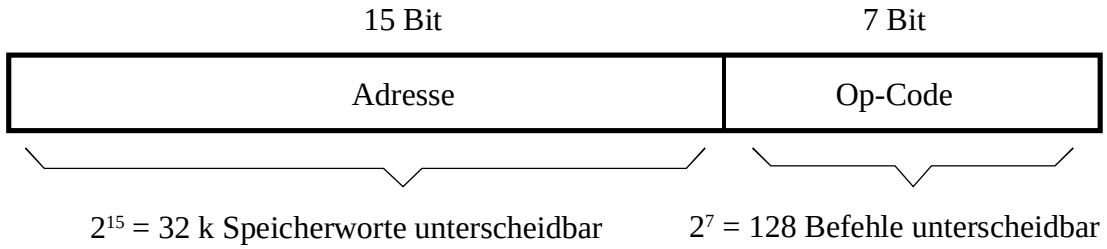


Bild 4-7: Speicherbefehlswortstruktur

Stehen die Operanden bereits in Registern (von denen es z.B. nur 16 gibt), wird für die Registeradressierung eine sehr viel kleinere Adressbreite benötigt. Man kann dann nach (a) zwei Befehle pro 22 Bit-Wort realisieren (Bild 4-8).

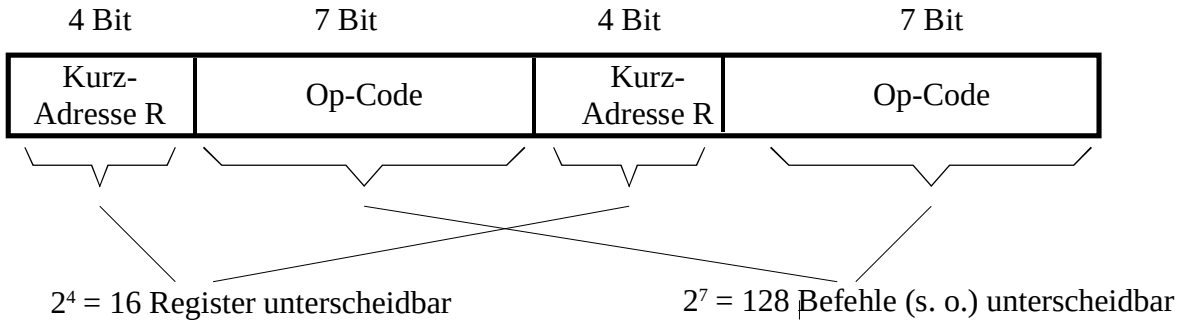


Bild 4-8: **Registerbefehlswortstruktur**

4.3.2 Adressierungsformen

Beim Motorola 68020 z.B. gibt es 18 verschiedene Adressierungsarten, die aus neun Basistypen gebildet werden. Hier sollen nur einige grundsätzliche Arten vorgestellt werden.

Unmittelbare Adressierung
Bei der unmittelbaren Adressierung (immediate) stellt die "Adresse" das Datenwort selbst dar (Bild 4-9).

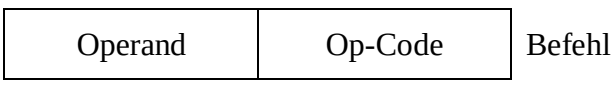


Bild 4-9: Unmittelbare Adressierung

Direkte Adressierung

Bei der direkten Adressierung wird die im Befehlswort enthaltene Adresse an den Decoder des Speichers gegeben, der den zu lesenden Operanden enthält (Bild 4-10).

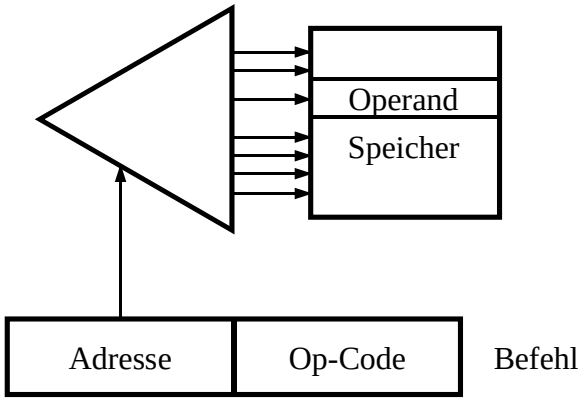


Bild 4-10: **Direkte Adressierung**

Relative Adressierungen

a) Adressenänderung I. Art (Basisadressierung)

Die im Befehl referierte Adresse stellt eine Adressverschiebung (displacement, auch offset genannt) dar, die additiv einer in einem Register gespeicherten Basisadresse hinzugefügt wird:

Absolute Adresse (n) = feste Basisadresse (n_o) + relative Adresse (Δn)

Das Programmieren mit relativen Adressen hat folgenden Vorteil:

a) Die relativen Adressen sind kürzer als die absoluten Adressen, weil sie nur eine Untermenge abdecken.

b) Man braucht sich bei der Adressenwahl nicht um die tatsächliche Speicherbelegung zu kümmern.

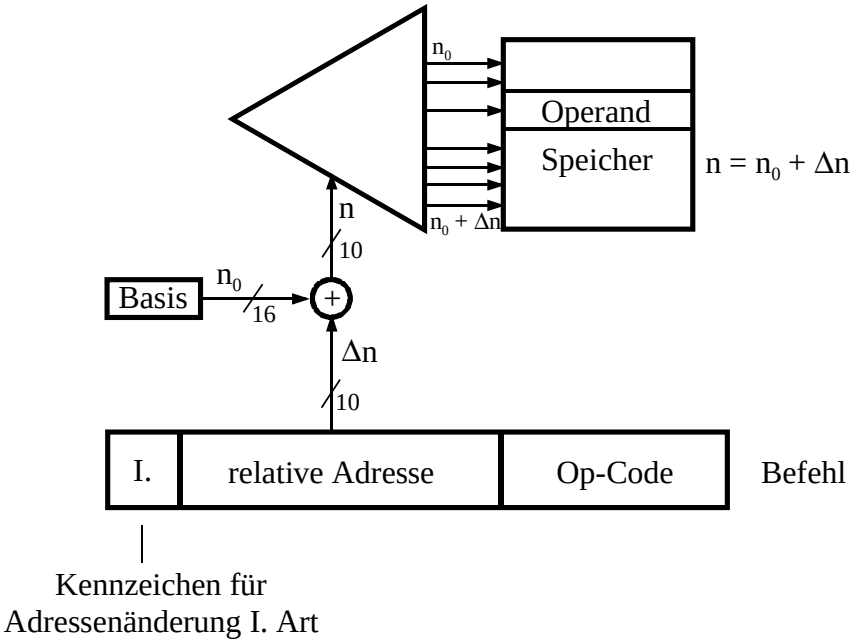


Bild 4-11: **Adressenänderung I. Art**

Bei der Eingabe des Programms und der Daten werden aus den relativen Adressen und der für einen Programmablauf festen Basisadresse (Anfangsadresse) die aktuellen Speicherplätze errechnet ($n = n_0$ tΔn) . Zum Wiederauffinden der in den Befehlen enthaltenen Operanden müssen deren relative Adressen jeweils um die Basisadresse erhöht werden. Dies geschieht entweder beim Einschreiben des Programms in den Speicher oder bei der Ausführung der einzelnen Befehle. Alle Befehle, die eine relative Adresse beinhalten, müssen eine entsprechende Kennzeichnung tragen (vgl. Bild 4-11).

Modifikation durch Zusatzbits

Durch Zusatzbits (Bild 4-12) können die Codes z.B. für die arithmetischen Befehle in Bezug auf Rundung, Normalisierung und höhere Genauigkeit modifiziert werden. Auch Adressenänderungsarten können damit gekennzeichnet werden.

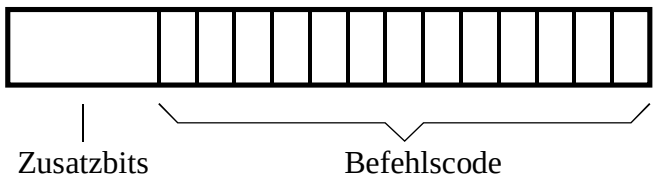


Bild 4-12: **Befehlsmodifikation durch Zusatzbits**

(b) Seitenadressierung (Paging)

Aus technischen Gründen kann es zweckmäßig sein, den logischen Adressraum in eine Anzahl von Unterräumen konstanter Größe einzuteilen. Da die Adressen eines Unterraumes zusammenhängend sind, lassen sich die entsprechenden Speicherplätze relativ adressieren. Diese Unterräume werden Seiten (pages) genannt. Die logische Adresse eines Speicherplatzes besteht dann aus der Seitennummer (Seitenadresse n.) und der relativen Adresse (Wortadresse n). Die gesamte logische Adresse n kann durch einfache Konkatenation von Seitennummer und relativer Adresse gebildet werden.

Beim Beispiel in Bild 7.12 bildet dann die Zusammenfassung der 5 Bit der Seitennummer und der 8 Bit der relativen Adresse die 13 Bit breite logische Adresse. Im Gegensatz zur Adressenänderung I. Art wird keine Addition zur Bestimmung der absoluten Adresse benötigt.

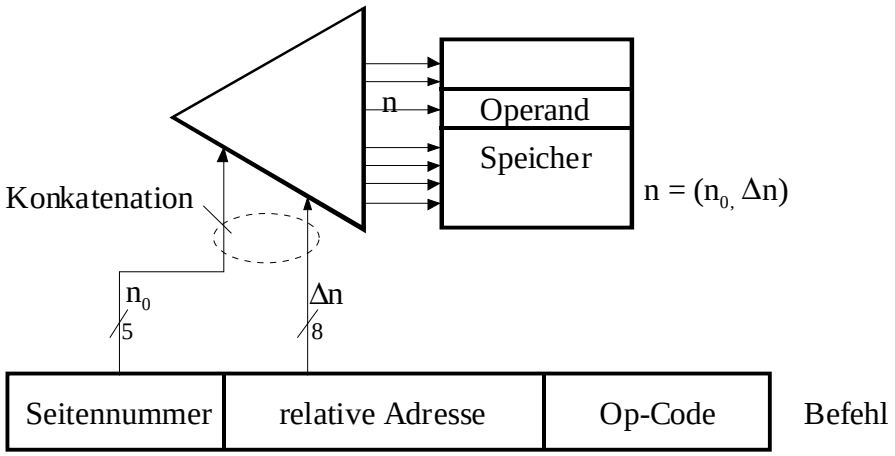


Bild 4-13: Seitenadressierung mit direkter Seitennummernangabe

Adressenänderung II. Art (indirekte Adressierung)
Bei der indirekten Adressierung ergibt sich die Adresse des gesuchten Wertes aus dem Inhalt des Speicherwortes der angegebenen Adresse (Adresse von Adresse):

Gesuchtes Wort = <<Angegebene Adresse >>

Diese Adressierungsart macht Sinn, wenn die eigentliche Adresse auch erst Ergebnis einer Berechnung ist.

