Technische Informatik 2
- Grundlagen der Computertechnik -

Prof. Dr. Wolfram Schiffmann

1. Komplexe Schaltwerke

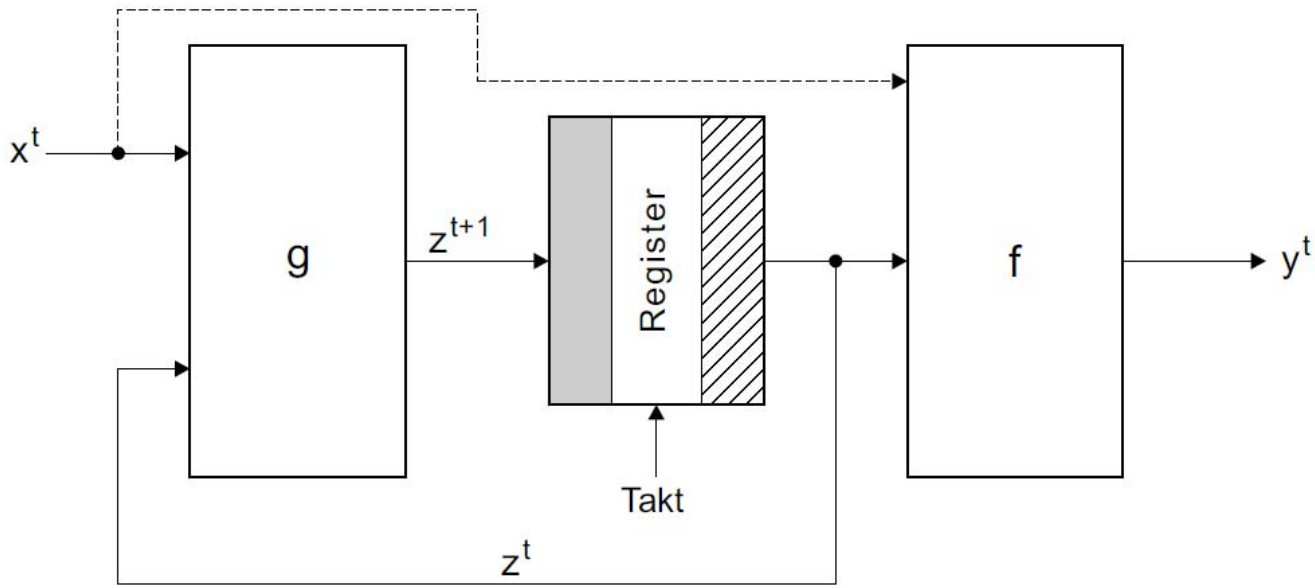


Abb. 1.1. Aufbau eines MEALY–Schaltwerks. Wenn die gestrichelte Verbindung nicht vorhanden ist, erhalten wir ein MOORE–Schaltwerk.

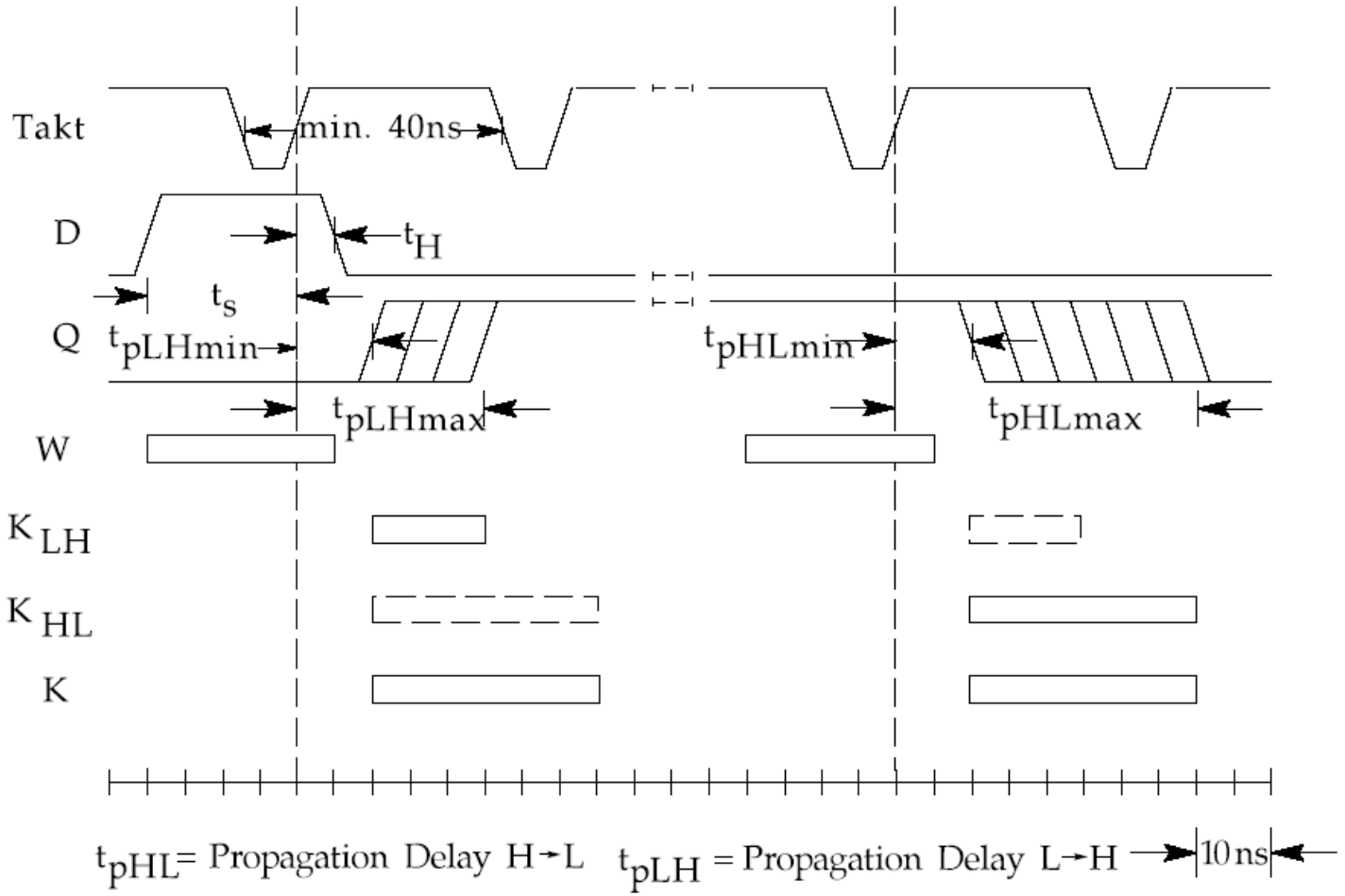


Abb. 1.2. Definition von Wirk– und Kippintervall bei dem taktflankengesteuerten *D*–Flipflop SN 7474

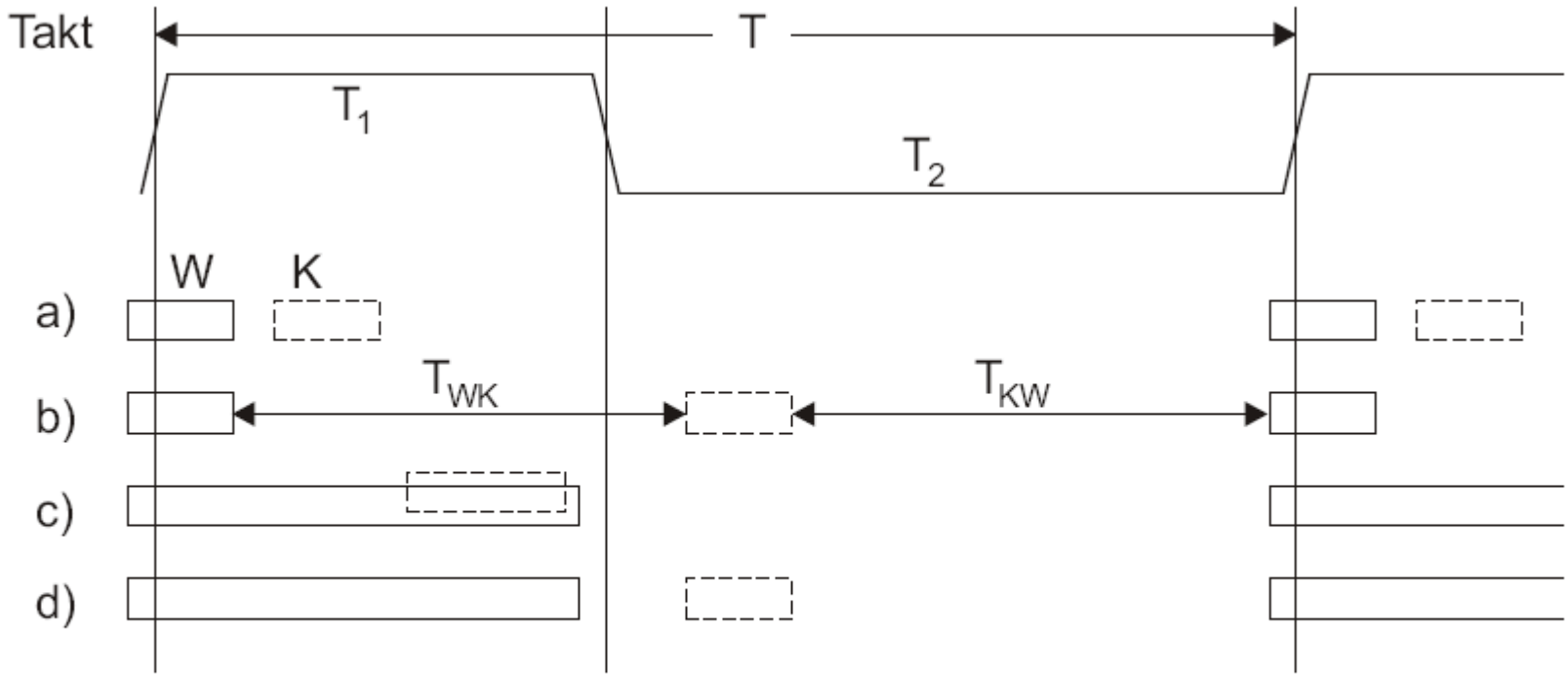


Abb. 1.3. Lage von Wirk– und Kippintervall bei verschiedenen Flipflop–Typen a) einflankengesteuert b) zweiflankengesteuert (Master–Slave) c) taktzustandsgesteuert (Latch) d) taktzustandsgesteuert (Master–Slave)

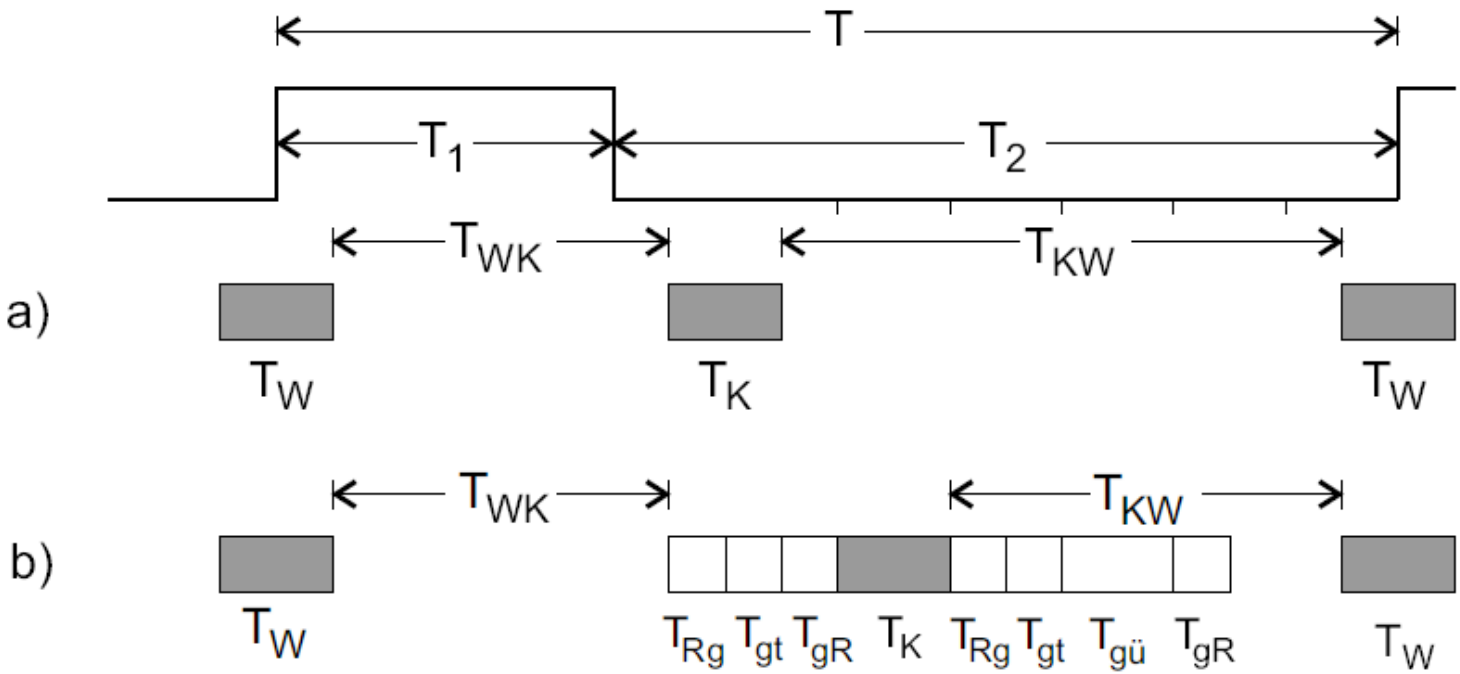


Abb. 1.4. Zur Herleitung der Rückkopplungsbedingungen. Zeitverhalten a) *ohne* b) *mit* Tot- und Übergangszeiten.