Die Wortstruktur und den Zugriffsmechanismus unter Benutzung von zwei Speichern zeigt Bild 4-14. In Bild 4-15 ist die Adresse der Wortadresse im gleichen Speicher enthalten, in dem auch das gesuchte Wort (Operand) abgespeichert ist. Bild 4-14 zeigt ein Speicherbelegungsbeispiel.

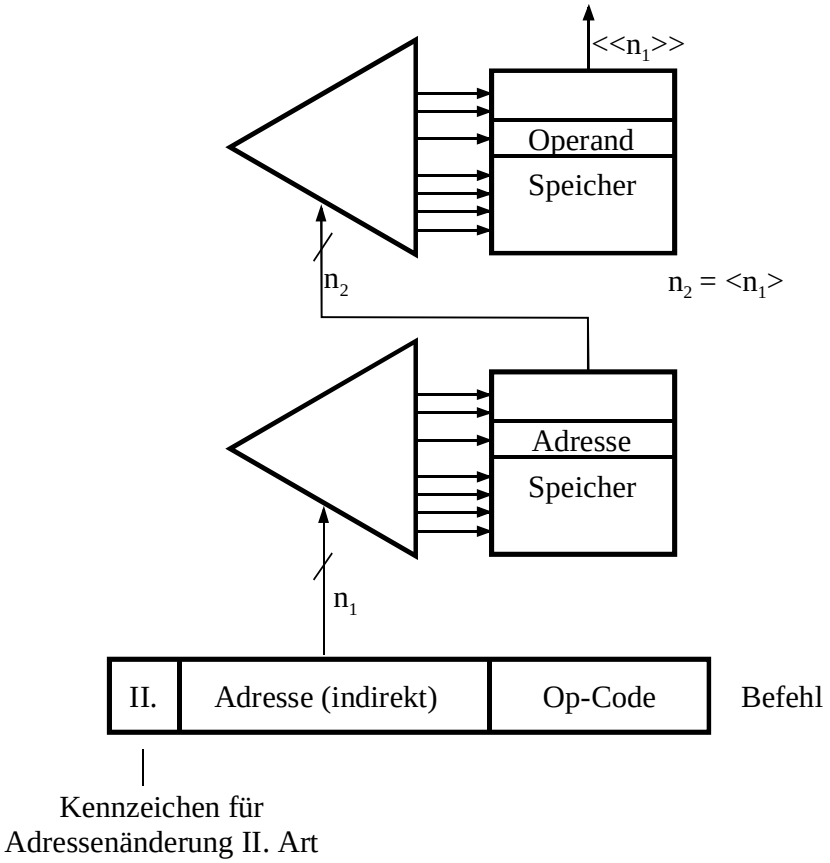


Bild 4-14: **Speicherbelegungsbeispiel für die indirekte Adressierung**

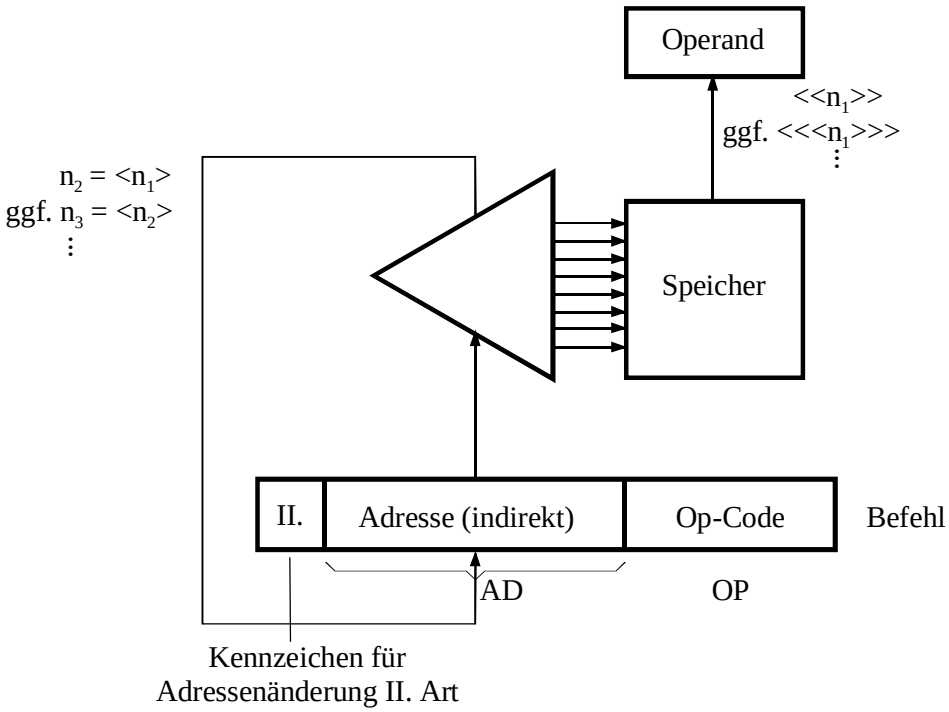


Bild 4-15: **Indirekte Adressierung**

Für die Ausführung der Adressänderung ist das Befehlswerk entsprechend dem Ablaufprogramm aus Bild 4-16a zu modifizieren.

Adressenänderung III. Art (indizierte Adressierung)
Bei der indizierten Adressierung werden Adressen benutzt, die additiv zu einer in einem Indexregister gespeicherten Adressverschiebung hinzugefügt werden. Im Gegensatz zur Adressierung mit Hilfe einer Basisadresse befindet sich dabei die Adressverschiebung (displacement) in einem im Befehl referierten Indexregister.

Adresse n:= Anfangs- oder Endadresse n_0 + Inhalt eines Indexregisters <I>

Für die Durchführung dieser Adressenrechnung wird folgendes benötigt:

Additionswerk für die Berechnung der Adresse (Addition kann in einem speziellen Werk oder auch im Rechenwerk erledigt werden), Indexregister (eines oder mehrere spezielle Register oder beliebige Speicherworte), Kennzeichen für Adressenänderung III. Art, Zusatzbefehle:

a) Lade Index-Register mit $-n_1$ (n_1 = Zahl der gewünschten Operationswiederholungen)

b) Verändere das Indexregister (+1, -1)

c) Bedingter Sprung, wenn das Index-Register den Bereichsendwert - z.B. 0 - erreicht hat

Diese Art wird im Prinzip wie ein Basisadressregister, aber mit der Möglichkeit die Adressberechnungsänderungen im Indexregister direkt auszuführen (+ 1, -1 etc.)

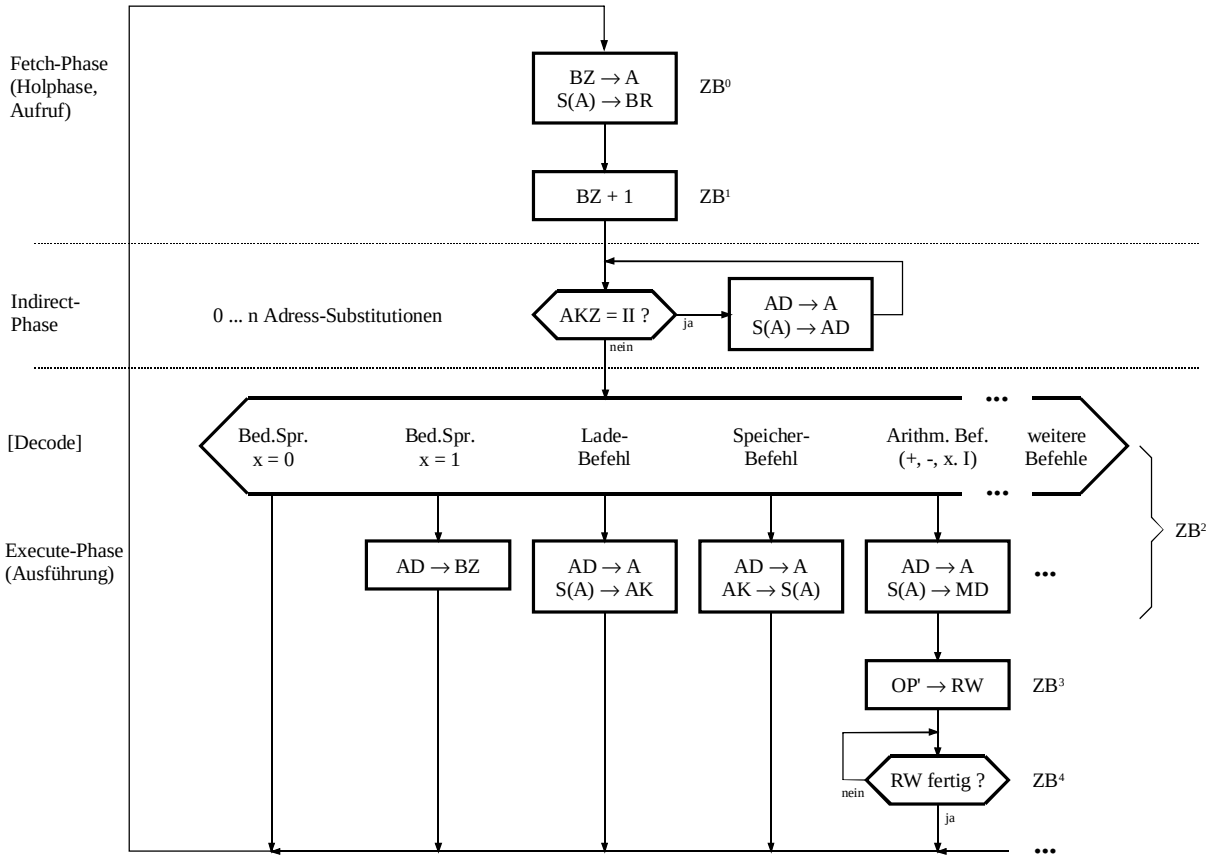


Bild 4-16 : Flussdiagramm des Befehlswerkes mit Indirect-Phase

Bild 4-17 zeigt ein Programmstück mit einer Adressenänderung III. Art und Belegung des Speicherraums mit den dazugehörigen Daten.

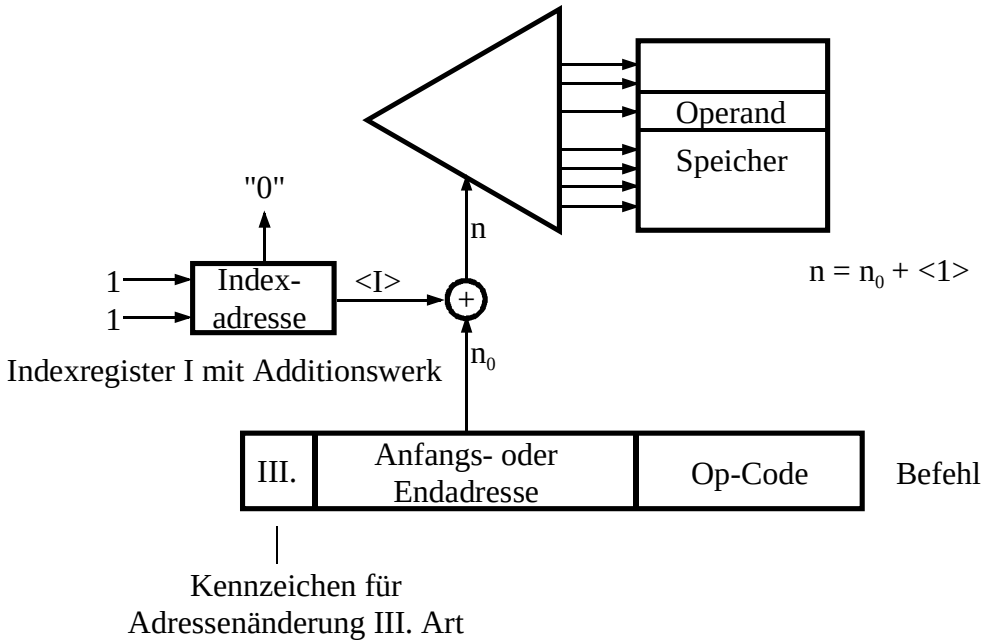


Bild 4-17: Indizierte Adressierung

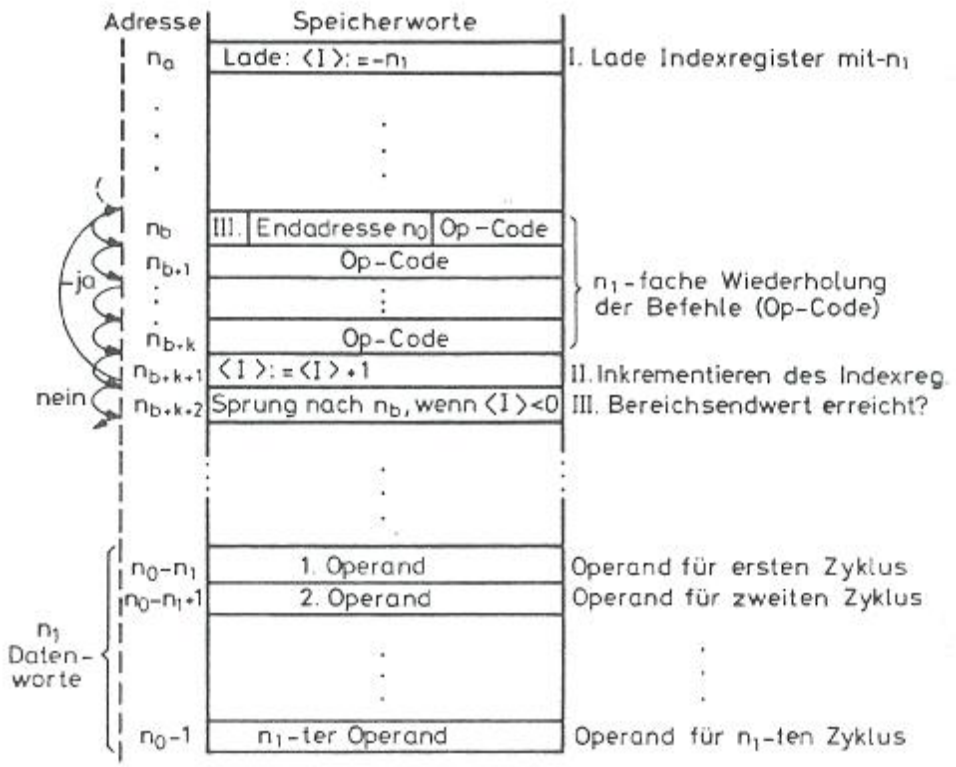


Bild 4-18: Programmausschnitt mit indizierter Adressierung

Ergänzung im Zusammenhang mit Registeradressierungen

Direkte Register-Adressierung: Analog zur direkten Adressierung (im Speicher) wird bei dieser Form die Registerbezeichnung R (Kurzadresse) zur Ermittlung des Operanden benutzt, d.h. der Operand ist gleich dem Inhalt des durch R bezeichneten Registers: Operand = <R>. Entsprechend gibt es die indirekte Register-Adressierung: Bei dieser Adressierungsform gilt: Operand = <<R>>, d.h. der Operand ist gleich dem Inhalt derjenigen Speicherzelle, deren Adresse in dem durch R spezifizierten Register steht.

Das folgende Programmbeispiel zeigt für den Universalrechner die Anwendung der Adressänderungen 2. und 3. Art. Dies ist im Befehlssatz nach Bild 4-4 möglich über das Indexregister (3. Art) und über den Speicher (2. Art) für Transportbefehle.

Programmbeispiel 2:

Adressenänderung 2. u. 3. Art (indirekt u. indiziert)

Statistische Auswertung von Prüfungsergebnissen

$n = 300$ Klausuren (numeriert mit k; $0 \le k \le n-1$), jede mit p Punkten bewertet ($0 \le p \le p_{max} = 60$)

Punktzahl p_k der Klausur k für wachsendes k in aufeinanderfolgenden Speicherzellen

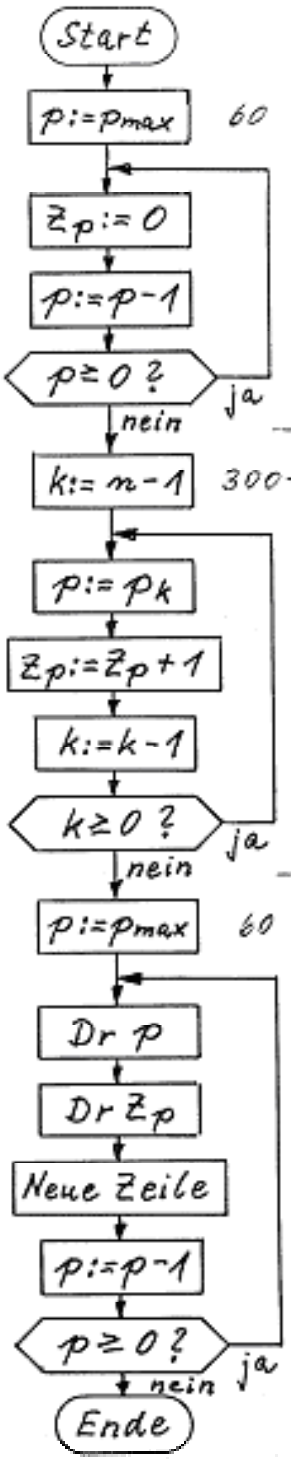
Auswertung (absolute Häufigkeit jeder Punktzahl):

In 61 Zählvariablen z_p (für $0 \le p \le p_{max} = 60$ Pkte) Klausuren mit der betreffenden Punktzahl p zählen (Strichliste)

Vor dem Zählen: Zählvariable löschen

Nach dem Zählen: Punktzahl p und Zählergebnis z_p als Wertepaar in einer Zeile drucken

Gegebene Liste			Auswertung		
k	p_k	z.B.	p	z_p	nach Abl.
0	p_0	$= 12$	0		$\rightarrow z_0$
1	p_1	$= 60$	1	/	$\rightarrow z_1$
⋮	⋮		⋮		
$n-1 = 299$	p_{299}	$= 1$	12	/	$\rightarrow z_{12}$
			⋮		
			$p_{max} = 60$	/	$\rightarrow z_{60}$



2a) Flußdiagramm

Zähler löschen:

p im Indexregister, deshalb p von 60 ... 0

Zählen:

k zur Auswahl der Klausur im Indexreg.

p zur Auswahl des Zählkanals in Speicher-var. zur indir. Adr.

Drucken:

p im Indexregister zur Auswahl des Zählkan. und für Endabfrage

p auch in Speicher-var. zum Drucken

26) Assemblerprogramm

Sp.-Adr.	Sp.-Inh. OP	Sp.-Inh. AD	<Ak>	<i>	
00	i	94	?	p_{max}=60	
01	A	91	0	p_{max}	
02	S	30+i	0	p	III
03	i-1		0	p:=p-1	
04	Ci	02	0	p	
05	i	95	0	n = 300	
06	i-1		0	k_{max}=299	
07	A	100+i	p_k	k	III
08	+	93	30+p_k	k	
09	S	96	30+p_k	k	
10	A	96"	Z_p	k	II
11	+	92	Z_p+1	k	
12	S	96"	$Z_{p\,neu}$	k	II
13	i-1		Z_p	k:=k-1	
14	Ci	07	Z_p	k	
15	i	94	Z_p	p_{max}=60	
16	A	94	p_{max}=60	p_{max}	
17	S	96	p_{max}	p_{max}	
18	Dr		p	p	
19	A	30+i	Z_p	p	III
20	Dr		Z_p	p	
21	Z		Z_p	p	
22	i-1		Z_p	p:=p-1	
23	A	96	p_{alt}	p_{neu}	
24	-	92	p_{alt}-1	p_{neu}	
25	S	96	p_{neu}	p_{neu}	
26	Ci	18	p	p	
27	Stop		-1	-1	

Sp.-Adr.	Sp.-Inh. Daten
30	(Z_0)
31	(Z_1)
⋮	⋮
90	(Z_{60})
91	0
92	1
93	(a_{Z0}) = 30
94	(p_{max})=60
95	(n) = 300
96	(Laufvar.)
100	(p_0)
101	(p_1)
⋮	
3299	(p_{299})

grüne Registerinhalte nur zur Verdeutlichung des Programmablaufes
rot: Adr.-änd.

markiert II: Adressänderung 2. Art
III: Adressänderung 3. Art

4.4 Weitere typische Befehle

4.4.1 Befehle von Mikrorechnern

Im Zusammenhang mit den Adressen hat sich gezeigt, dass diese im wesentlichen die Anweisungen der Gruppe C festlegen. Anweisungen zur Steuerung der Ein-/Ausgabe (Gruppe D) sollen hier nicht weiter besprochen werden, so dass sich dieser und der nächste Abschnitt auf die beiden ersten Gruppen beschränken kann. Zur groben Charakterisierung lässt sich sagen, dass die Anweisungen der Gruppe A Aktionen im Rechenwerk und die der Gruppe B Aktionen im Befehlswerk auslösen.

Logische und arithmetische Befehle
Zu den elementarsten Manipulationen von Binärfolgen zählen logische Operationen. Allgemein üblich sind die vier folgenden Anweisungen:

- AND : logisches UND
- OR: logisches ODER
- XOR : exklusives ODER
- NOT : Negation

Die jeweils korrespondierenden Stellen des Operanden werden gemäß der spezifizierten Operation miteinander verknüpft.

Arithmetische Anweisungen für ganzzahlige Operanden
Für die Manipulation ganzzahliger Operanden gibt es Additions-, Subtraktions-, Multiplikations- und Divisionsanweisungen. Ihre Realisierung kann unterschiedlich erfolgen.

Arithmetische Anweisungen für Dezimalzahlen
Einige Rechner verfügen zusätzlich auch über solche Anweisungen, die eine Manipulation von Dezimalzahlen erlauben.

Arithmetische Anweisungen für Gleitkomma-Operanden
Für Gleitkomma-Operanden gibt es wiederum die vier Operationen Addition, Subtraktion, Multiplikation und Division. Bei Kleinrechnern sind derartige Anweisungen allerdings meist nicht vorhanden. Aus diesem Grunde soll im folgenden die Wirkung der Gleitkomma-Addition beschrieben werden, damit ersichtlich wird, wie mit Hilfe von ganzzahligen Operationen und Verschiebeoperationen eine Realisierung in Form eines Programms vorgenommen werden kann.