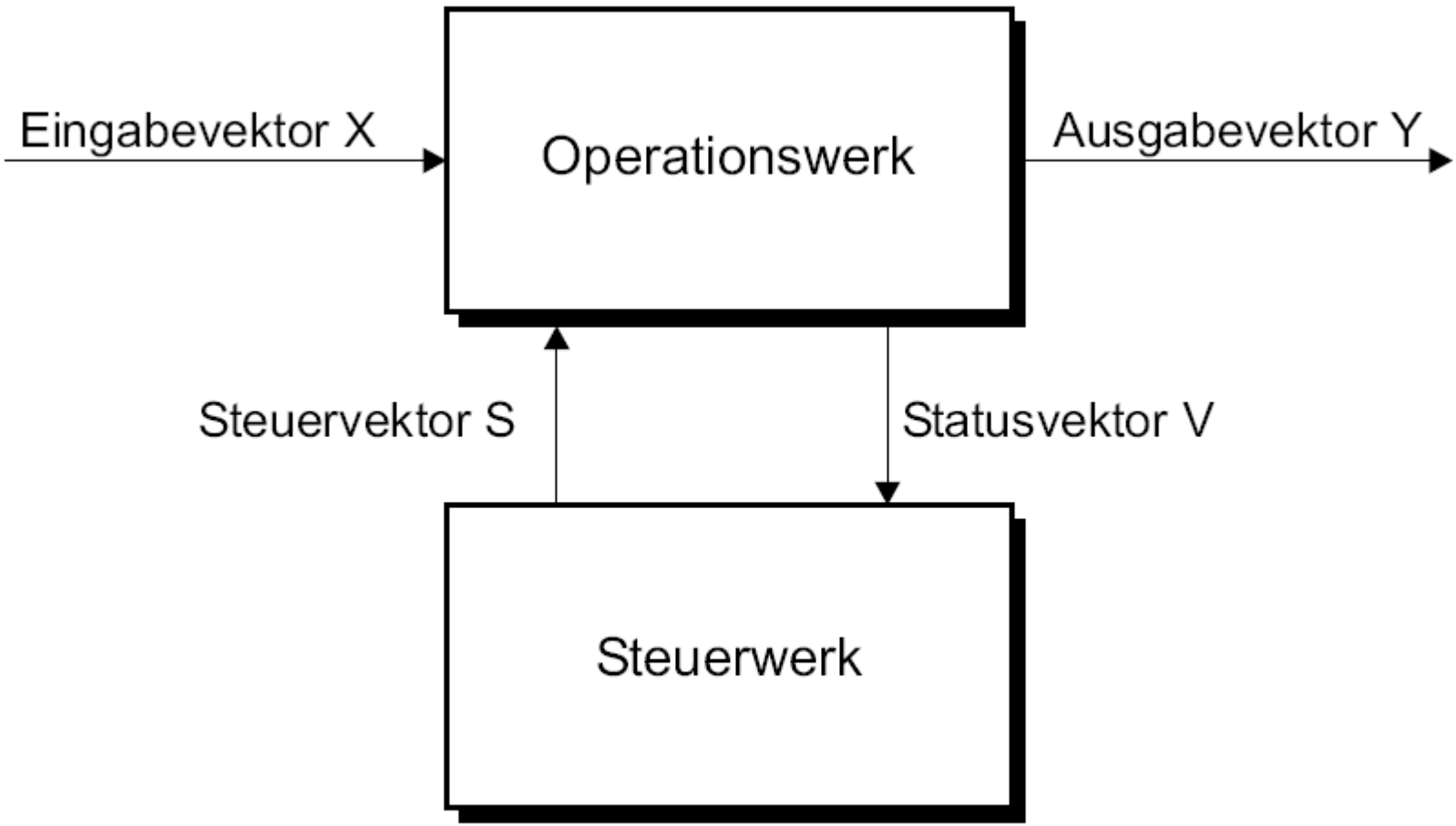


Abb. 1.5. Aufbau eines komplexen Schaltwerks

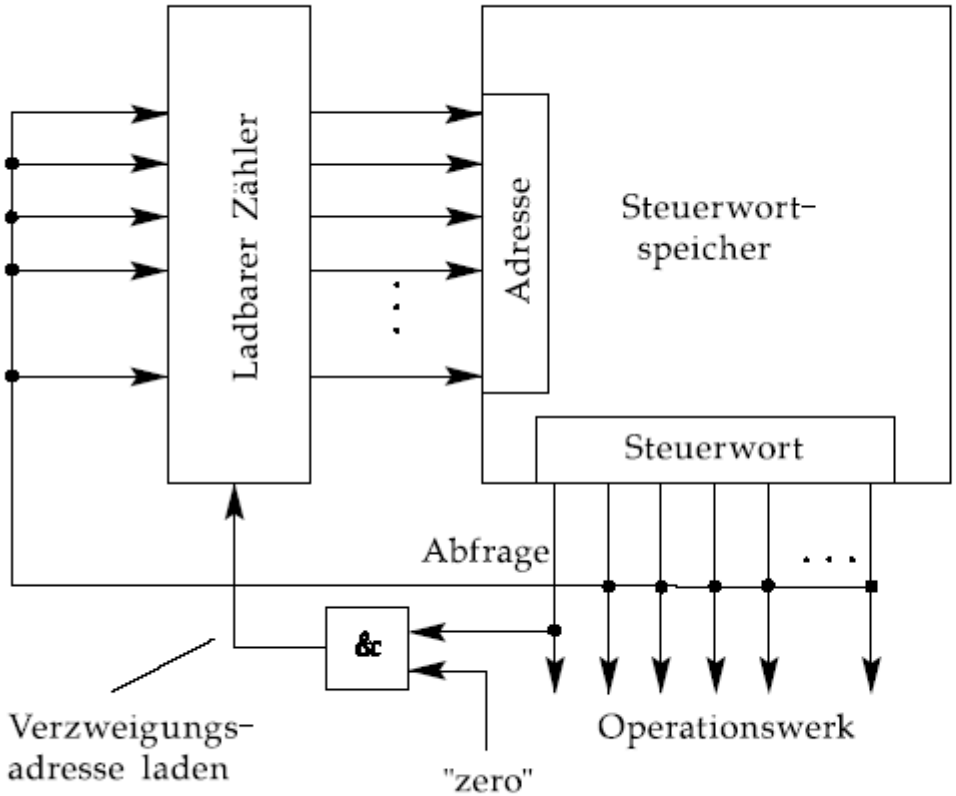
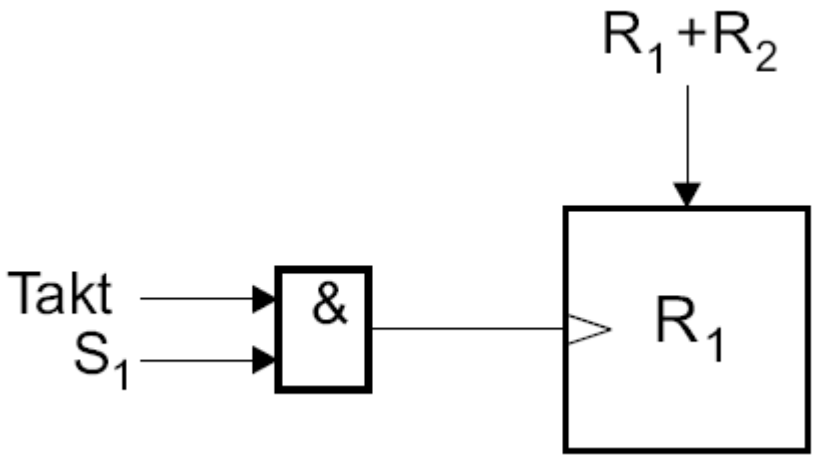


Abb. 1.6. Prinzip eines mikroprogrammierbaren Steuerwerks, das bedingte Verzweigungen ausführen kann.



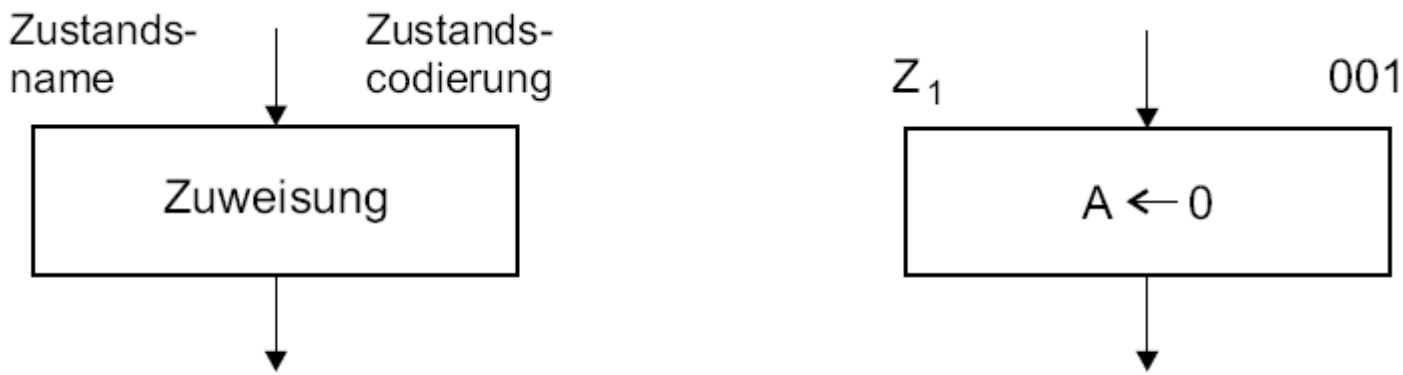


Abb. 1.7. Zustandsbox: links allgemeine Form, rechts Beispiel

Abb. 1.8. Entscheidungsbox: links allgemeine Form, rechts Beispiel

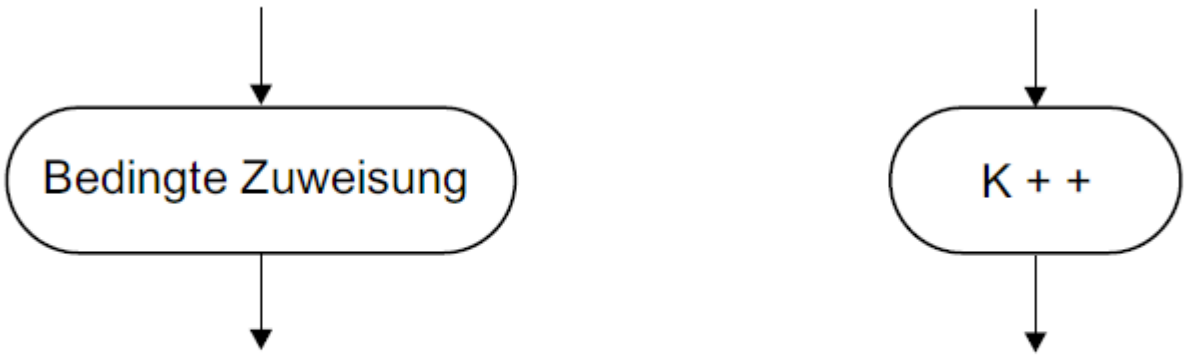
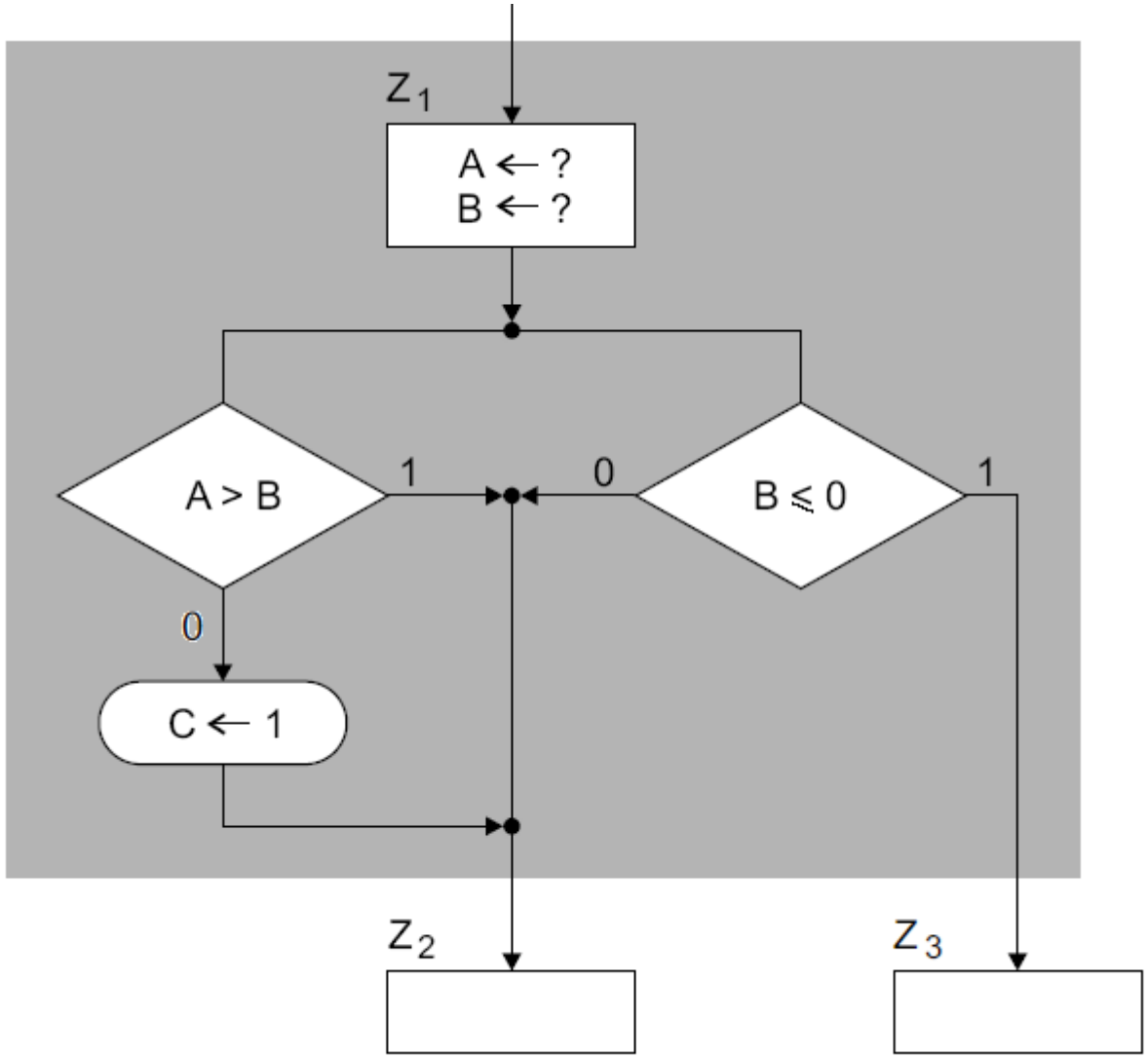