Gleitkomma-Addition

1. Angleichung der Exponenten:
   Der kleinere Exponent wird an den größeren durch Rechtsverschiebung angeglichen. Falls die Differenz zwischen beiden größer ist als die Mantissenlänge, werden alle signifikanten Stellen heraus geschoben, so dass die Summe gleich der größeren Zahl ist.

2. Addition der Mantissen:
   Die Mantissen werden als ganze Zahlen interpretiert und entsprechend addiert.

3. Abschlussbehandlung:
   Abhängig von den Vorzeichen der beiden Operanden werden im letzten Schritt Korrekturen am Ergebnis vorgenommen.

gleiche Vorzeichen:
Ist bei der Addition der Mantissen ein Überlauf aufgetreten, so ist die Mantisse des Ergebnisses um eine Stelle nach rechts zu schieben (bei gleichzeitigem Nachziehen einer "l"). Hat der Exponent allerdings schon eine maximale Größe erreicht, führt dieser Schritt zu einem Überlauf ungleiche Vorzeichen:
Ist das Ergebnis Null, wird eine saubere Null als Ergebnis produziert. Andernfalls erfolgt eine Normalisierung der Matisse.

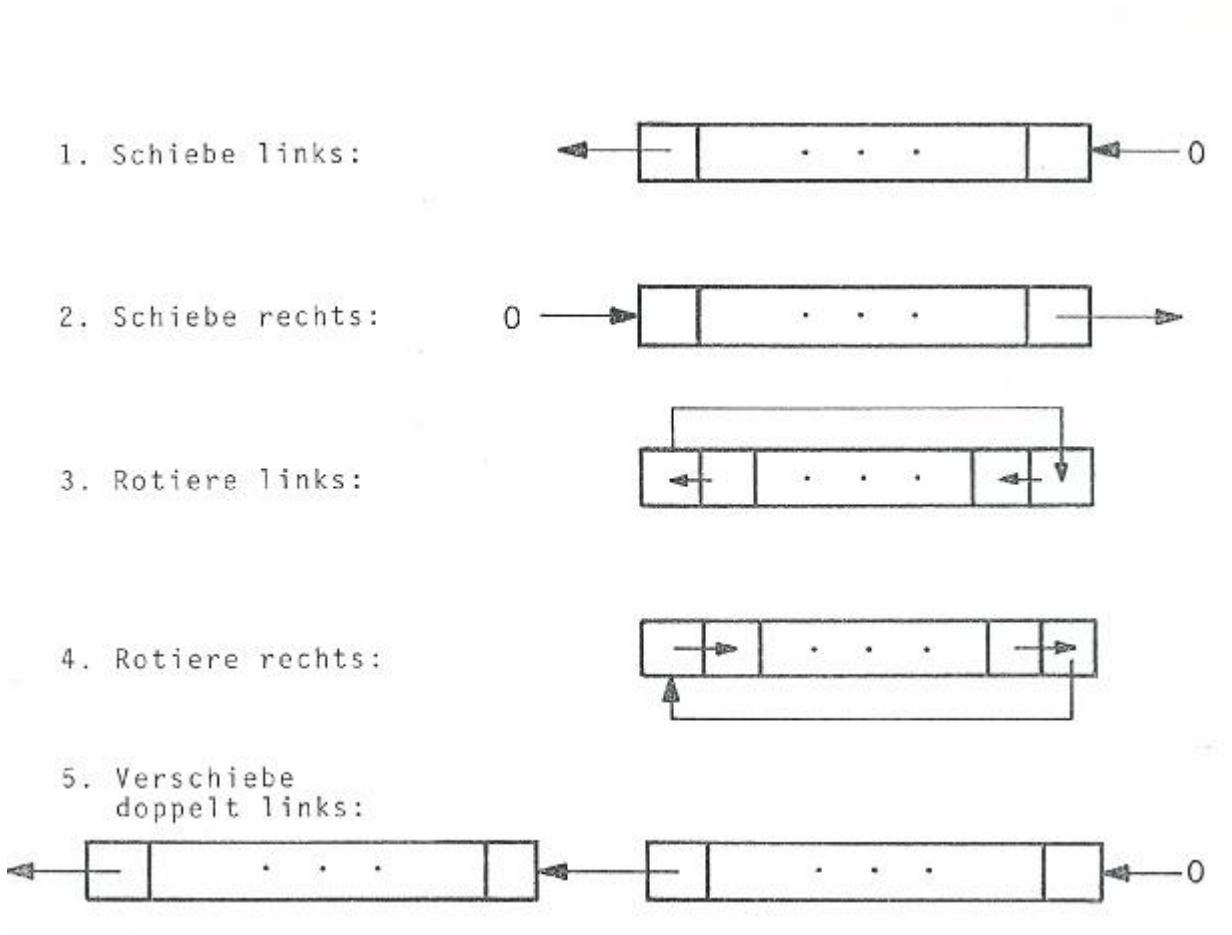
Schiebebefehle
Die meisten Rechner verfügen über zwei Arten von Schiebeoperationen (Shift):

- logische: Hierbei wird der Inhalt eines Registers oder einer Speicherzelle als Bitfolge aufgefasst

- arithmetische: Hierbei wird der Inhalt eines Registers als Binärzahl in 2-Komplement-Darstellung aufgefasst

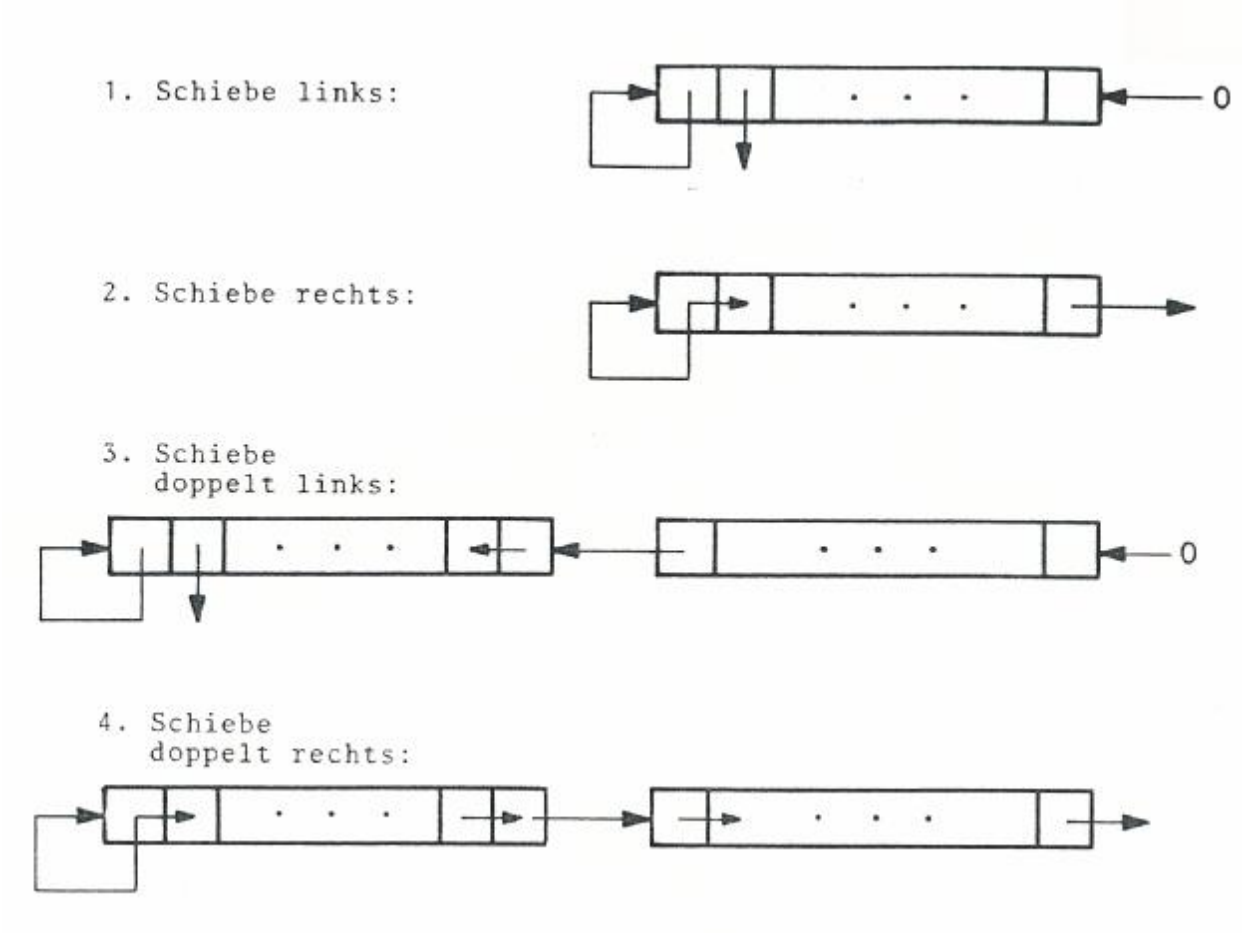
Die Verschiebungen erfolgen je nach Rechner entweder grundsätzlich um eine Stelle oder aber um beliebig viele Stellen.

Logische Verschiebungen



Analog zur 5. Anweisung gibt es auch die Anweisungen 2 bis 4 für doppelt lange Operanden.

Arithmetische Verschiebungen:
Bei beiden Schiebeoperationen bleibt das Vorzeichenbit erhalten. Die Verschiebung um eine Stelle nach links entspricht demnach einer Multiplikation mit 2, die Verschiebung nach rechts einer (ganzzahligen) Division durch 2.



Vergleichsbefehle
Vergleichsbefehle ermöglichen ein Setzen von Anzeigen, ohne dass das Ergebnis des Vergleichs wie bei anderen Anweisungen explizit abgespeichert werden muss. Sie haben damit große Bedeutung für die weiter unten beschriebenen Sprungbefehle.

Vergleichsbefehle kann man ganz allgemein so charakterisieren, dass sie auf Hauptspeicherzellen und/oder Registern oder Teilen davon (durch entsprechende Masken ausgewählt) operieren und deren Inhalt im Sinne von Bitfolgen, ganzen Zahlen o.ä. miteinander vergleichen, d.h. entweder eine logische Verknüpfung oder eine Subtraktion durchführen.

4.4.2 Befehle zur Steuerung des Programmablaufs

Wie bereits erwähnt, werden Maschinenanweisungen sequentiell ausgeführt, wobei sich die Reihenfolge durch die Anordnung der Anweisungen im Hauptspeicher ergibt. Für die Unterbrechung dieser Reihenfolge gibt es spezielle Anweisungen. Sie ermöglichen die Verlagerung der Interpretation zu einer anderen Anweisung als der sequentiell folgenden.

Sprungbefehle
Sprünge stellen eine Möglichkeit zur Steuerung des Programmablaufs dar. Man unterscheidet auf Maschinensprachebene zwischen unbedingten und bedingten Sprüngen. Ihre Wirkung besteht darin, dass die in der Anweisung enthaltene Hauptspeicheradresse neuer Inhalt des Programmzählers wird. Bei bedingten Sprüngen wird diese Veränderung vom Zustand verschiedener Statusanzeigen oder von explizit angegebenen Vergleichsoperationen abhängig gemacht. Gebräuchliche Formen bedingter Sprünge sind:

- Springe, falls Ergebnis gleich null
- Springe, falls Ergebnis kleiner null
- Springe, falls erster Operand kleiner als zweiter
- Springe, falls Anzeige mit Maske xxx übereinstimmt etc.

Unterprogrammsprünge
Eine besondere Form zur Steuerung des Programmablaufs stellen Unterprogrammsprünge dar. Sie unterscheiden sich von den oben genannten Sprüngen in erster Linie dadurch, dass die Stelle, von der aus der Sprung erfolgt, auch noch nach dessen Durchführung bekannt sein muss, um nach Beendigung des Unterprogramms eine Rückkehr an diese Stelle zu ermöglichen.

Für die Rettung der Rückkehradresse gibt es kein allgemeines Konzept. Es kann daher an dieser Stelle nur exemplarisch auf Realisierungsmöglichkeiten eingegangen werden.

a) Die Rücksprungadressen werden auf einem Stapelspeicher abgelegt, wodurch geschachtelte Unterprogrammaufrufe sehr leicht möglich sind.

b) Die Rücksprungadresse wird in einem allgemeinen oder speziellen Register abgelegt. Bei geschachtelten Unterprogrammaufrufen wäre dann vom Programmierer eine entsprechende Aufbewahrung aller alten Rücksprungadressen notwendig.

c) Am Anfang eines Unterprogramms wird eine Speicherzelle reserviert, in die die Rücksprungadresse automatisch eingetragen wird.

Die Rückkehr an die aufrufende Stelle eines Unterprogramms kann im letzten Fall durch einen unbedingten Sprung ausgeführt werden, in den anderen Fällen gibt es eine spezielle Anweisung, die den Aufrufvorgang umkehrt.

4.4.3 Unterbrechungen

Eine Unterbrechung, auch Interrupt genannt, ist eine Art impliziter Unterprogrammaufruf, der automatisch bei Vorliegen gewisser Bedingungen erfolgen kann. Die Ursachen, die das Eintreten von Unterbrechungsbedingungen haben kann, lassen sich in drei Gruppen einteilen:

a) Interne Unterbrechungsursachen:

- Programmfehler
  - § Division durch Null
  - § Bereichsüberschreitung bei arithmetischen Operationen
  - § Missachtung von Schutzmechanismen (z.B. Schreiben in
  - § eine Seite, auf die nur lesender Zugriff erlaubt ist)
  - § undefinierter Operationscode
  - § unzulässige Hauptspeicheradresse

- § Stacküberlauf
- § Versuch, ein nicht vorhandenes E/A-Gerät anzusprechen

- Hardwarefehler
  - § Stromausfall

  Andere Fehler, wie der Ausfall eines Speichermoduls, werden in älteren Rechenanlagen nur indirekt als Programmfehler sichtbar. In neuerer Zeit findet man in größeren Anlagen sog. Wartungsprozessoren, die die einzelnen Baugruppen einer Platine kontrollieren und somit beim Auftreten eines Hardwarefehlers gezielte Information über die Fehlerquelle geben können.

- generierte Unterbrechungen

  Viele Rechenanlagen erlauben die explizite Generierung von Unterbrechungen. Dies geschieht dadurch, dass im Programmstatuswort ein Trap-Bit gesetzt wird, das die Unterbrechung des ausgeführten Programms nach der nächsten Anweisung bewirkt. Diese Möglichkeit wird beispielsweise bei Testhilfen für Maschinenprogramme genutzt. Alle internen Unterbrechungen, auch Traps genannt, treten synchron zu dem jeweils ablaufenden Programm auf.

b) Externe Unterbrechungsursachen:
Externe Unterbrechungen werden von Ein-/Ausgabegeräten verursacht und zwar asynchron zum jeweils ablaufenden Programm. Ursache ist das Senden bestimmter Statusnachrichten eines Gerätes, wie "Controller bereit", "Fehler" u.ä. Diese bewirken, dass in der Hardware, welche die Anpassung des Gerätes an das Ein-/Ausgabesystem des Prozessors vornimmt (Schnittstelle, Interface), ein Unterbrechungssignal erzeugt wird.

c) Programmierte Unterbrechungen:
Fast alle Rechner verfügen über spezielle Maschinenanweisungen, welche die gleiche Wirkung wie die beiden anderen Gruppen von Unterbrechungsursachen haben. Man nennt diese Anweisungen Systemaufrufe, weil sie dazu dienen, Betriebssystemfunktionen von Benutzerprogrammen her aufzurufen. Diese Methode hat gegenüber Unterprogrammaufrufen den Vorteil, dass der Aufrufer nicht die Anfangsadresse der aufzurufenden Prozedur kennen muss, was zu einer Trennung von Betriebssystem und Benutzersoftware führt.

Bei allen genannten Unterbrechungen müssen in einem Prozessor prinzipiell folgende Schritte realisierbar sein:

1. Der Prozessor wird durch ein Unterbrechungssignal veranlasst, nach Abarbeitung der gerade interpretierten Anweisung nicht zur sequentiell folgenden überzugehen.

2. Die Adresse dieser Anweisung, die im Befehlszähler steht, muss an einer vereinbarten Stelle im Hauptspeicher bzw. auf einem Stack (falls ein solcher vorhanden ist) gespeichert werden, um eine spätere Fortsetzung des unterbrochenen Programmes zu ermöglichen.

3. Die Anfangsadresse der Unterbrechungsbearbeitungsroutine muss an einer fest vereinbarten Stelle im Hauptspeicher verfügbar sein.

4. Für diese Routine müssen Hinweise über die Unterbrechungsursache im Hauptspeicher oder in speziellen Registern zur Verfügung stehen.

5. Es sollte möglich sein, die Annahme von Unterbrechungssignalen für eine gewisse Zeit zu unterbinden (maskieren). (Dies ist allerdings nicht in allen Prozessoren möglich.)

Überlagern sich Unterbrechungen, d.h. trifft während der Ausführung einer Unterbrechungsbearbeitungsroutine ein weiteres Unterbrechungssignal ein, so muss wieder nach diesem Schema verfahren werden. Ob ein derartiges Signal jedoch akzeptiert wird, wird in einem Prioritätsschema geregelt. Jedem möglichen Unterbrechungssignal wird dabei eine Maßzahl für die Dringlichkeit der Bearbeitung (Priorität) zugeordnet; nur Unterbrechungssignale, deren Priorität größer ist als die der gerade bearbeiteten Unterbrechungsursache, können wirksam werden.

4.5 Speicher

1) Grundelement: 1 Bit-Speicher

z.B. Flipflop	Grundfunktionen	Schreiben
		Lesen
		Speichern

Wir können durch entsprechended Anordung der Grundfunktion größere Informationsmengen speichern, z.B. Bytes, Worte, Menge von Worten. Dann kann dazu noch die Grundfunktion des Auswählens kommen. Man kann die Speicherung von Informationen verschieden strukturiert vornehmen und kommt so zu unterschiedlichen Speichertypen. Außerdem ergibt sich durch verschiedene Technologien ein unterschiedliches Verhalten.

2) Allgemeine Kenngrößen eines Speichers:

Kapazität: z.B. Bit, Byte, Wort, Block (Menge von Worten).
Sie begrenzt den Umfang der Aufgaben, die bearbeitet werden können.

Zugriffszeit: Sie ist abhängig von der technischen Realisierung und der Organisation des Speichers und ein Maß für die Verarbeitungsgeschwindigkeit. Man versteht darunter die zum Auffinden und Lesen bzw. Schreiben einer Information benötigte Zeit. Unter Zykluszeit (wesentlich bei Speichern mit zerstörendem Leseverfahren, z.B. dynamische Speicher, wird die Zeit verstanden, die zum Auffinden, Lesen und Wiedereinschreiben einer Information benötigt wird.

Kosten: Meist gerechnet in Preis/Bit. Die Kosten für den/die Speicher beeinflussen oft beträchtlich den Gesamtpreis einer DV-Anlage.

3) Speicherarten (nach Speichermedium unterschieden):

Mechanische Speicher	Lochkarte, Lochstreifen (historisch)
Magnetische Speicher	a) Kernspeicher (historisch) b) Oberflächenspeicher, z.B. Magnetplatte, Magnetband, Floppy, magneto-optische Verfahren (CD-RW) c) neue Technologien wie MRAM

Optische Speicher	a) Holographische Speicher b) Laserprinzip (CD-ROM, DVD)
Halbleiterspeicher	Register, FIFO, RAM (Schreib/Lesespeicher) statisch-dynamisch, ROM (Festwertspeicher) Neue Technologien wie Ferroelektrische RAM´s

4) Speicherorganisation:

Hier ist zwischen zwei großen Gruppen von Speichern zu unterscheiden:

- Speicher mit sequentiellem (seriellem) Zugriff; die Zugriffszeit zu einer Speicherzelle hängt von ihrer räumlichen Lage im Speicher ab (z.B. Platten, Bänder).

- Speicher mit beliebigem (random) Zugriff; die Zugriffszeit zu jeder Zelle ist etwa gleich groß (RAM), z.B. Matrixspeicher

	Kapazität	Zugriffszeit	Art
Magnetplatte	>100GB	einige ms	sequentiell/blockorientiert
Halbleiterspeicher Dyn. RAM	>100MB	100ns	wahlfrei/blockorientiert

Bild 4-19: Typische Kennwerte für Speicher (Stand 2004)

4.5.1 Register

Mit einem FF kann ein Informationsbit gespeichert werden. In Registern können mehrere Informationsbits gespeichert werden. Zum Aufbau werden entsprechend der Länge der zu speichernden Information mehrere FFs benötigt.

Serielle Register

Bei vielen Anwendungen ist es üblich, die Information seriell in eine Kette von Flipflops zu schieben. Nutz man dazu elementare RS-FF´s, muss grundsätzlich dafür gesorgt werden, dass der "alte" Inhalt eines Elements übertragen wurde und das Flipflop "leer" ist, bevor ein "neuer" Inhalt eingeschrieben wird. Die Transportsteuerimpulse müssen also vom Ausgang zum Eingang hin zeitlich gestaffelt sein. Mehrere derartige Steuerwellen können gleichzeitig existieren ("Pipelines", 11Systolic Arrays"). Der übliche Extremfall ist, dass jedes zweite Flipflop gleichzeitig getaktet wird, was wiederum zu einer Realisierung mit Master-Slave-Flipflops führt.

Das Blockschaltbild eines Schieberegisters ist in Bild 4-20 dargestellt. Mit jedem Schiebetakt (Shift-Mikrooperation, Sh) wird der Inhalt des Registers A um eine Position nach rechts geschoben. Das vollständige Auslesen eines Registers mit n Bits dauert n Taktperioden, aber die Zahl der äußeren logischen Anschlüsse ist - im Gegensatz zur parallelen Transportweise - minimal.