Abb. 1.9. Bedingte Ausgangsbox: rechts allgemeine Form, links Beispiel



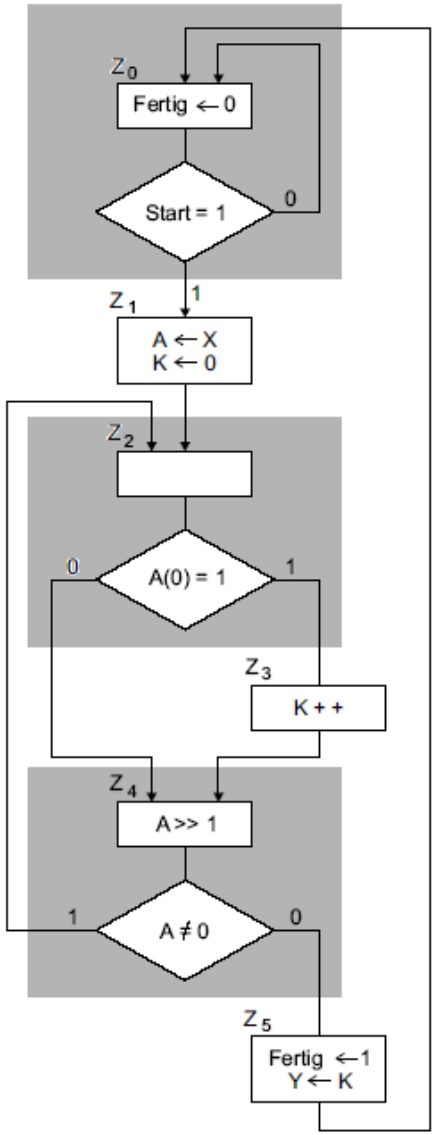


Abb. 1.10. ASM–Diagramm für ein komplexes MOORE–Schaltwerk

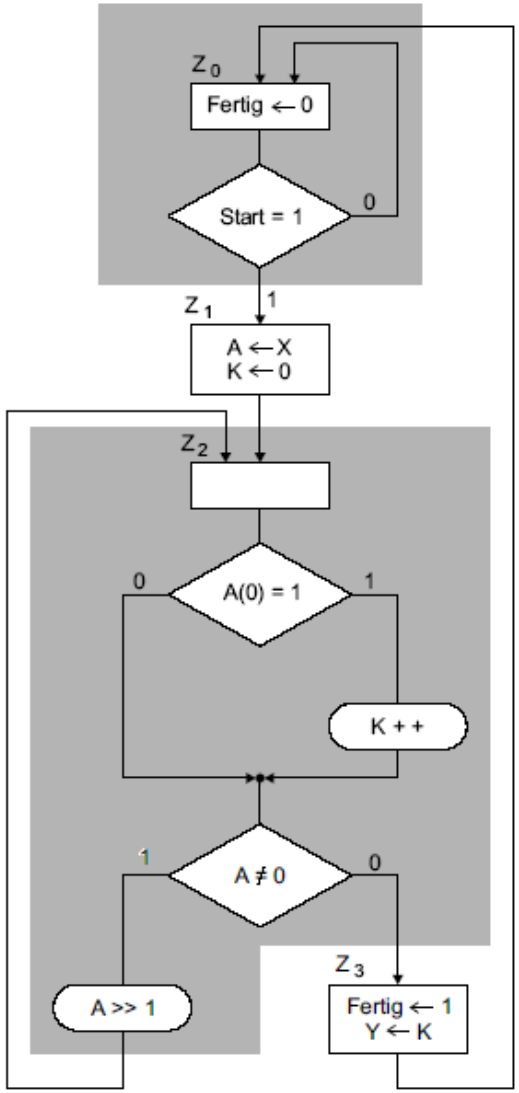


Abb. 1.11. ASM–Diagramm für ein komplexes MEALY–Schaltwerk

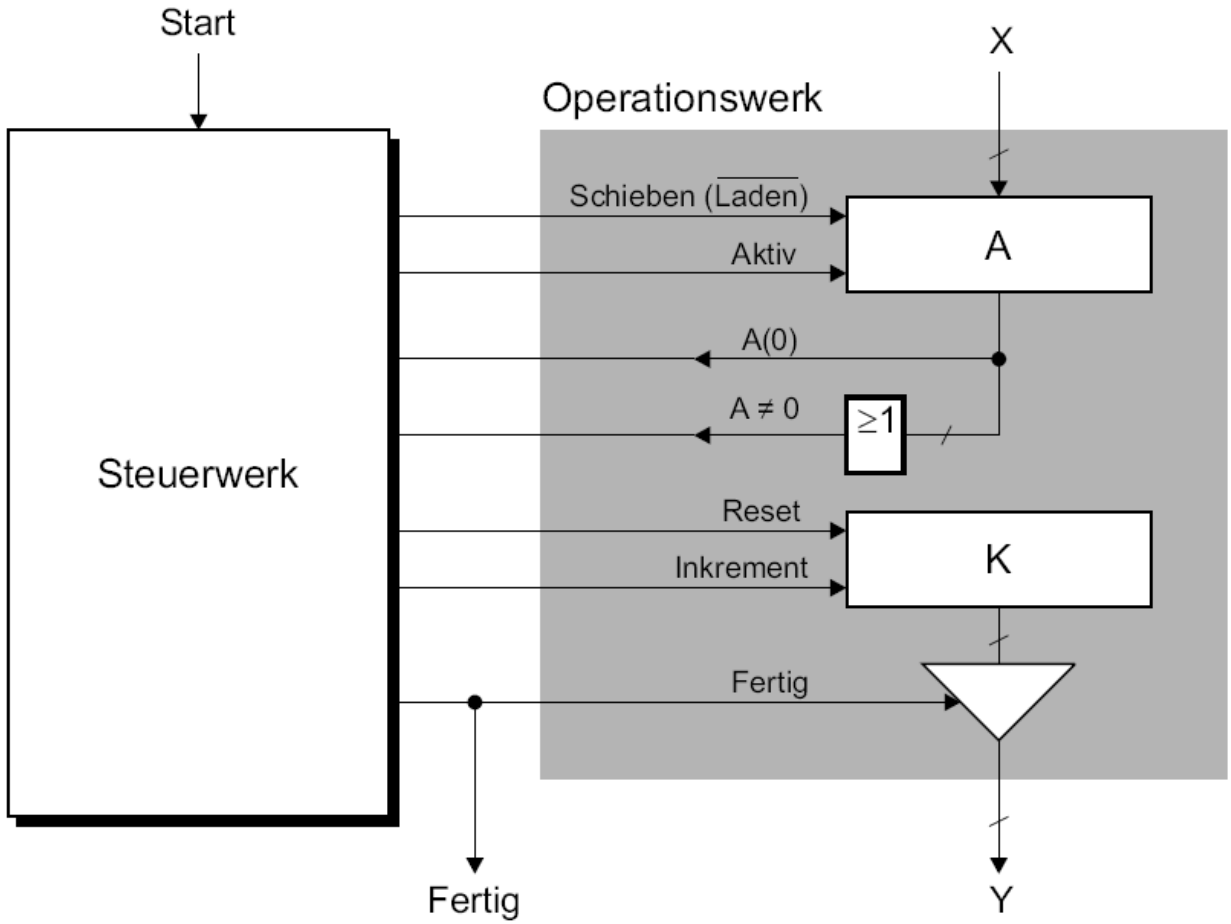


Abb. 1.12. Aufbau des Operationswerks für den Einsen–Zähler

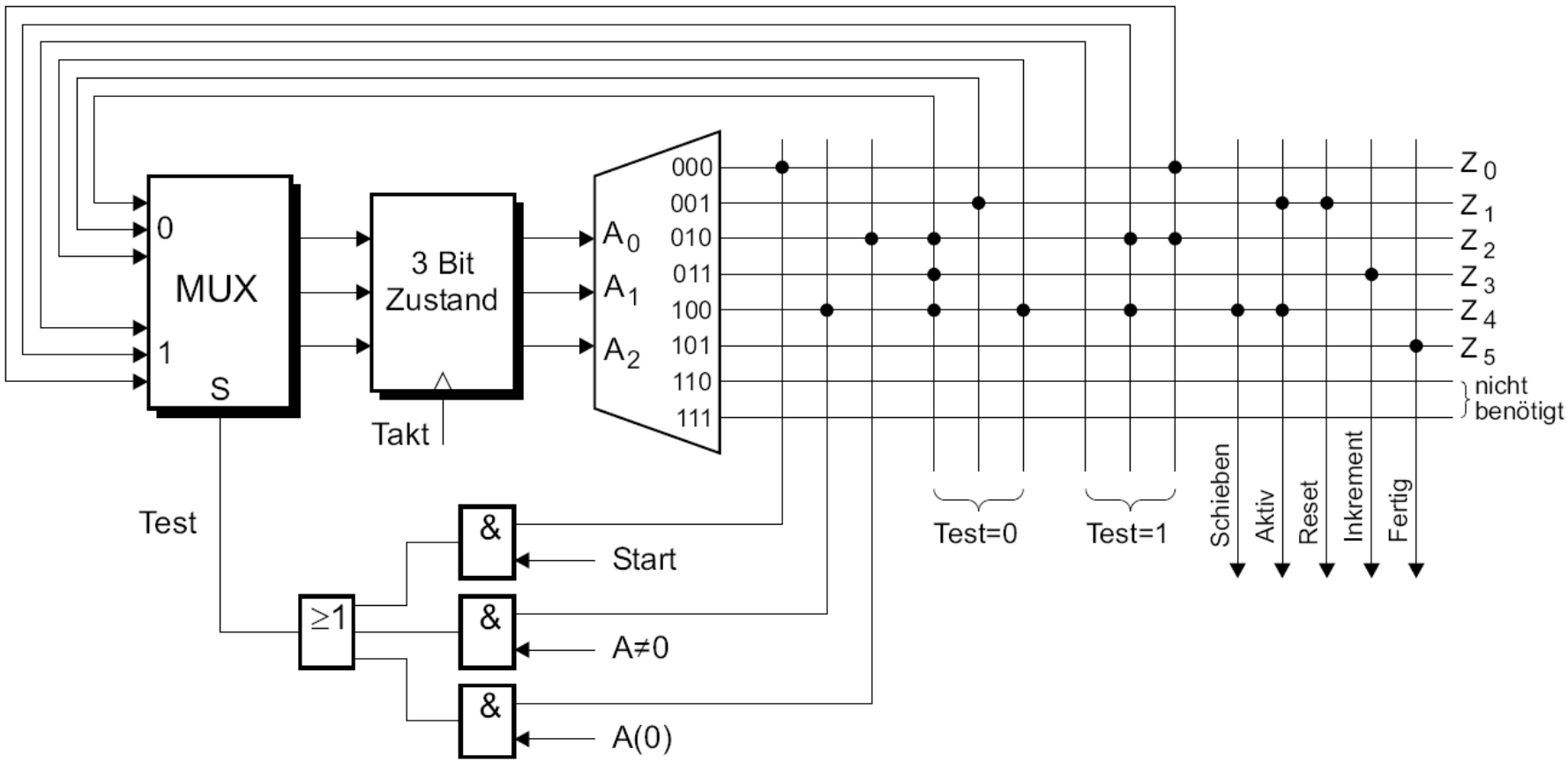


Abb. 1.13. Mikroprogrammsteuerwerk für die MOORE–Variante

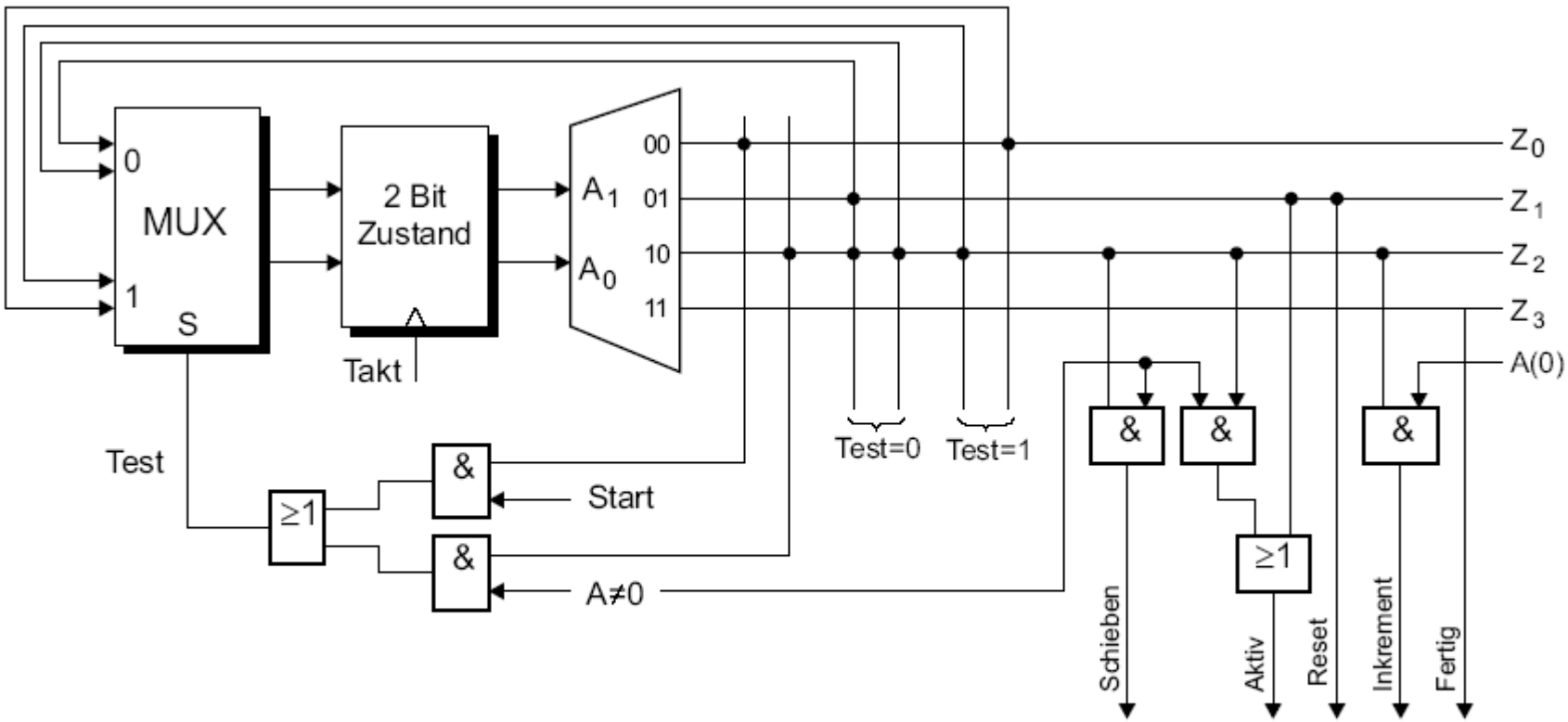


Abb. 1.14. Mikroprogrammsteuerwerk für die MEALY–Variante

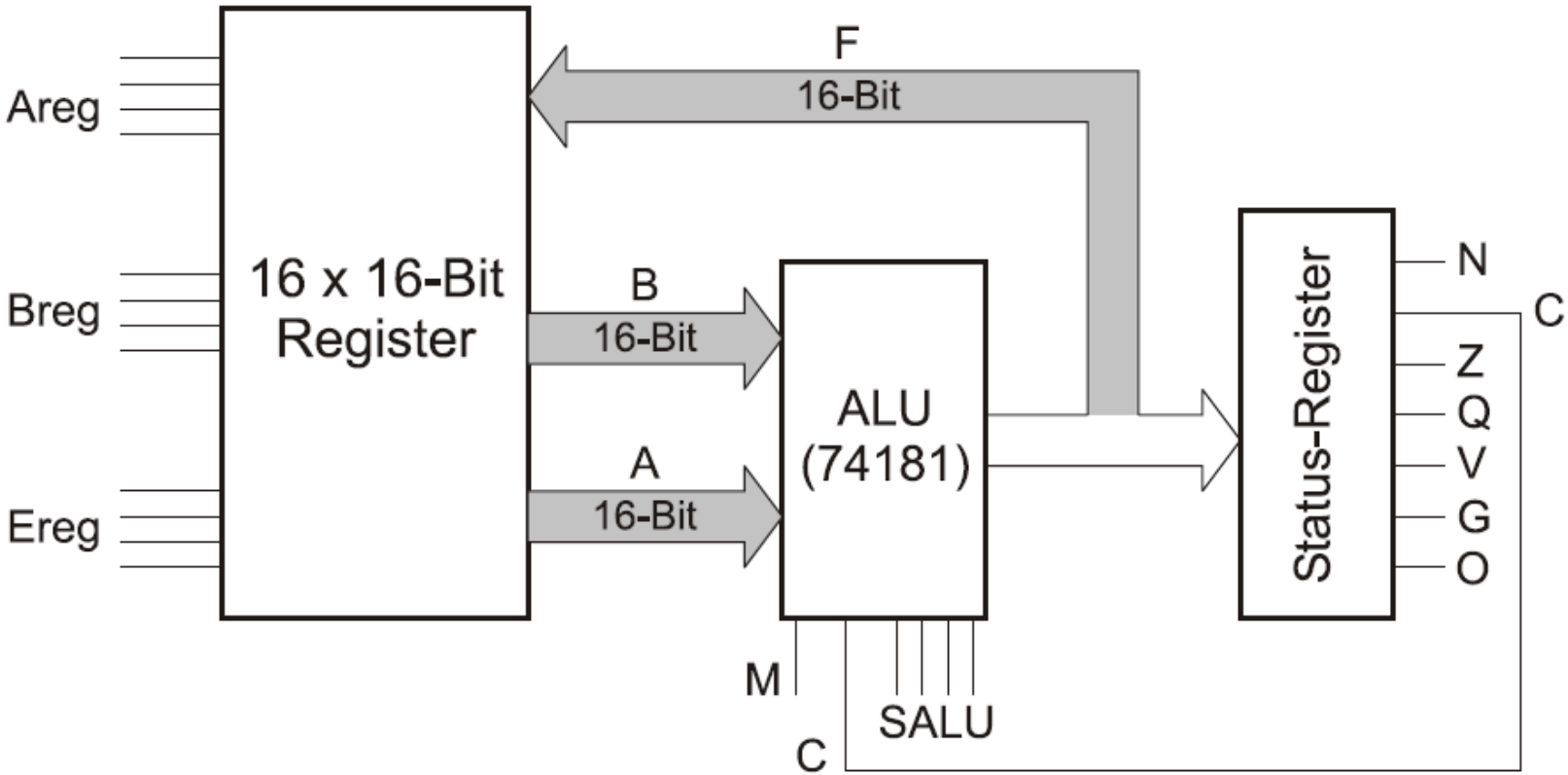


Abb. 1.15. Aufbau einer RALU mit zwei Leseports und einem Schreibport.

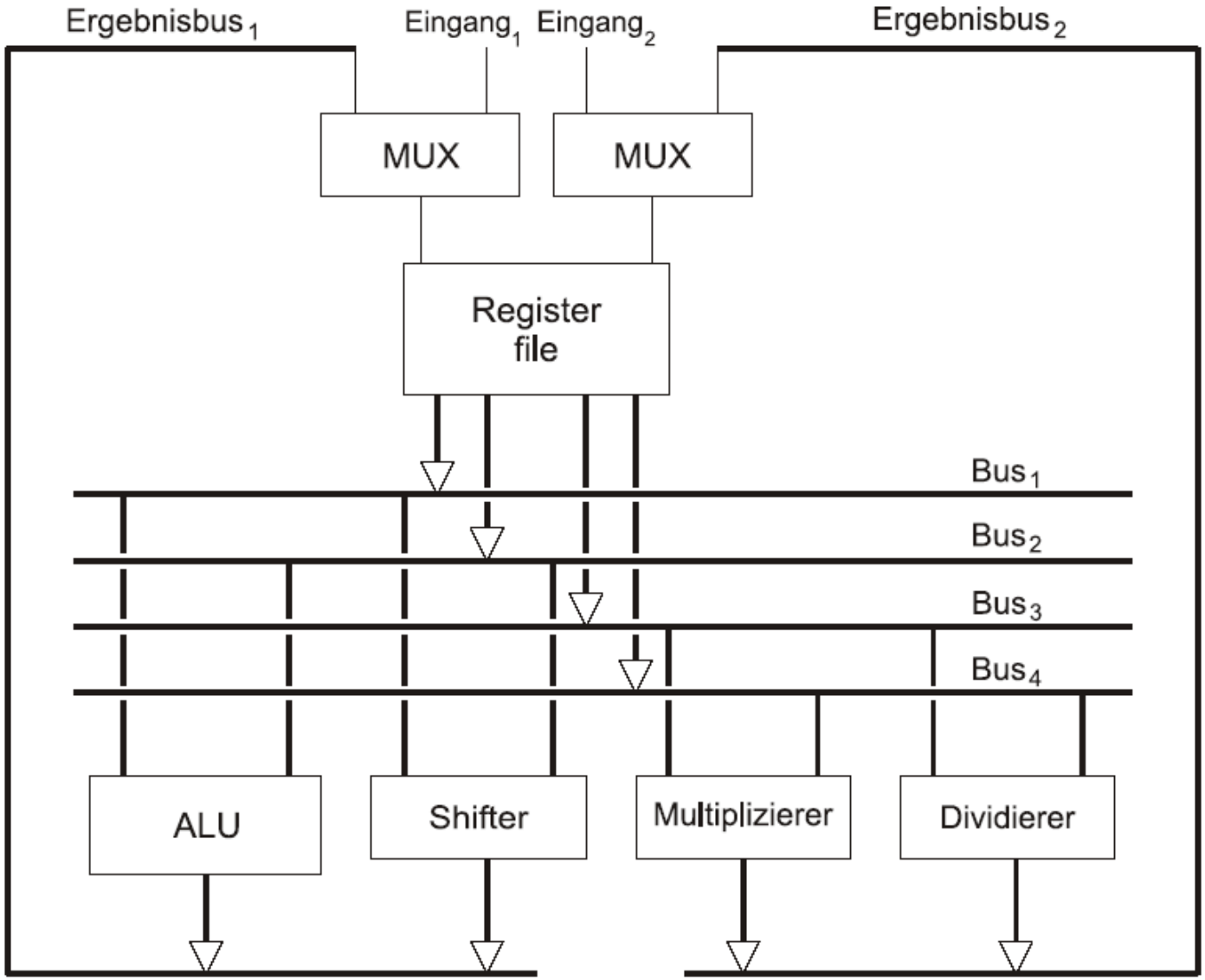


Abb. 1.16. Aufbau eines Rechenwerks mit zwei parallel nutzbaren Datenpfaden

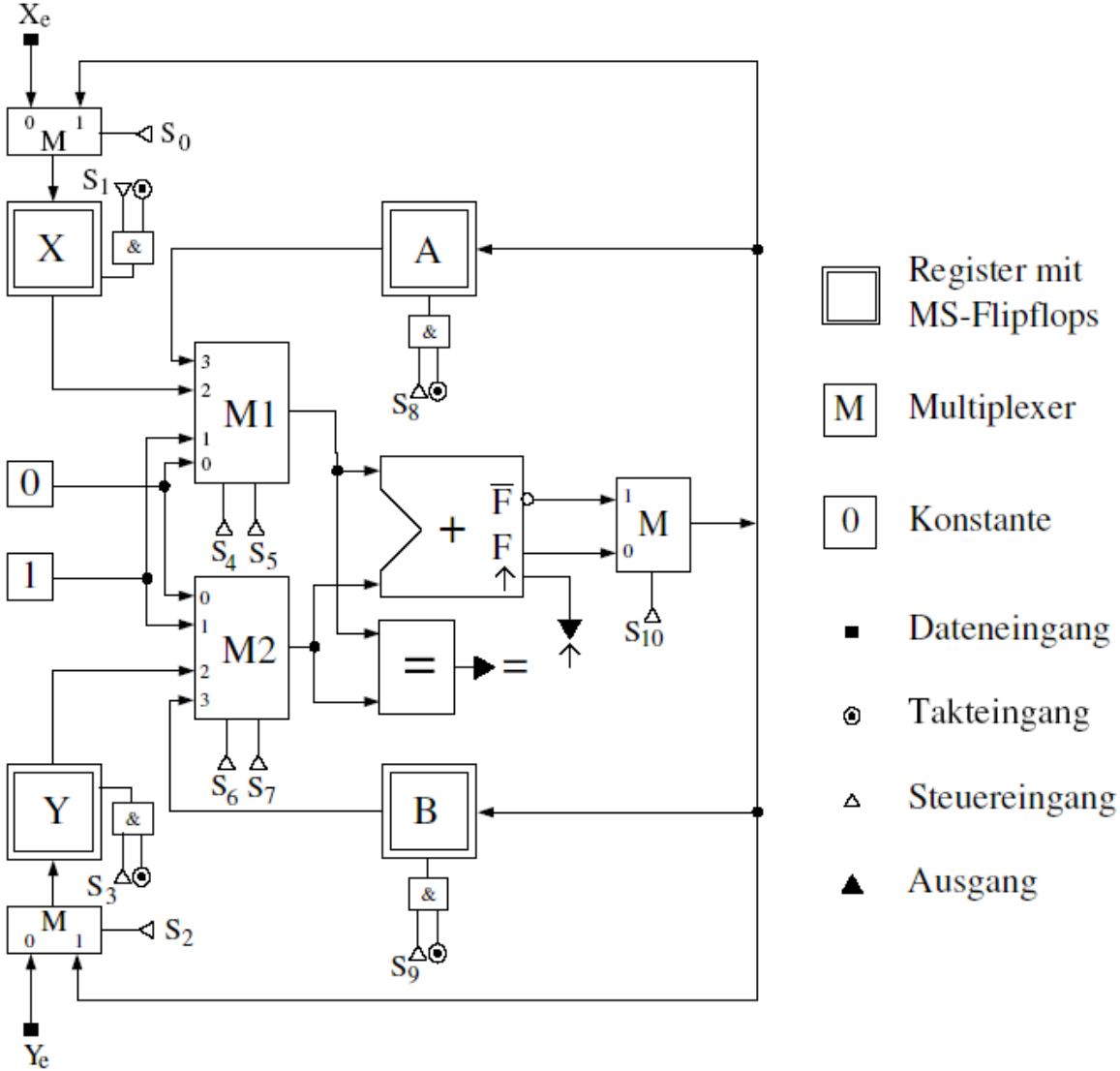


Abb. 1.17. Blockschaltbild des simulierten Operationswerks

2. von NEUMANN-Rechner

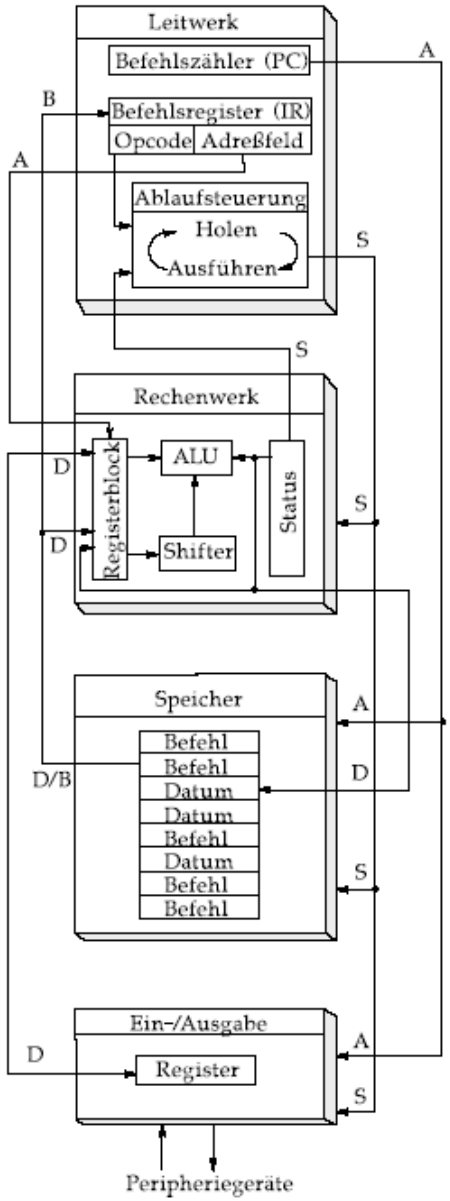


Abb. 2.1. Blockschaltbild eines von NEUMANN–Rechners

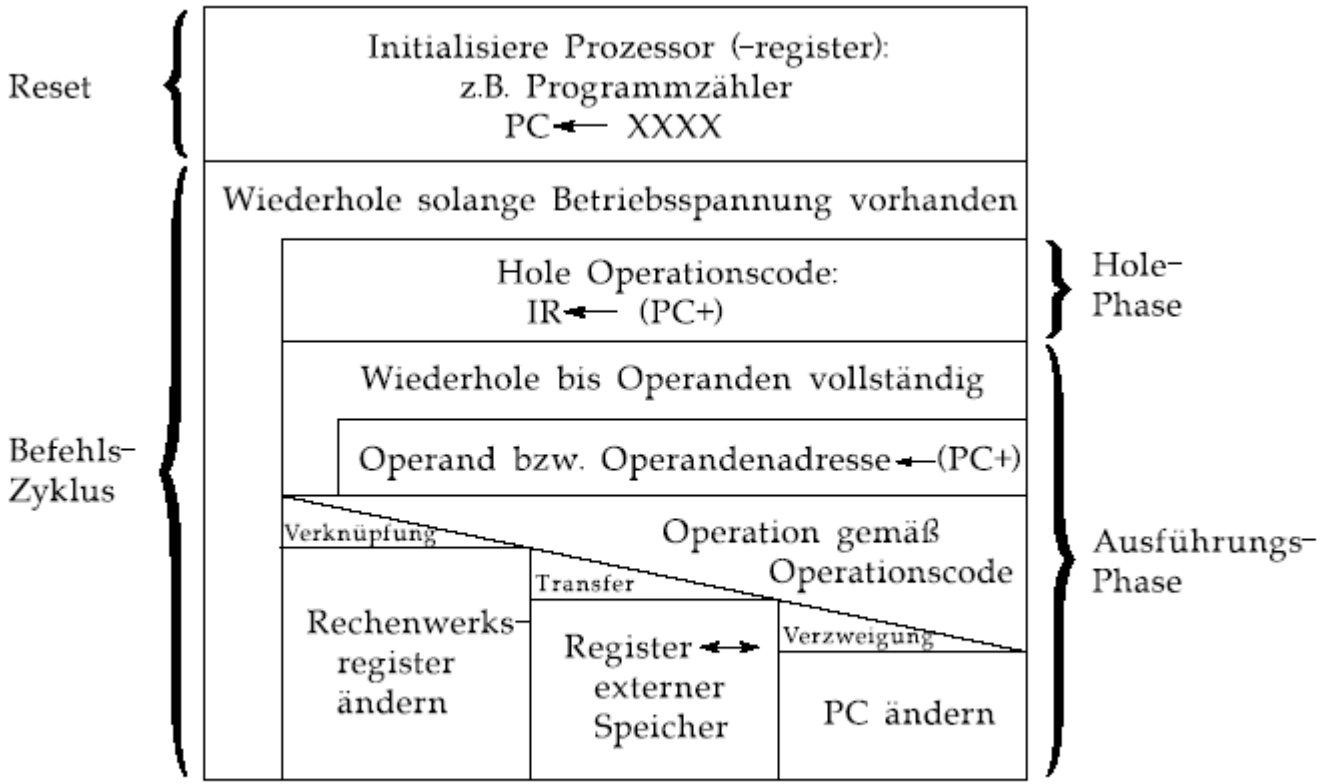


Abb. 2.2. Befehlsabarbeitung beim von NEUMANN–Rechner

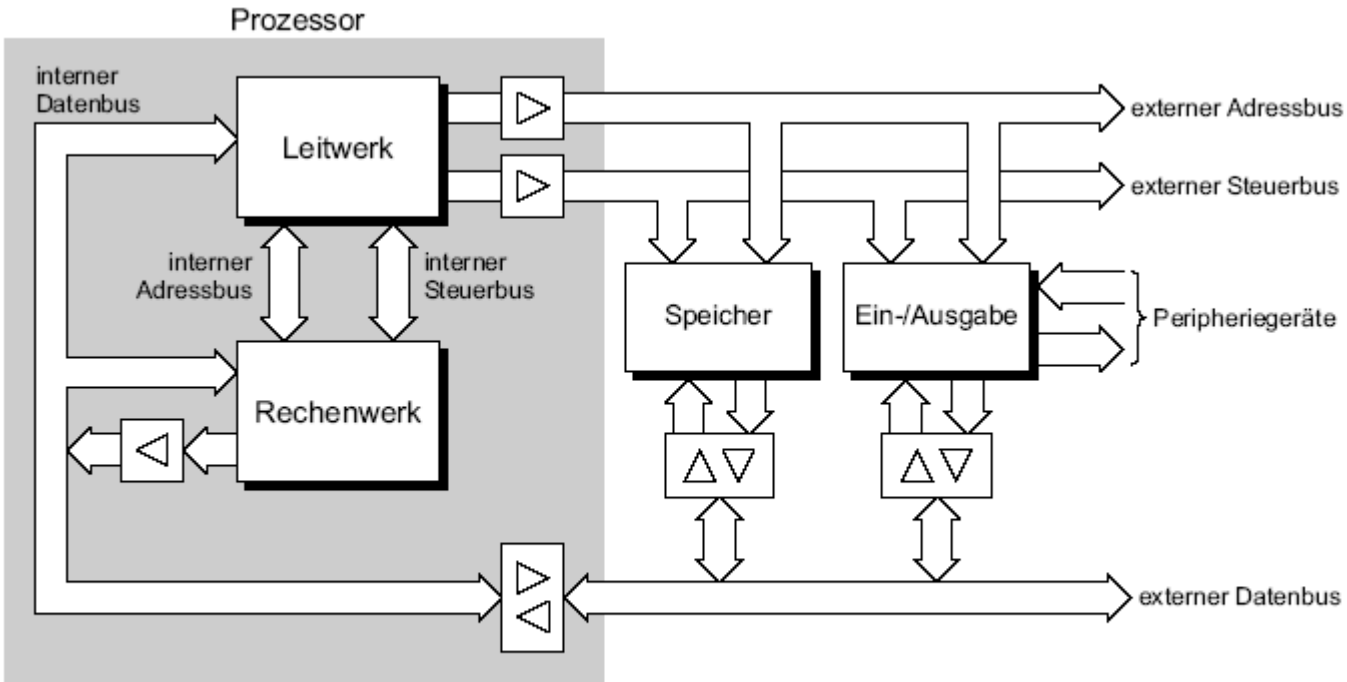


Abb. 2.3. Interne und externe Busse erleichtern die Realisierung eines Prozessors

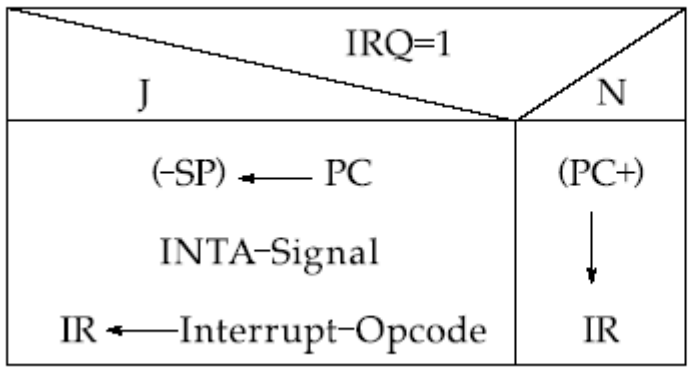


Abb. 2.4. Erweiterung der Holephase zur Bearbeitung eines Interrupts

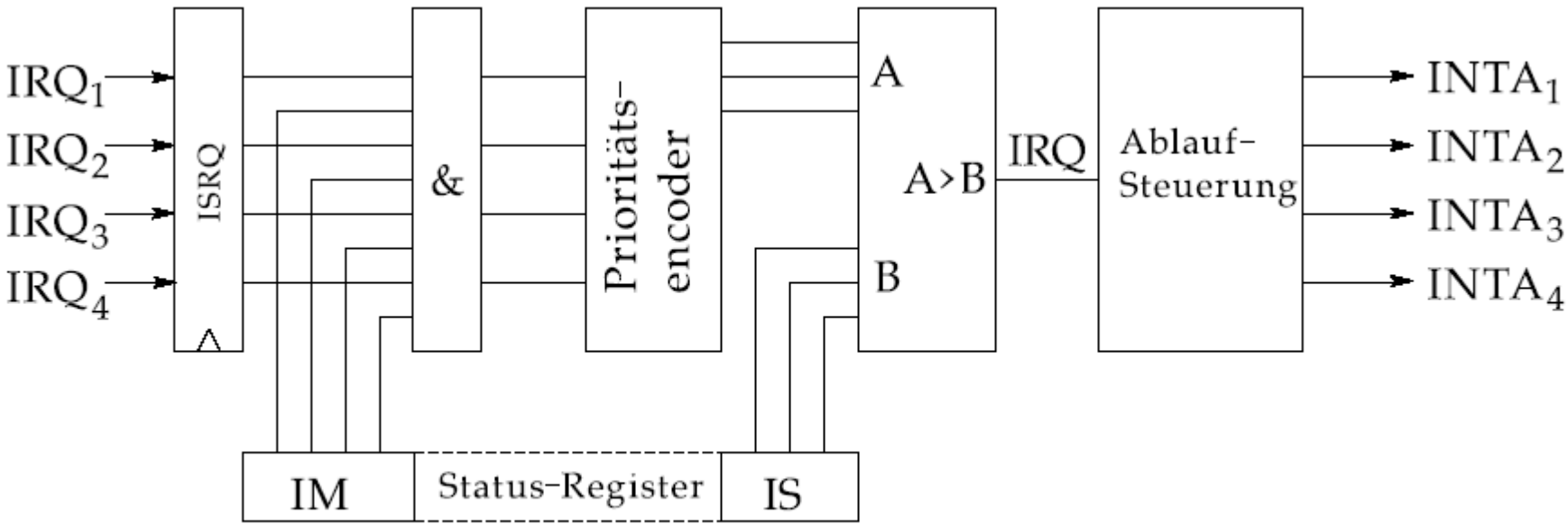


Abb. 2.5. Aufbau eines Interrupt–Controllers für 4 Prioritätsebenen

IRQ_4	IRQ_3	IRQ_2	IRQ_1	Code
1	X	X	X	100
0	1	X	X	011
0	0	1	X	010
0	0	0	1	001
0	0	0	0	000

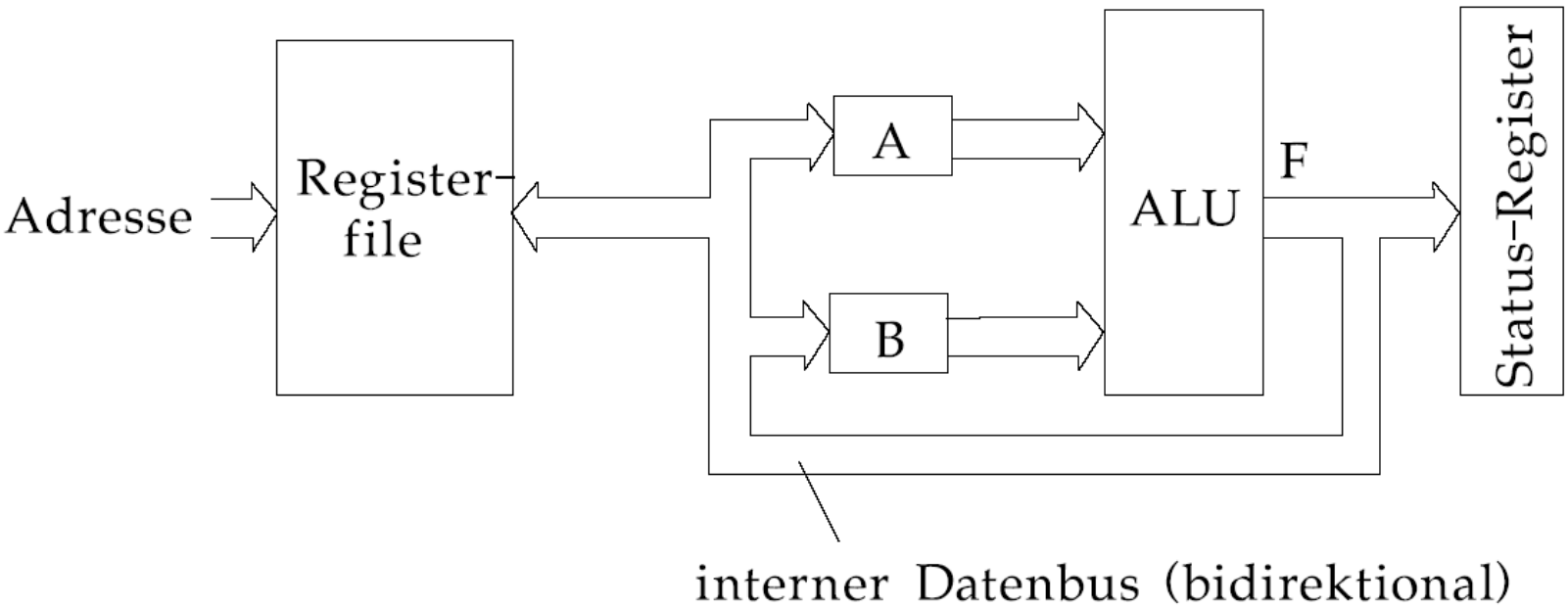


Abb. 2.6. Rechenwerk einer Ein–Adress Maschine

Taktzyklus	Operation
1	$R1 \rightarrow A$
2	$R2 \rightarrow B$
3	$F \rightarrow R3$