Oft ist es notwendig, den Ausgang eines Schieberegisters wieder auf den Eingang zurückzuführen (Bild 4-21). Bei derartigen (seriellen) Betriebsweisen gibt es außer dem Bit-Takt (Sh) auch den

"Worttakt", der die n-fache Periodenlänge hat und angibt, wann die Information in der "richtigen" Position im Schieberegister (Shiftregister) steht. Der Transport eines ganzen Wortes in ein Shiftregister (A) erfolgt nach Bild 4-22 durch Steuerung des Eingangsmultiplexers über n Taktperioden.

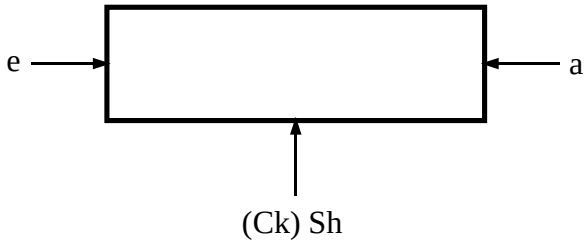


Bild 4-20: Serieller Datentransport (Schieberegister)

Parallel ladbare Schieberegister kann man, wie in Bild 4-23 gezeigt, in kompakter Weise darstellen.

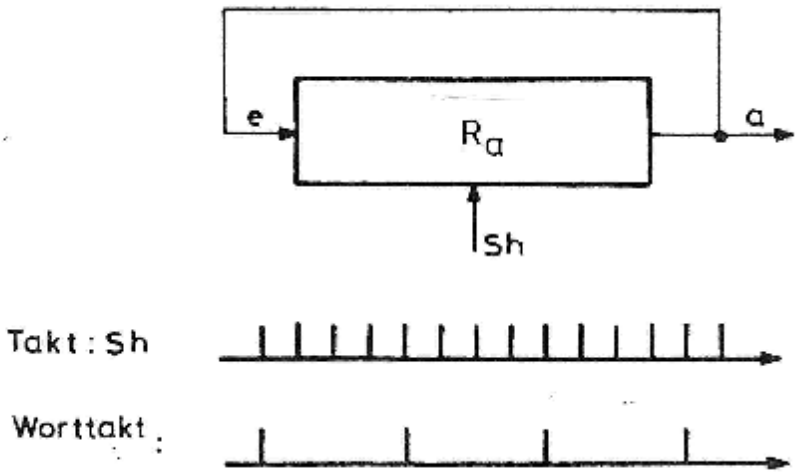


Bild 4-21: Ring-Schieberegister (Rückgekoppelte) Schiebekette

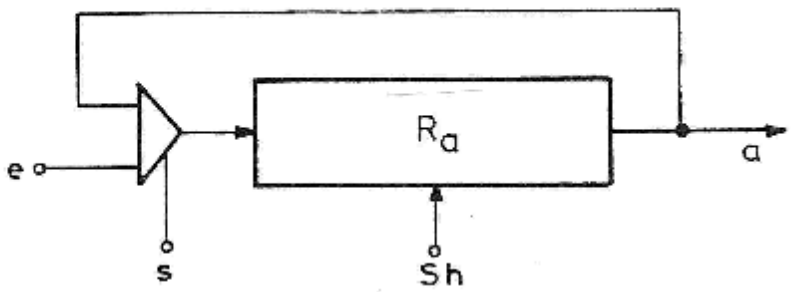


Bild 4-22: Ladbares Ring-Schieberegister

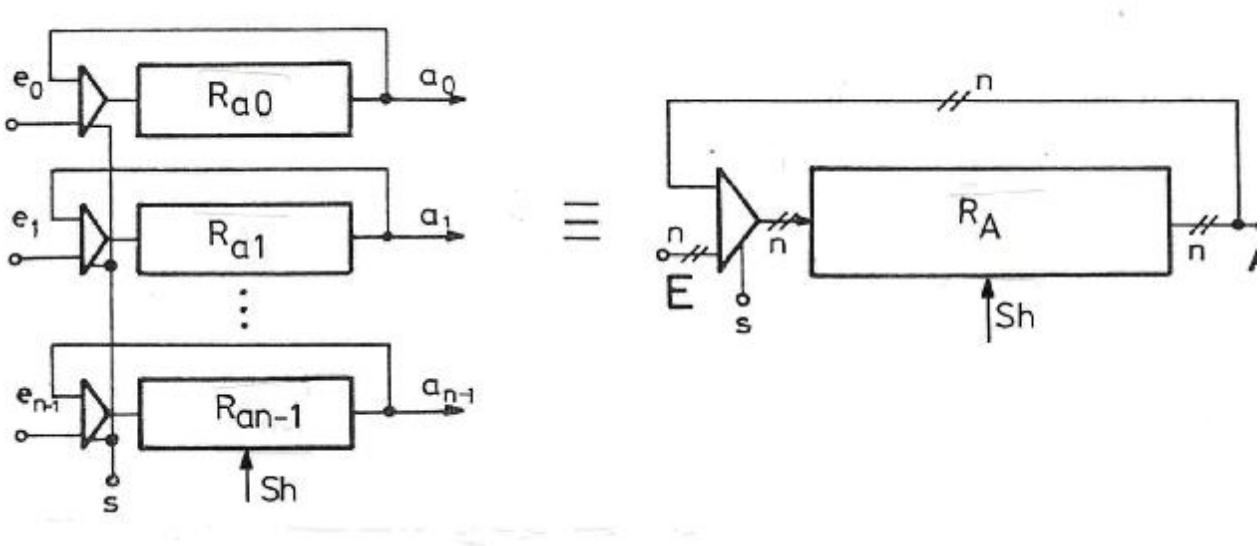


Bild 4-23: Parallele Schieberegister

Parallele Register (im Kontext von parallelen Transporten)

Eine häufige Grundoperation ist der parallele (gleichzeitig, concurrent) Informationstransport ("Worte") von ganzen bzw. in ganze Gruppen von Flipflops ("Register"). Bild 4-24a und b zeigt den Transport von Register A nach Register B wiederum für RS- und D-Flipflops. Die Teilbilder 4-24c und d zeigen Kurzsymbole, die auch schon früher erwähnt wurden.

Typisch für parallele Transporte ist die hohe Geschwindigkeit, aber auch der große Aufwand an Gattern und Verbindungsleitungen.

4.5.2 Schreib-/Lesespeicher (RAM)

Unter einem Adressspeicher (Random Access Memory = RAM) versteht man eine matrixförmige Anordnung von "Speicherworten", die mit Hilfe einer "Adresse" (A) Über einen Decoder in beliebiger ("random11) Reihenfolge zum Schreiben oder Lesen ausgewählt werden können. Bild 4-25 zeigt das Blockschaltbild eines RAM mit einem zusätzlichen "Adressregister" (AR) und einem "Speicherregister" (SR) und Bild 4-26 zeigt das prinzipielle Schaltbild einer Speicherzelle innerhalb einer Matrix.

Bild 4-27 zeigt ein mögliches logisches Detailschaltbild mit RS-Flipflops und anderen Standard-Logikelementen. Zum Schreiben wird die Steuerleitung W (= write) aktiviert, so dass die Information, die an den Eingangsleitungen ($I = (i_o, i_1 \ldots i_{m-1})$ liegt, die Setz- bzw. Rücksetzeingänge der entsprechenden Flipflops des ausgewählten Wortes durch die "verdrahtete "ODER-Schaltung" auf die Leseleitungen (1_μ) gegeben. Der Ausgang ($0 = (0_0, 0_1 \ldots 0_{m-1})$ wird meist (wie gezeigt) erst durch ein Steuersignal R (= read) an diese Leseleitungen gelegt, und zwar ebenfalls durch eine verdrahtete Logik, so dass mehrere Speicher auf die gleiche Ausgangsleitung (o_μ) geschaltet werden können.

Die Kenngrößen eines RAMs sind - ebenso wie bei ROM und PROM -Anzahl der (adressierbaren) Worte · Breite der Worte (Bitzahl) (z.B. hat ein l K · 8 Bit-RAM 1024 Worte mit einer Breite von jeweils 8 Bit).