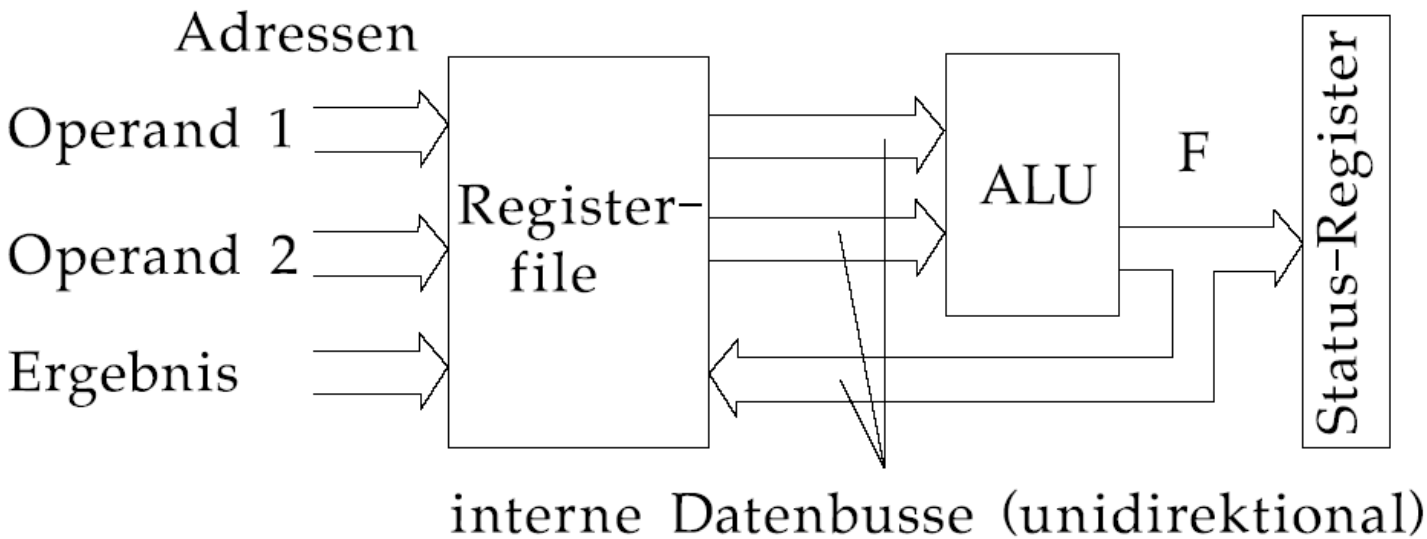


Abb. 2.7. Rechenwerk einer Drei–Adress Maschine (z.B. RISC–Prozessor)

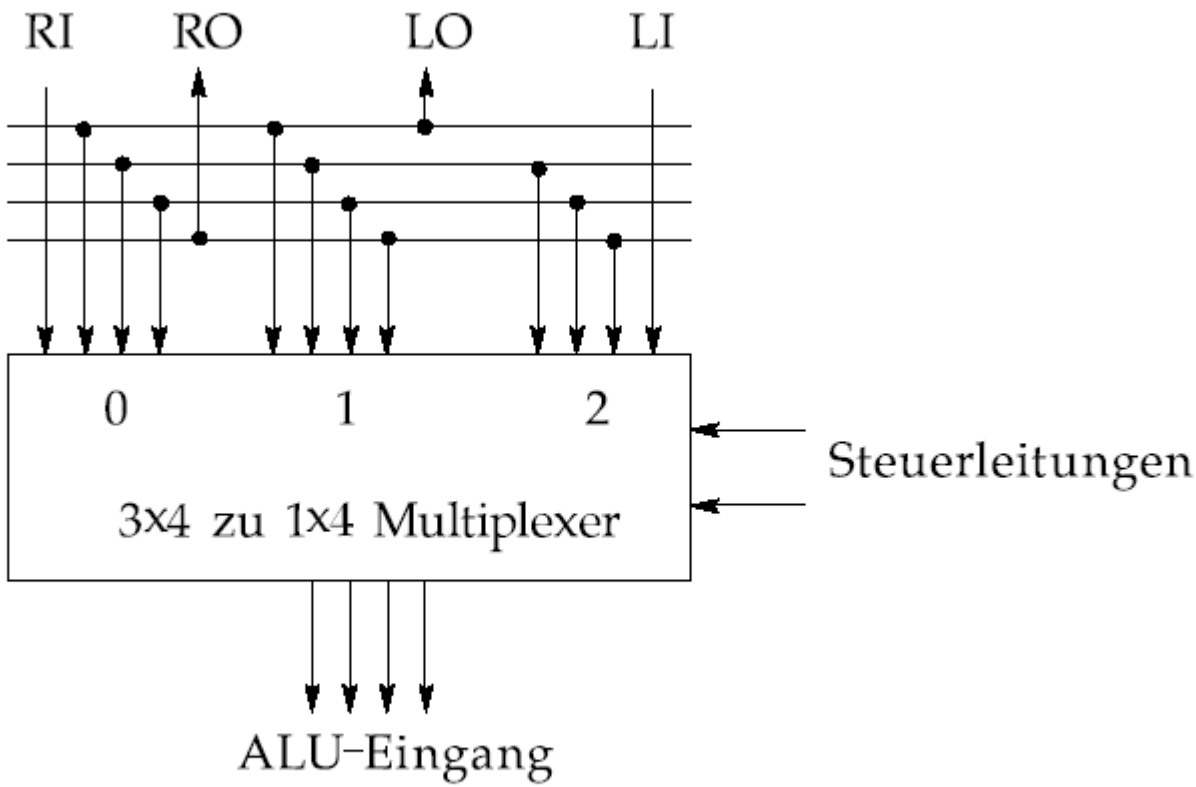


Abb. 2.8. 4–Bit Schiebemultiplexer (Shifter) vor einem ALU–Eingang

c_n	x_n	y_n	s_n	c_{n+1}
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

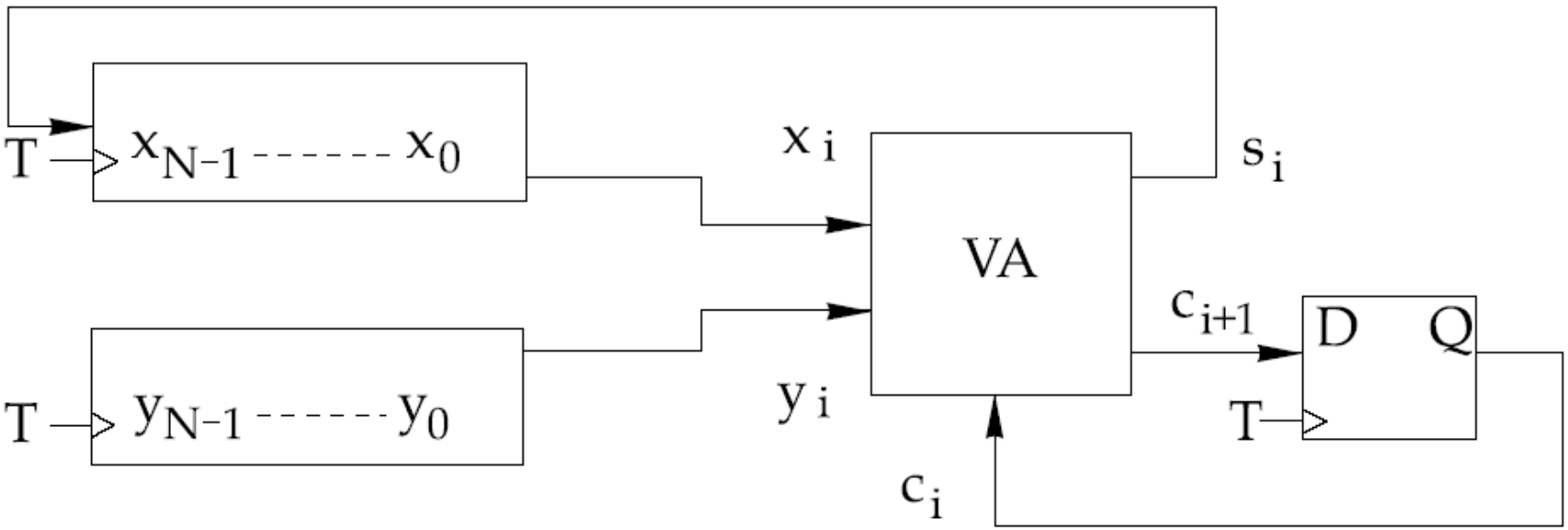


Abb. 2.9. Aufbau eines N–stelligen Serienaddierers

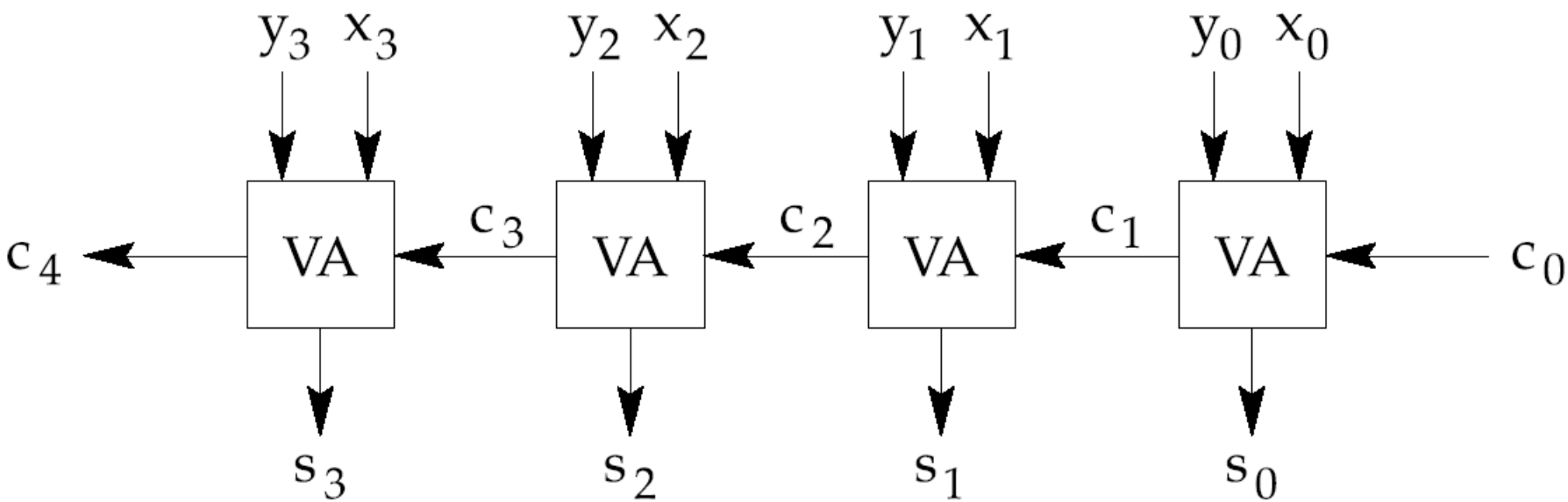


Abb. 2.10. Aufbau eines 4–Bit Ripple Carry Adders

Tabelle 2.1. Bestimmung des Hardwareaufwands bei einem Carry Look Ahead Adder ohne Hilfssignale

n	1	2	3	4	5	6	7	8
max. Eing. pro Schaltgl.	3	7	15	31	63	127	255	511
$\#UND(n)$	3	7	15	31	63	127	255	511
$\sum UND(n)$	3	10	25	56	119	246	501	1012
Schaltgl. insges.	4	12	28	60	124	252	508	1020

Tabelle 2.2. Hardwareaufwand bei einem Carry Look Ahead Adder mit Hilfssignalen

n	1	2	3	4	5	6	7	8
Verknüpfungsglieder insgesamt	2	6	12	20	30	42	56	72
max. Anzahl Eingänge pro Schaltglied	2	3	4	5	6	7	8	9

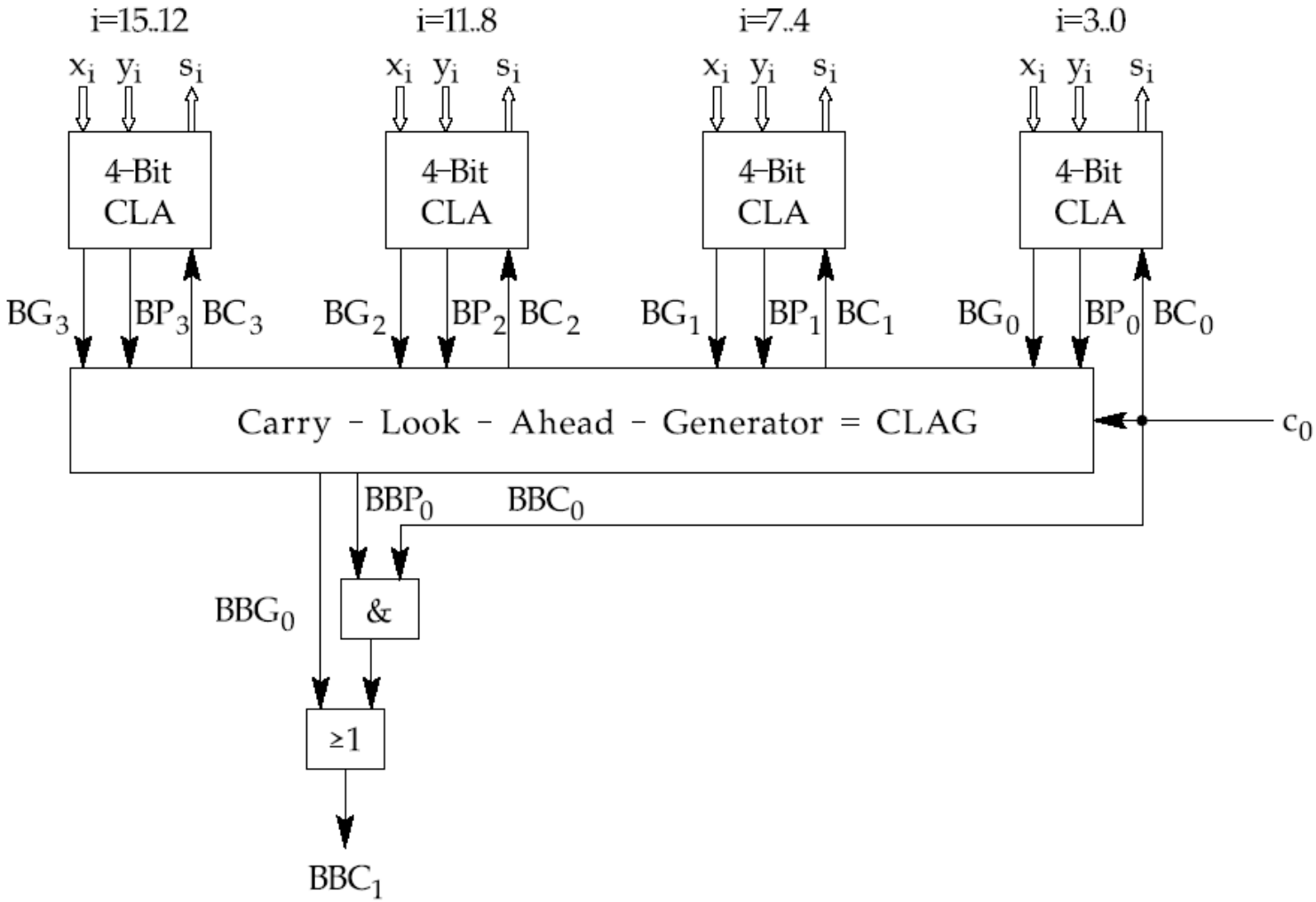


Abb. 2.11. Zweistufiger 16–Bit Carry Look Ahead Adder

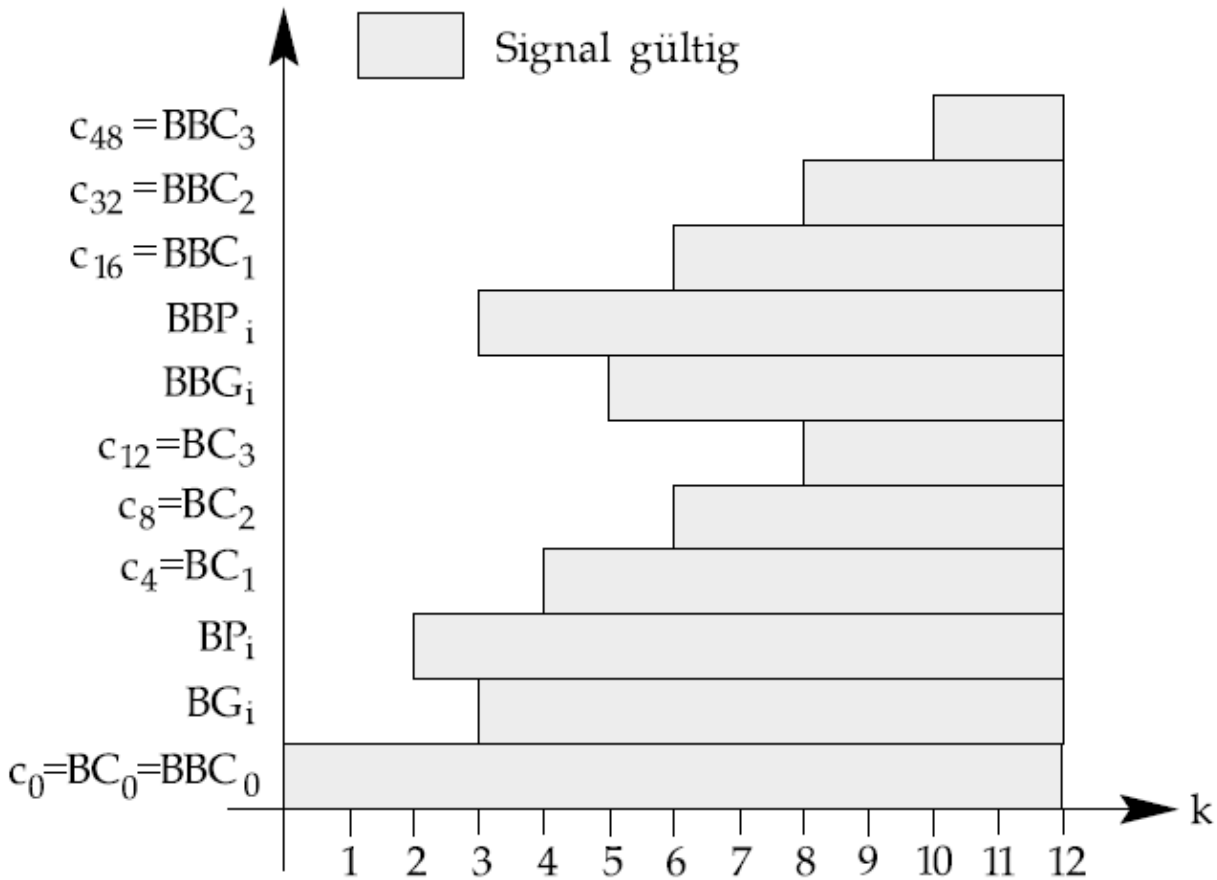


Abb. 2.12. Zur Bestimmung der Verzögerungszeiten bei einem dreistufigen 64–Bit Carry Look Ahead Adder ($k = \frac{t}{\tau}$)

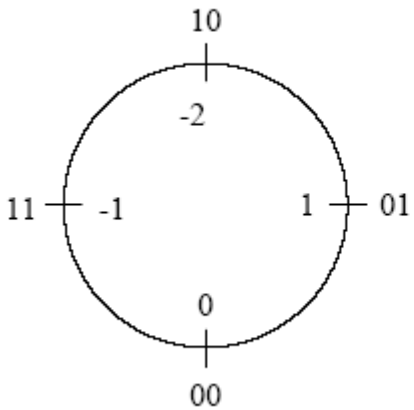


Abb. 2.13. Zweierkomplement–Darstellung einer 2–Bit Zahl. Innen: Dezimalwert, außen: duale Codierung

Tabelle 2.3. Zur Bestimmung der Schaltfunktion V

x_{N-1}	y_{N-1}	c_{N-1}	s_{N-1}	V
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

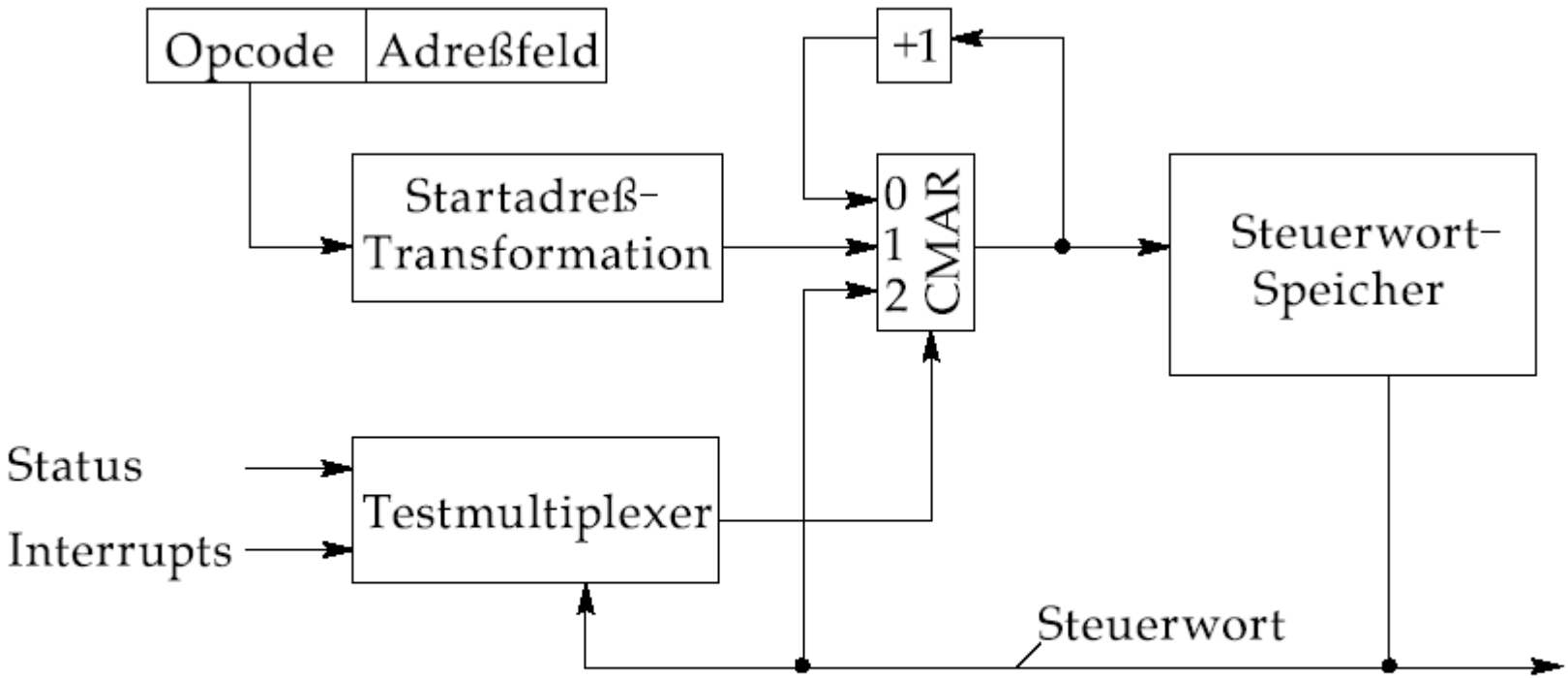


Abb. 2.14. Aufbau eines reagierenden Mikroprogramm–Steuerwerks zur Ablaufsteuerung

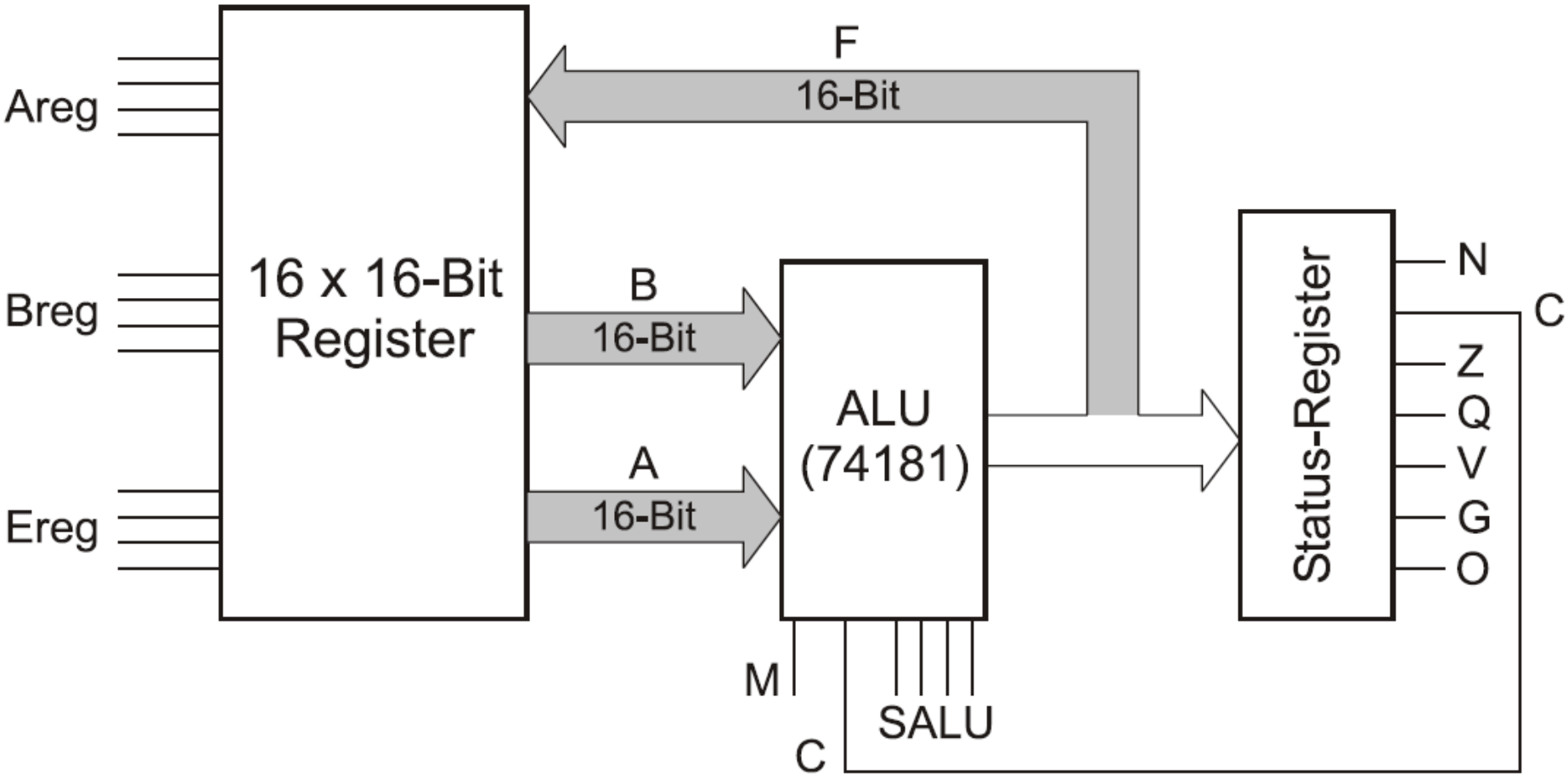
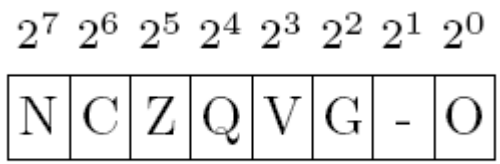


Abb. 2.15. Blockschaltbild der simulierten RALU

Steuerung	Rechenfunktionen		
SALU	M=1	M=0; Arithmetische Funktionen	
	Logische	C=0	C=1
$S_3S_2S_1S_0$	Funktionen	(kein Übertrag)	(mit Übertrag)
0 0 0 0	$F = \overline{A}$	$F = A$	$F = A + 1$
0 0 0 1	$F = \overline{A \vee B}$	$F = A \vee B$	$F = (A \vee B) + 1$
0 0 1 0	$F = \overline{A} \wedge B$	$F = A \vee \overline{B}$	$F = (A \vee \overline{B}) + 1$
0 0 1 1	$F = 0$	$F = -1$ (2erKompl.)	$F = 0$
0 1 0 0	$F = \overline{A \wedge B}$	$F = A + (A \wedge \overline{B})$	$F = A + (A \wedge \overline{B}) + 1$
0 1 0 1	$F = \overline{B}$	$F = (A \vee B) + (A \wedge \overline{B})$	$F = (A \vee B) + (A \wedge \overline{B}) + 1$
0 1 1 0	$F = A \oplus B$	$F = A - B - 1$	$F = A - B$
0 1 1 1	$F = A \wedge \overline{B}$	$F = (A \wedge \overline{B}) - 1$	$F = A \wedge \overline{B}$
1 0 0 0	$F = \overline{A} \vee B$	$F = A + (A \wedge B)$	$F = A + (A \wedge B) + 1$
1 0 0 1	$F = \overline{A \oplus B}$	$F = A + B$	$F = A + B + 1$
1 0 1 0	$F = B$	$F = (A \vee \overline{B}) + (A \wedge B)$	$F = (A \vee \overline{B}) + (A \wedge B) + 1$
1 0 1 1	$F = A \wedge B$	$F = (A \wedge B) - 1$	$F = A \wedge B$
1 1 0 0	$F = 1$	$F = A << B$ (rotate left)[12]	$F = A >> B$ (rotate right)
1 1 0 1	$F = A \vee \overline{B}$	$F = (A \vee B) + A$	$F = (A \vee B) + A + 1$
1 1 1 0	$F = A \vee B$	$F = (A \vee \overline{B}) + A$	$F = (A \vee \overline{B}) + A + 1$
1 1 1 1	$F = A$	$F = A - 1$	$F = A$



1. Programm: Addition zweier Register

```
;
;           ** Addition zweier Register **
;
set     1       #4      ; Reg1=Operand 1
set     2       #5      ; Reg2=Operand 2
;
dump                    ; Register ausgeben
control $09120          ; Reg0=Reg1+Reg2
carry   0               ; Carrybit loeschen
dump                    ; Register ausgeben
clock                   ; Takten->Ausfuehrung der Addition
dump                    ; Register ausgeben
quit                    ; Programm beenden
```

3. Hardware-Parallelität

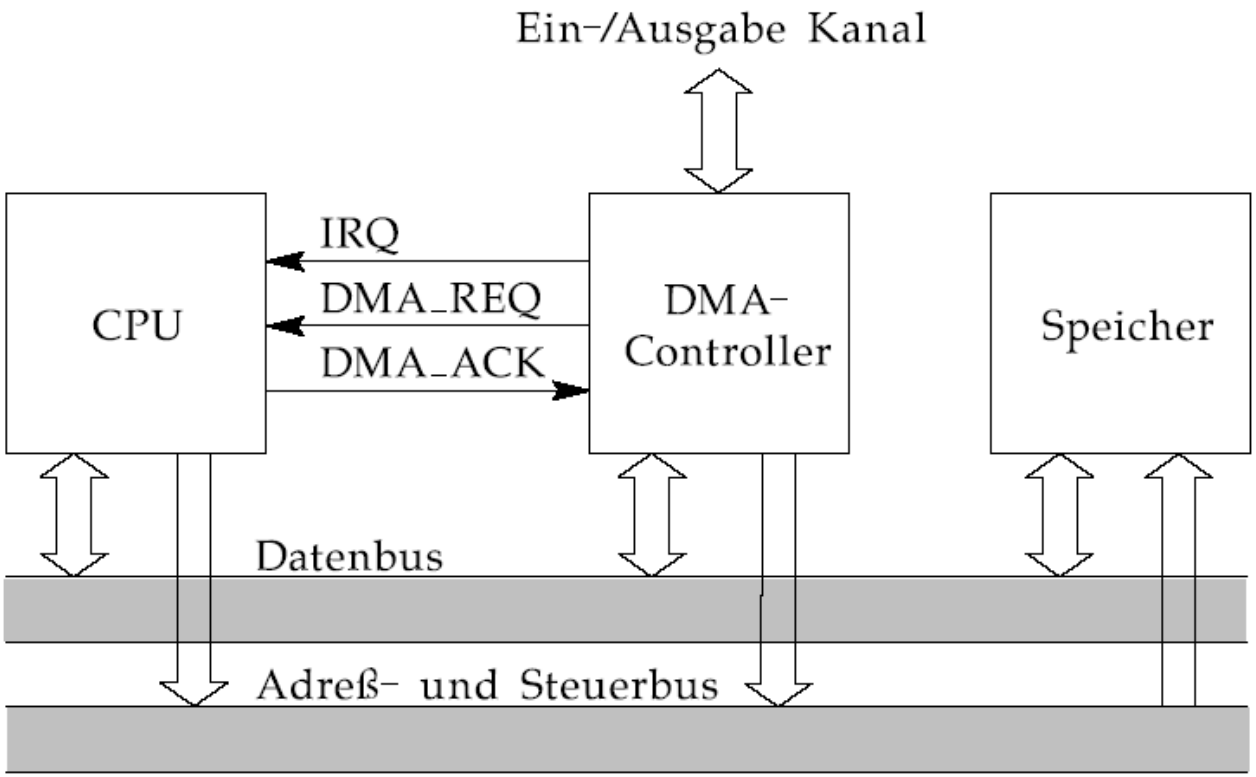


Abb. 3.1. Ein–/Ausgabe mit direktem Speicherzugriff durch einen DMA–Controller

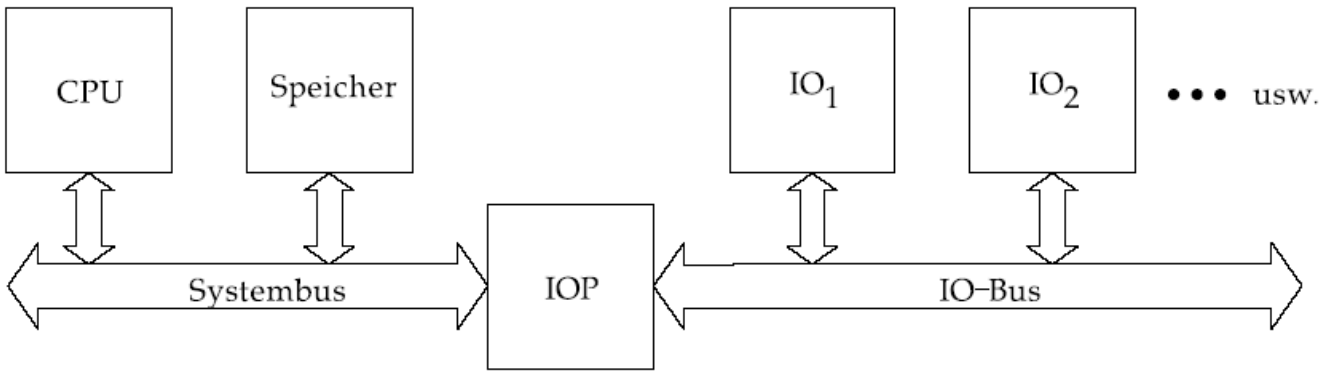


Abb. 3.2. Anschluss von Ein–/Ausgabe Geräten über einen IOP (Input Output Prozessor)