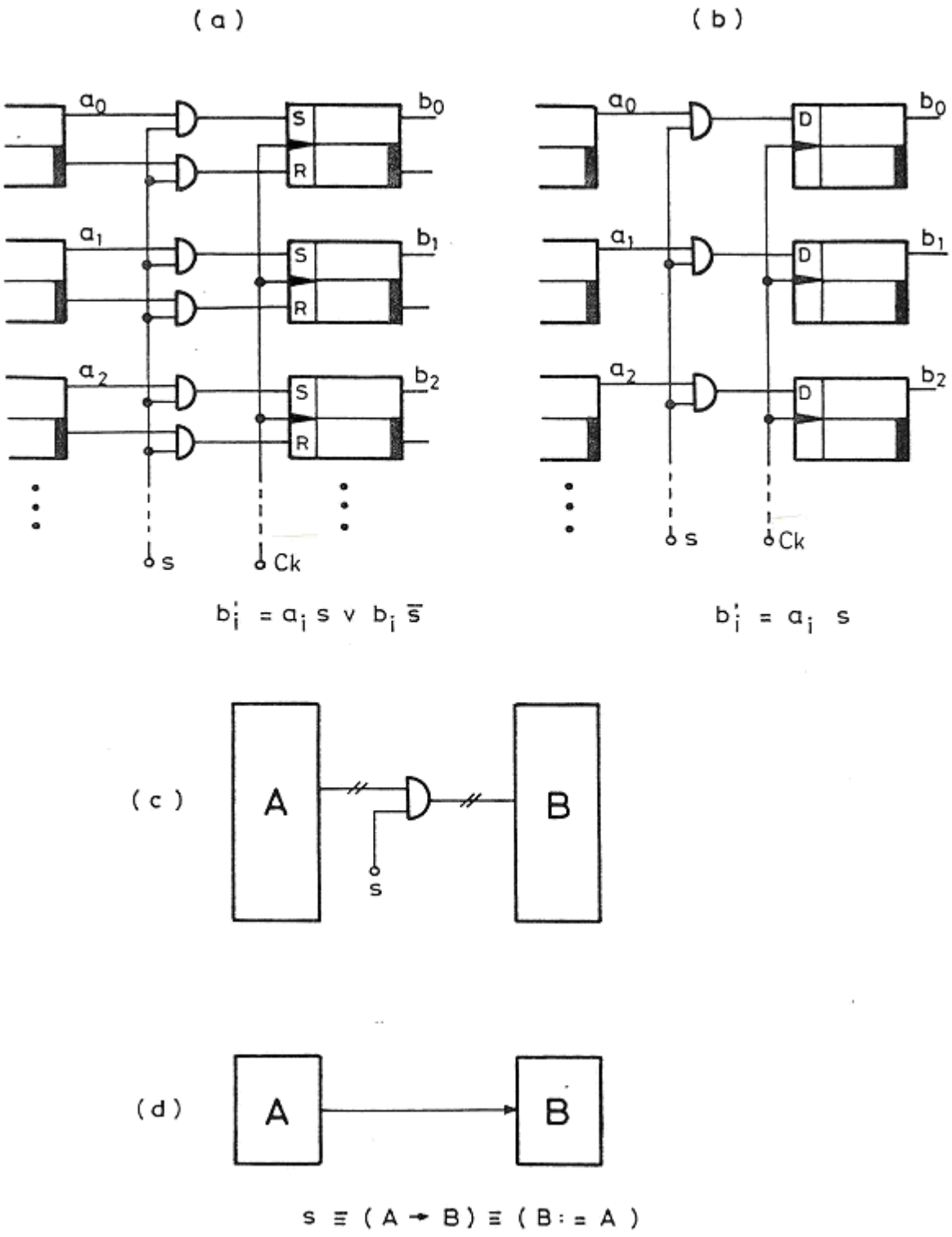


Bild 4-24: Parallele Datentransporte

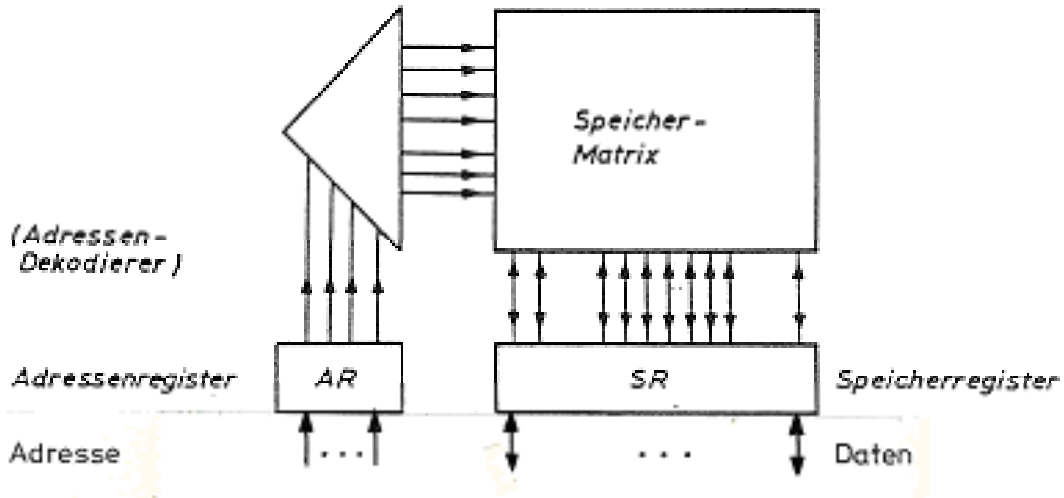


Bild 4-25: Blockschaltbild eines wortorganisierten ortsadressierten Speichers (RAM)

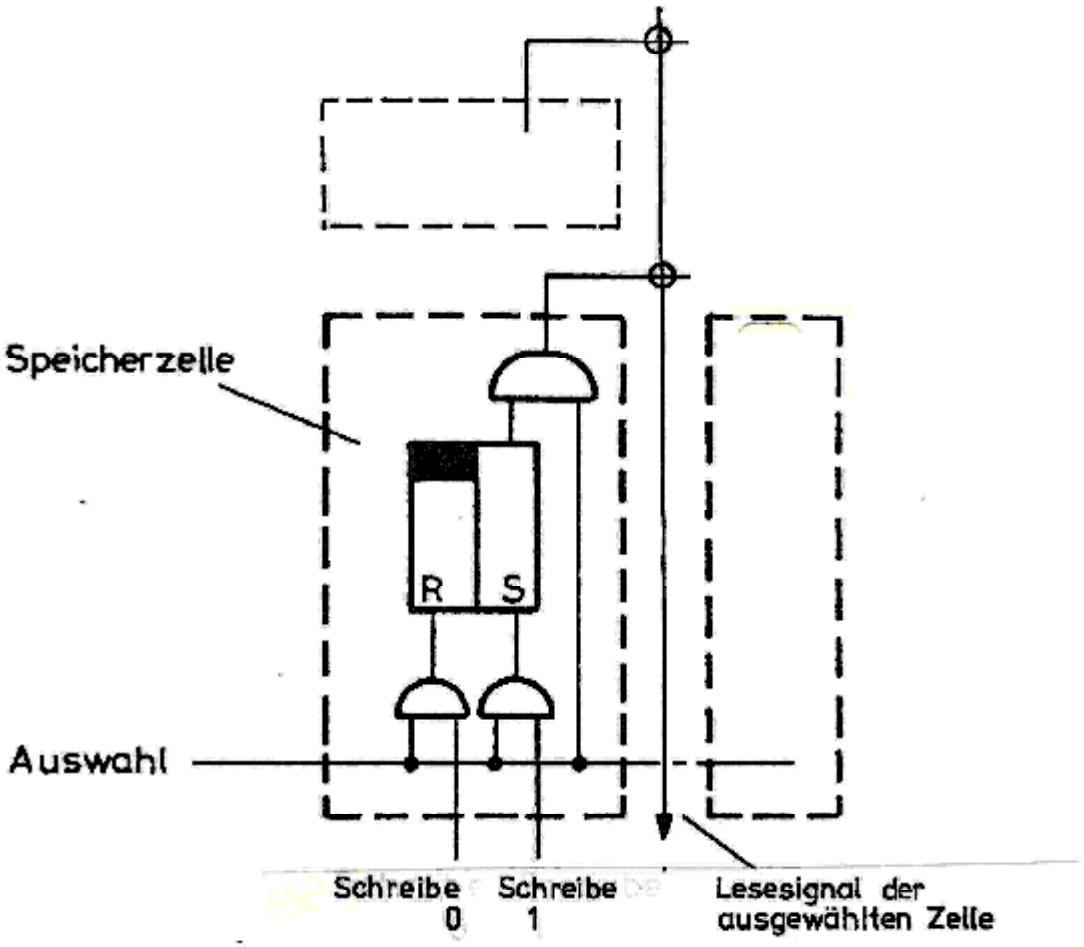


Bild 4-26: Speicherzelle

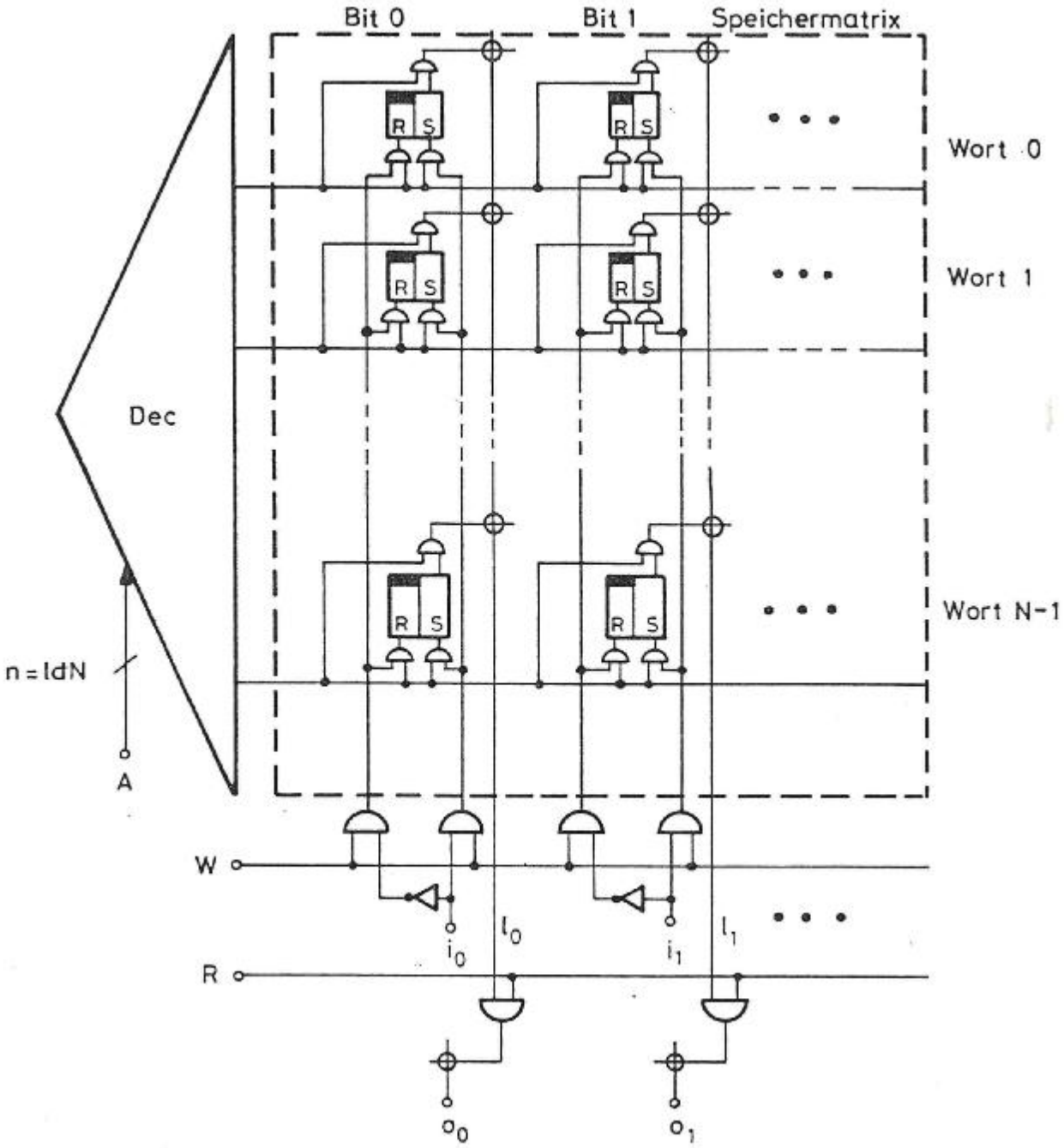


Bild 4-27: Realisierung eines wortorganisierten Speichers mit Hilfe von RS-Flipflops (RAM)

(Diese logische Darstellung eines Adressspeichers entspricht nicht dem praktischen technischen Aufbau, weil man nicht universelle Flipflops und Gatter verwendet, sondern - je nach Technologie -spezielle Speicherelemente und Verknüpfungsschaltungen, die wesentlich günstigere technische Eigenschaften erreichen lassen, z.B. einige 100 MBit pro "Chip" und 10 ns Zugriffszeit.)

4.5.3 Festwertspeicher (ROM)

Eine andere, häufig gebrauchte Gruppe der Halbleiterspeicher bilden die ROMs (Read Only Memory) , die in den unterschiedlichsten Varianten auf dem Markt zu finden sind (Bild 6.10).