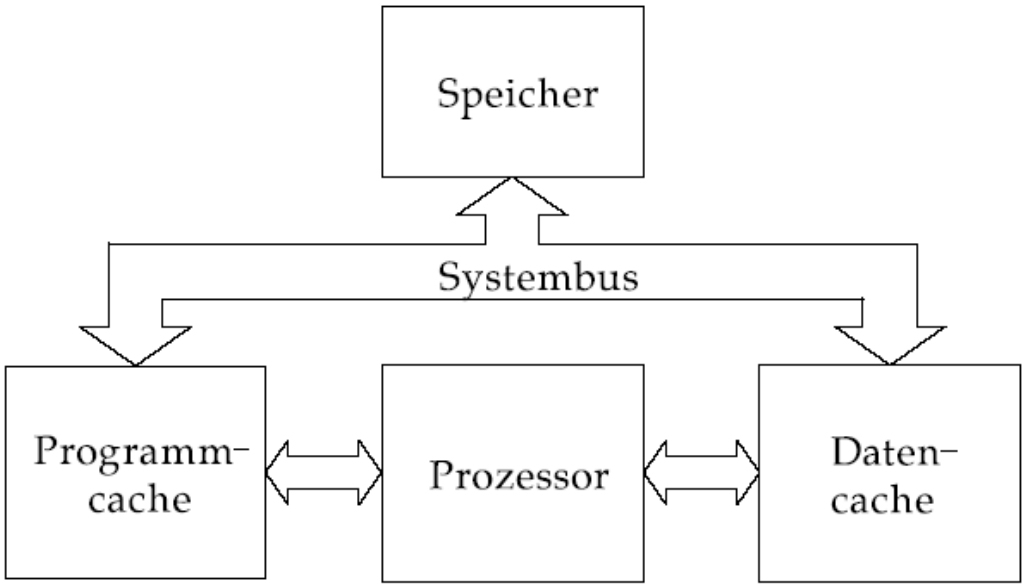
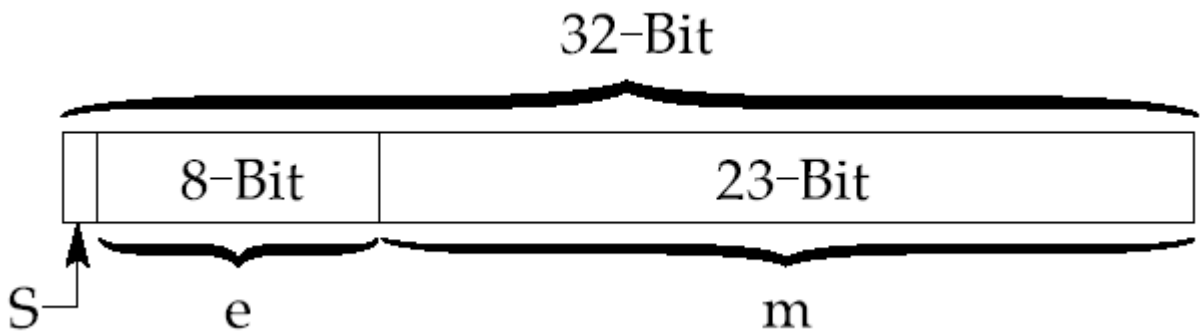


Abb. 3.3. Prozessor mit HARVARD–Architektur und großem Hauptspeicher


32-Bit
8-Bit
23-Bit
S
e
m

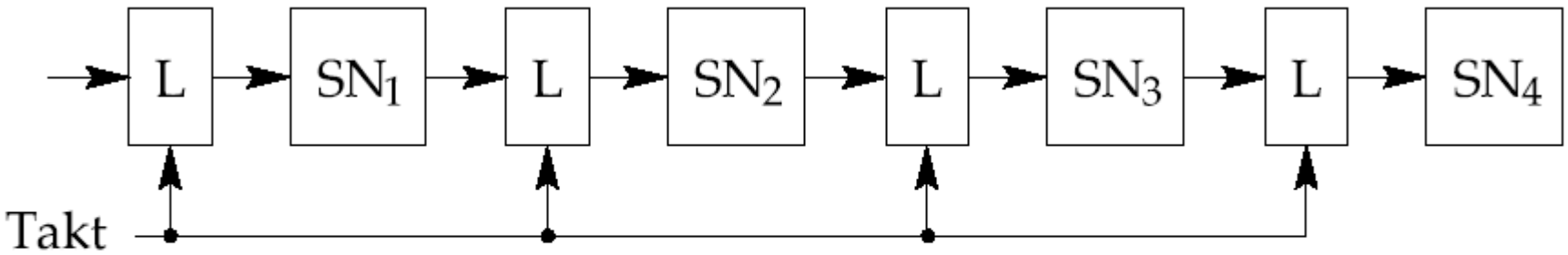


Abb. 3.4. Vierstufige Pipeline mit Latches (L) und Schaltnetzen (SN)

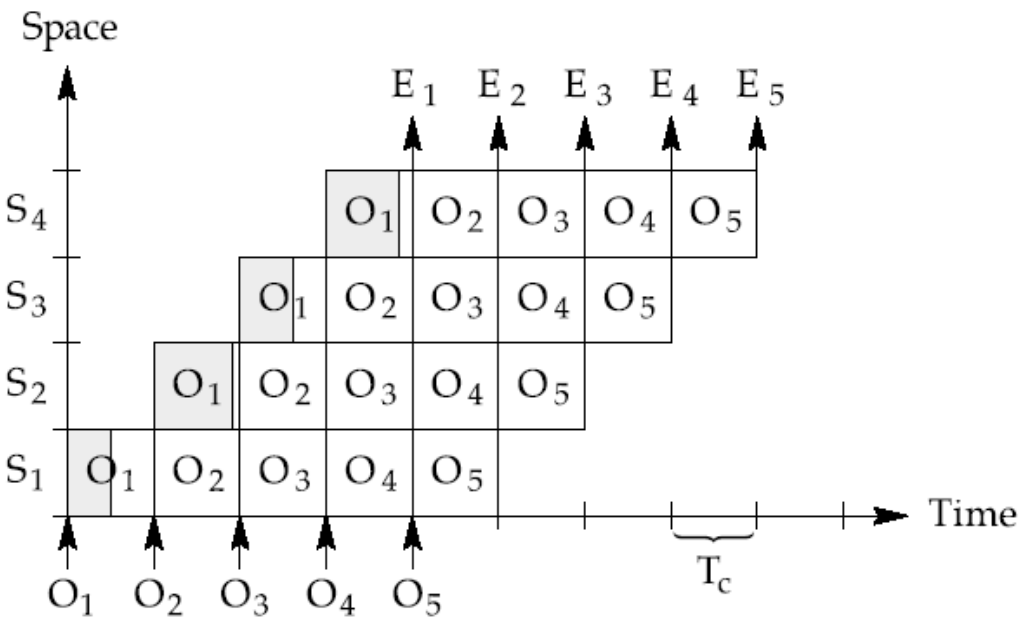


Abb. 3.5. Time–Space Diagramm zur Darstellung der überlappenden Verarbeitung in einer Pipeline

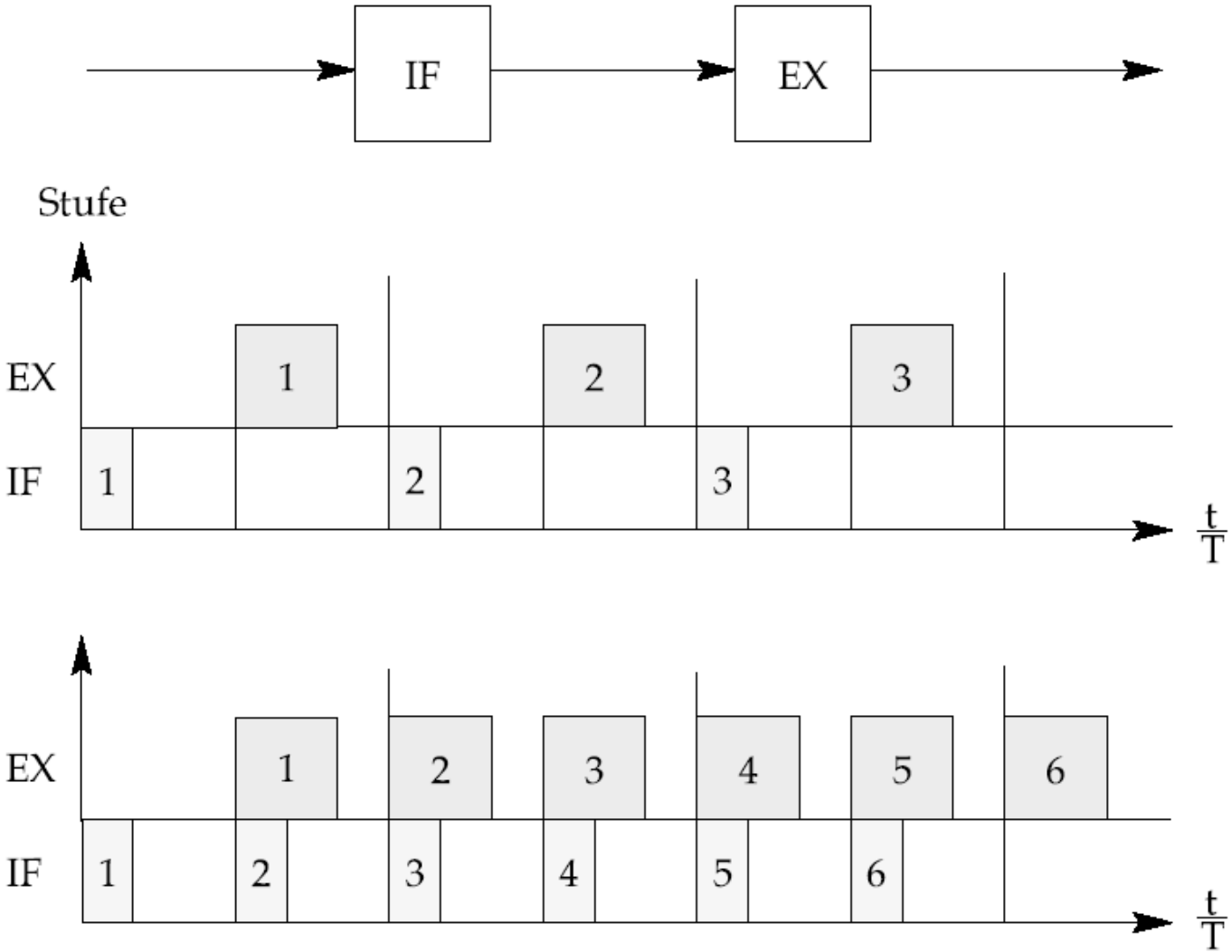


Abb. 3.6. Einfaches Befehlspipelining (Oben: Aufbau; Mitte: ohne Pipelining; Unten: mit Pipelining

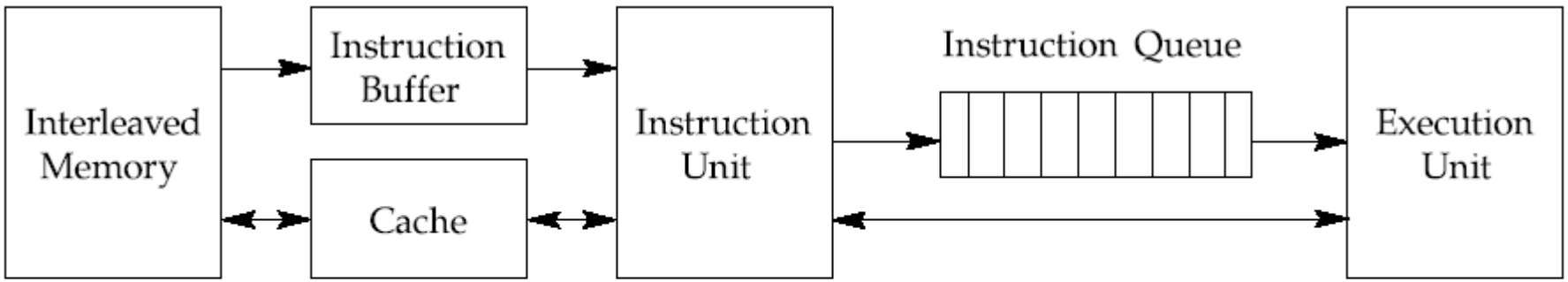


Abb. 3.7. Architektur eines Prozessors, der Befehls– und Arithmetik–Pipelines unterstützt.