ROM:	Maskenprogrammiert
PROM:	Elektrisch programmierbar
EPROM:	Elektrisch programmier- und durch UV-Licht löschbar
EEPROM:	Elektrisch programmier- und löschbar
Flash-PROM	Elektrisch blockweise löschbar und programmierbar

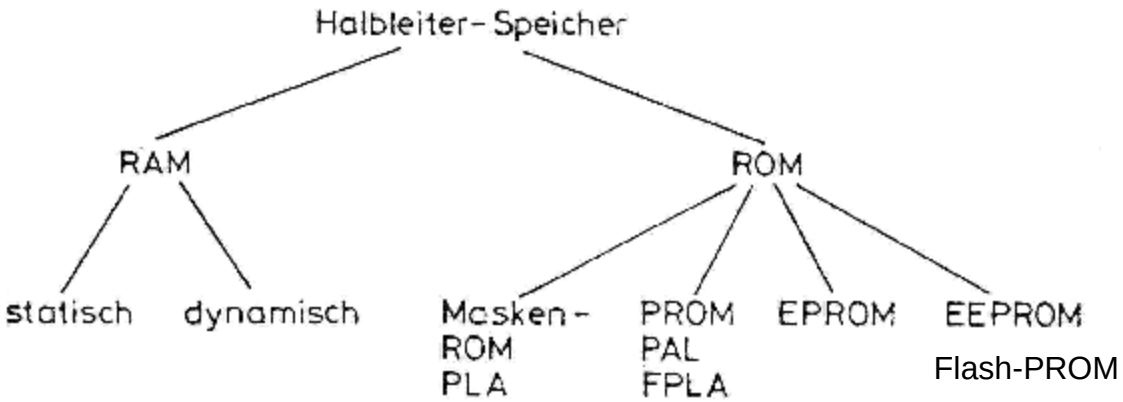


Bild 4-28: Übersicht über gebräuchliche Halbleiterspeicher

Ein ROM ist ein Festwertspeicher, dessen Informationsinhalt nach der Herstellung nicht mehr verändert werden kann. ROMs werden durch entsprechende Masken in der Fabrik "programmiert". Ein PROM (Programmable Read Only Memory) kann beim Anwender durch Ausbrennen von Verbindungen ("Sicherungen" = Fuses) codiert werden. Generelle Funktion solcher Festwertspeicher ist die eines einseitigen Zuordners: Einem Eingangsvektor (Adresse) sind fest gespeicherte Ausgangsgrößen (Daten) zugeordnet (Bild 4-29). Die Adressen (n Bits) aktivieren über einen "Decoder" die zugehörigen Wortleitungen w v der Speichermatrix. Der Inhalt der Speicherzellen kann dann über die Speicherleitungen b 11 und die Leseverstärker ausgegeben werden. Die Kenngrößen eines (P)ROMs sind Wortanzahl (2^n) · Anzahl der Bits pro Wort. Wenn man in der DNF-Version eines Schaltnetzes alle Vollkonjunktionen genau einmal realisiert, erhält man die in Bild 4-30 dargestellte (P)ROM-Version.

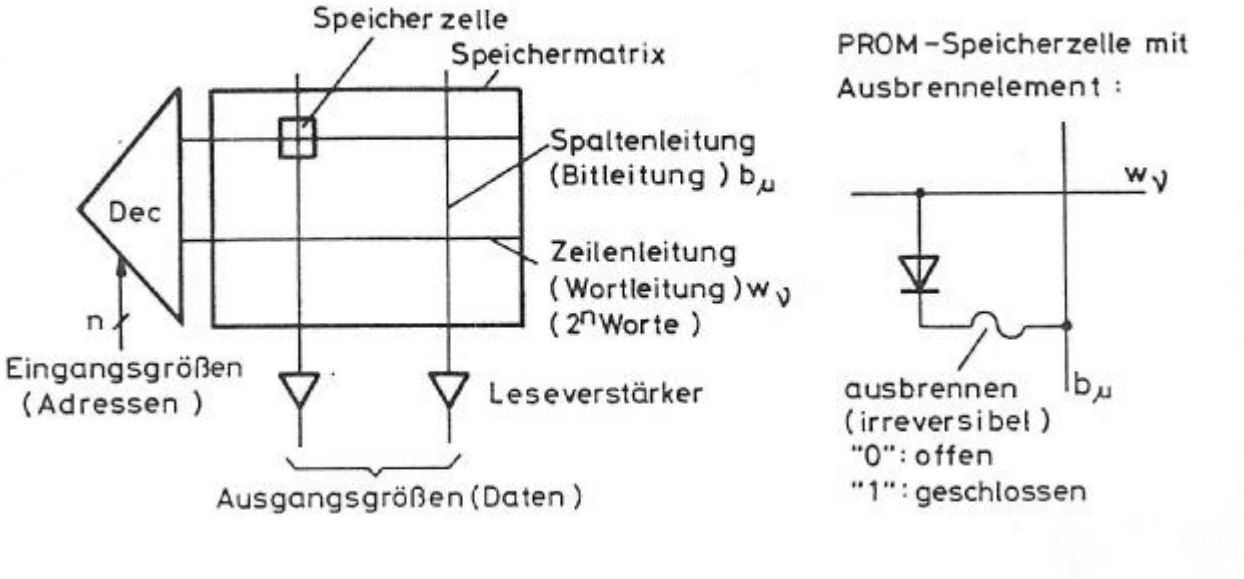


Bild 4-29: Prinzipieller Aufbau eines (P)ROM

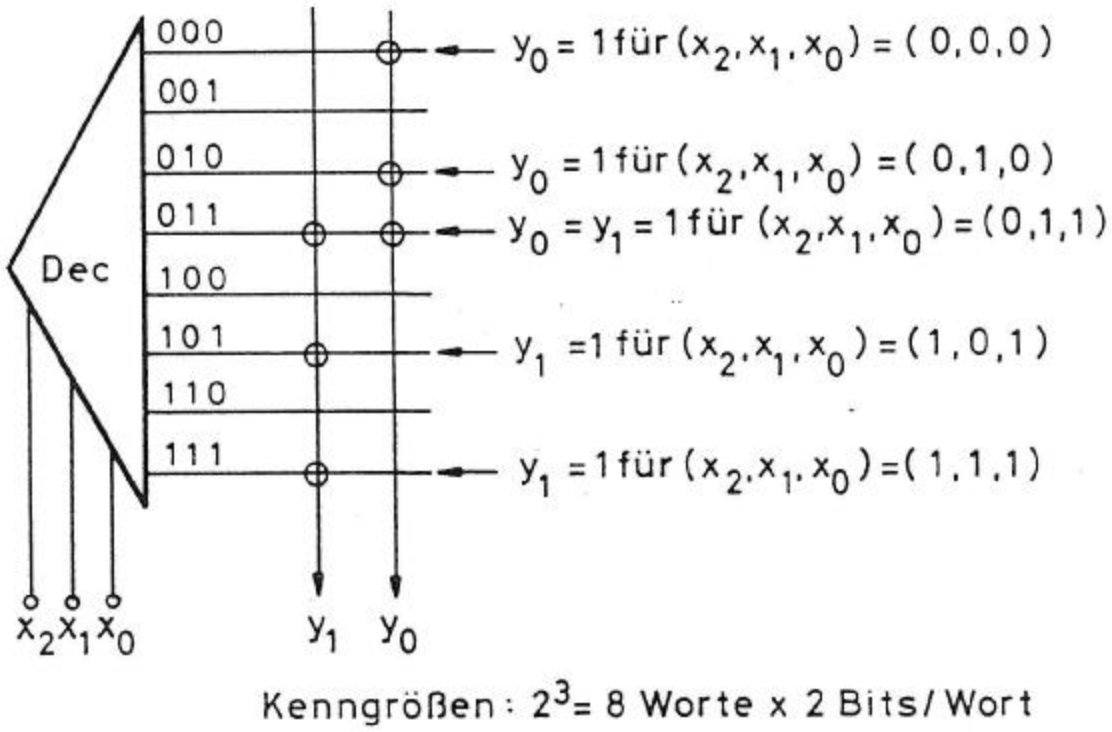


Bild 4-30: (P)ROM-Realisierung eines Schaltnetzes

4.5.4 FIFO-Speicher

Ein FIFO-Speicher (FIFO = First In First Out) ist eine Art asynchrones Schieberegister, d.h. die Daten erscheinen am Ausgang in derselben Reihenfolge, wie sie eingegeben wurden: das zuerst eingelesene Wort (First In) wird auch wieder zuerst ausgelesen (First Out). Im Unterschied zu einem Schieberegister kann dieser Vorgang jedoch völlig asynchron erfolgen, d.h. der Auslesetakt ist unabhängig vom Einlesetakt. Deshalb eignen sich FIFOs vor allem zur Kopplung von Systemen mit unterschiedlicher Taktung.

Die Funktion ist ähnlich der einer Warteschlange: Die Daten wandern nicht mit einem festen Takt vom Eingang zum Ausgang, sondern warten nur so lange im Register, bis alle vorhergehenden Daten ausgegeben sind.

In der Regel werden drei Steuersignale zur Verfügung gestellt: Speicher voll, Speicher leer, und Speicher halbvoll. Ist der Speicher leer, darf nicht mehr ausgelesen werden. Ist er voll, darf nichts mehr eingeschrieben werden. Das letzte Signal "halbvoll" kann dazu benutzt werden, die Datenraten rechtzeitig zu verändern, bevor ein Überlauf oder Leerlauf eintritt. Extreme sind dann vermeidbar, wenn die mittleren Datenraten für Lesen und Schreiben gleich sind. Dann kann das FIFO kurzzeitige Schwankungen auffangen, wenn seine Speicherkapazität hinreichend groß bemessen ist.

Für die Realisierung eines FIFO gibt es verschiedene Möglichkeiten, z.B. nach dem Durchlaufprinzip (Schiebregister), mit Zweitorspeicher (Dual Port RAM) und mit Standard-RAMs und zusätzlicher Steuerlogik.

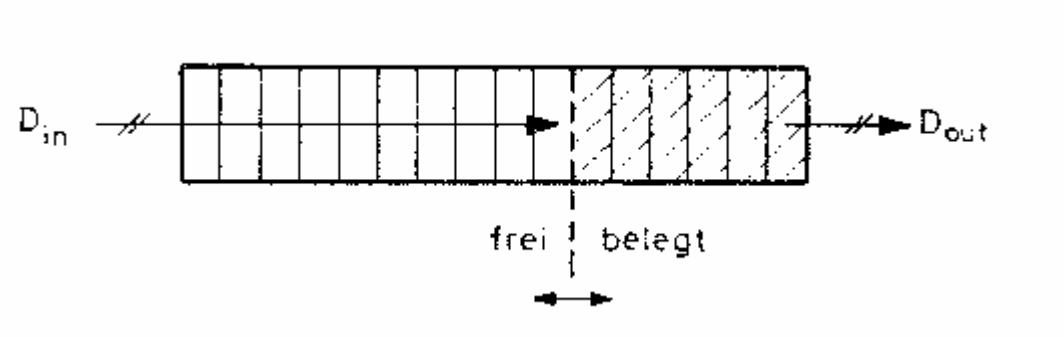


Bild 4-31: Prinzip des FIFO-Speichers

FIFO-Speicher eignen sich insbesondere dazu, Schaltsysteme, die mit unterschiedlichen Taktzyklen betrieben werden, miteinander zu koppeln. Die „lose“ Kopplung mit einem FIFO erlaubt über die Datenblockgröße des FIFO´s eine Vergrößerung der Reaktionszeiten in der Datenübertragung zwischen den Schaltsystemen.