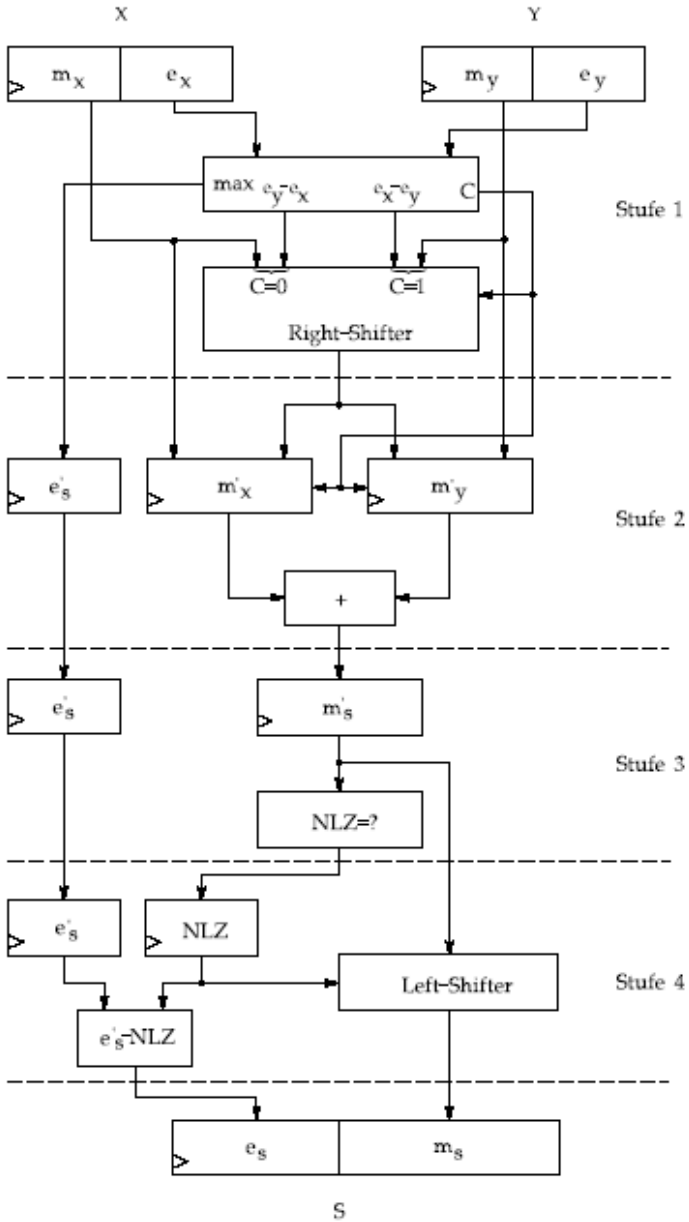


Abb. 3.8. Aufbau eines Gleitkomma–Addierers mit vierstufigem Pipelining

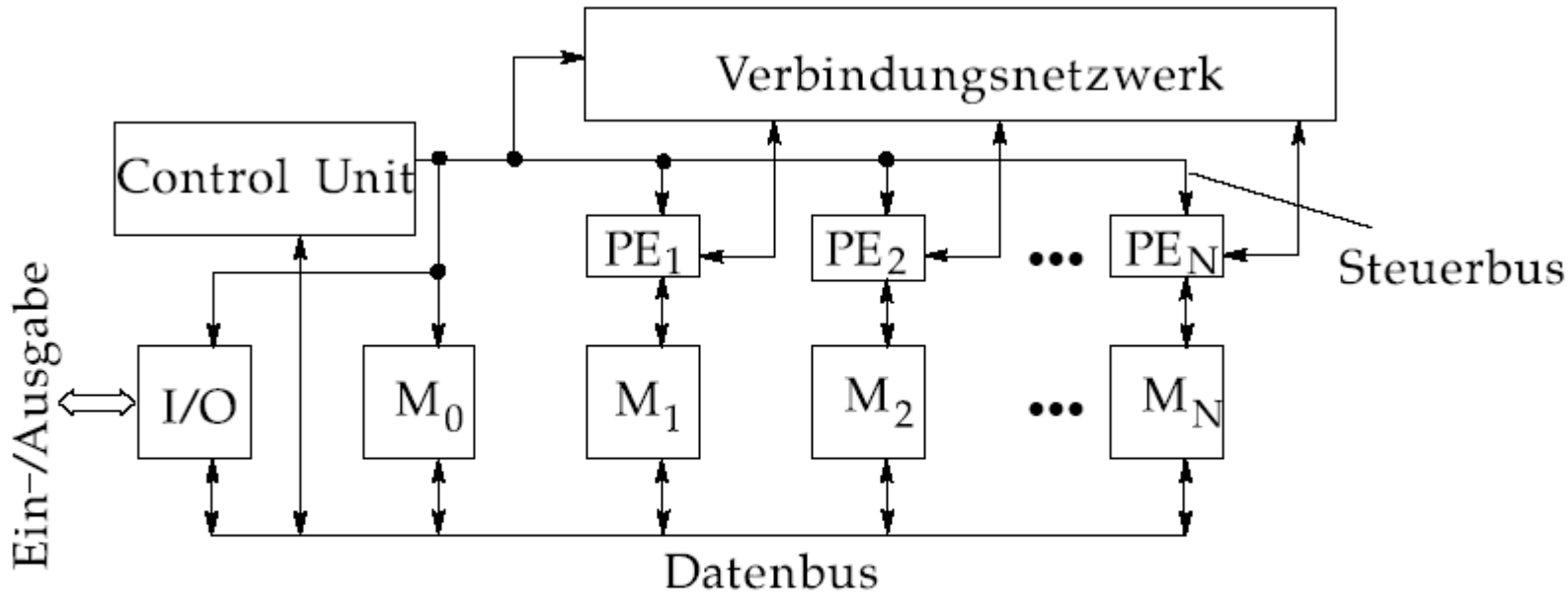


Abb. 3.9. Aufbau eines Array–Prozessors

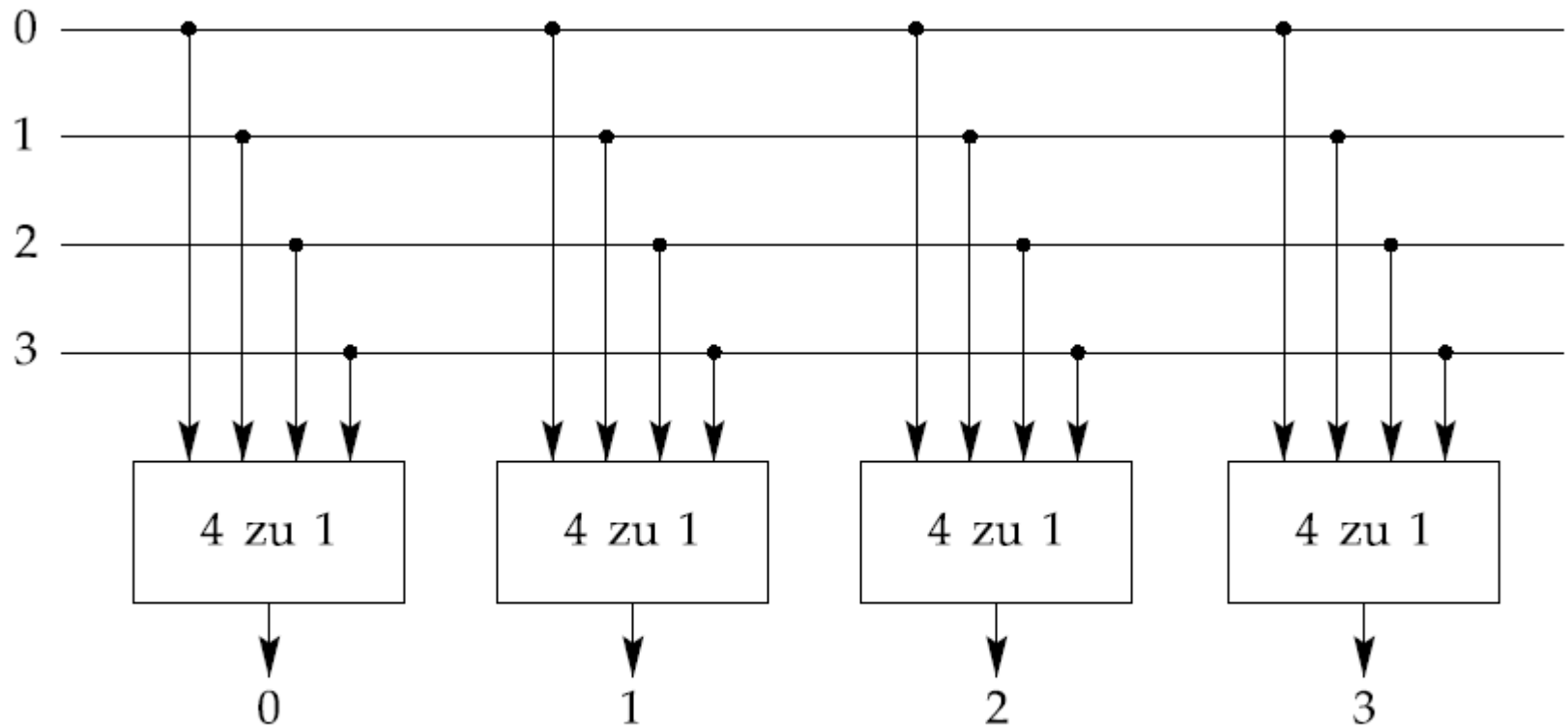


Abb. 3.10. Aufbau eines Kreuzschienenverteilers für 4 Verarbeitungselemente

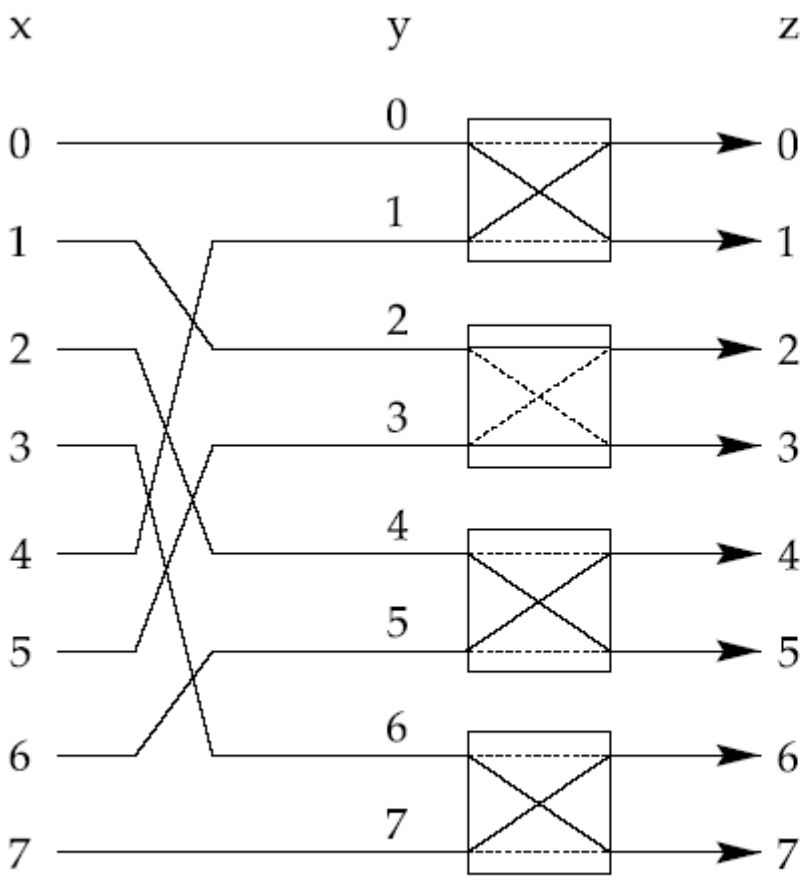


Abb. 3.11. Aufbau eines Shuffle–Exchange Netzes

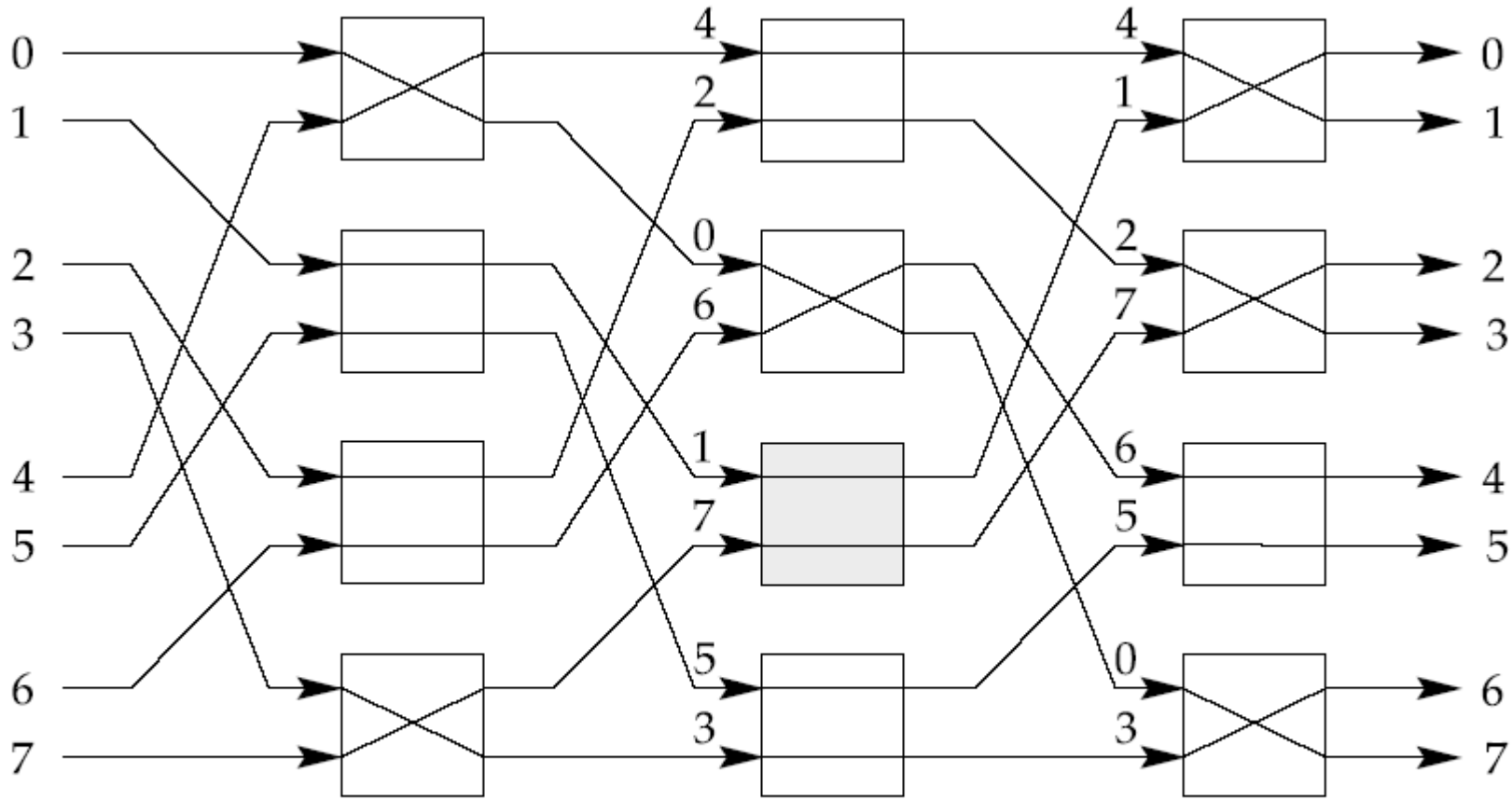


Abb. 3.12. Aufbau eines Omega–Netzwerks mit einzelnen steuerbaren Schaltereinheiten

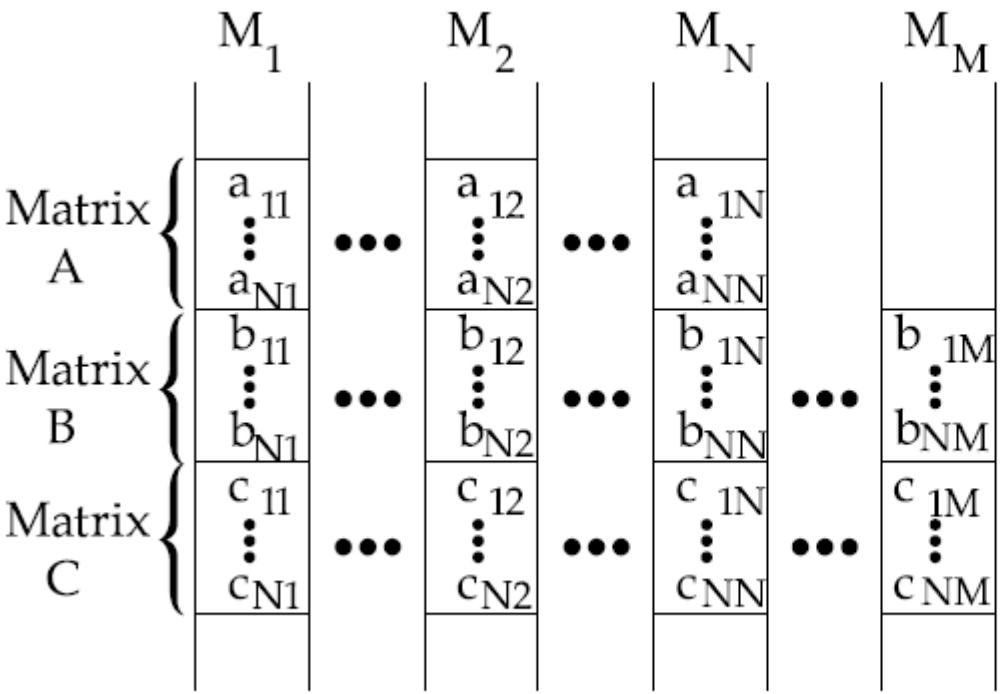


Abb. 3.13. Aufteilung der Matrizen A, B und C auf die lokalen Speicher der Verarbeitungselemente.

i	j	M_1	M_2	M_3	$Netz$
1	1	$c_{11} = c_{11} + a_{11}b_{11}$	$c_{12} = c_{12} + a_{11}b_{12}$	$c_{13} = c_{13} + a_{11}b_{13}$	a_{11}
	2	$c_{11} = c_{11} + a_{12}b_{21}$	$c_{12} = c_{12} + a_{12}b_{22}$	$c_{13} = c_{13} + a_{12}b_{23}$	a_{12}
2	1	$c_{21} = c_{21} + a_{21}b_{11}$	$c_{22} = c_{22} + a_{21}b_{12}$	$c_{23} = c_{23} + a_{21}b_{13}$	a_{21}
	2	$c_{21} = c_{21} + a_{22}b_{21}$	$c_{22} = c_{22} + a_{22}b_{13}$	$c_{23} = c_{23} + a_{22}b_{23}$	a_{22}

4. Prozessorarchitektur

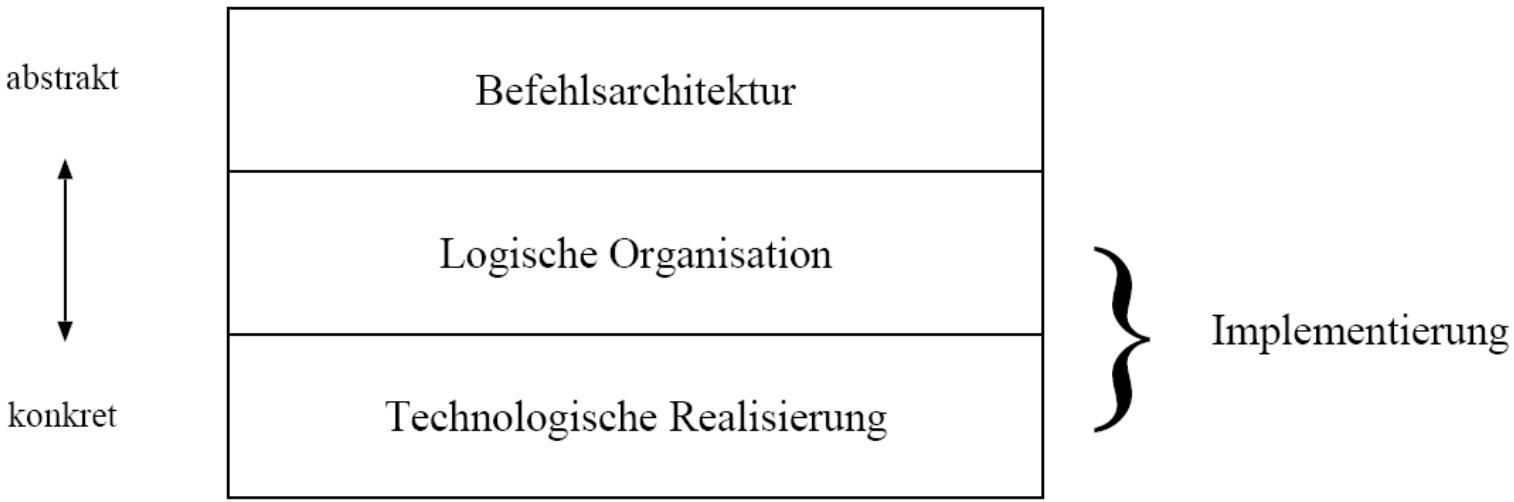


Abb. 4.1. Sichtweisen der Rechnerarchitektur

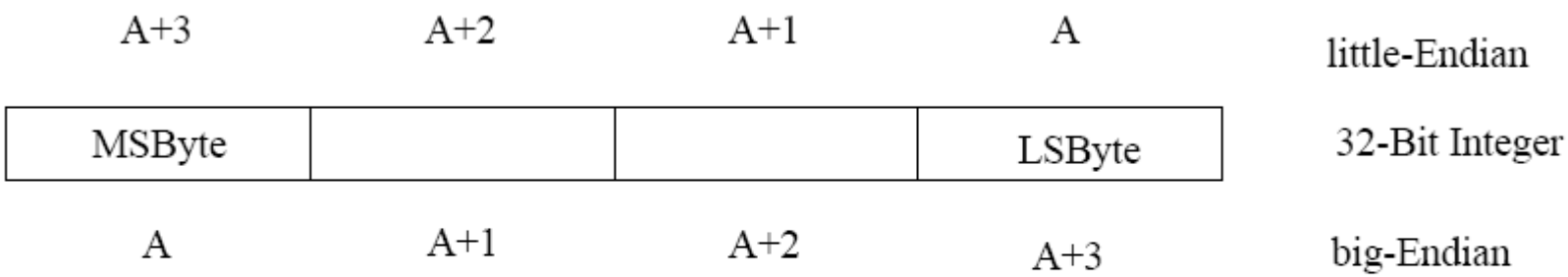


Abb. 4.2. Vergleich zwischen Little– und Big–Endian Speicheradressierung

1. Datenübertragung
   - Register zu Register
   - Register zu Speicher bzw. Ein–/Ausgabe
2. Datenmanipulation
   - Arithmetische Verknüpfungen
   - Logische Verknüpfungen
   - Schiebe–Operationen
3. Verzweigungen (bedingt oder unbedingt)
   - Sprünge
   - Unterprogramme
4. Maschinensteuerung
   - Interrupts oder Traps
   - Speicherverwaltung

Tabelle 4.1. Die zehn häufigsten Befehle beim INTEL 80x86 (nach [*Hennessy und Patterson*, 1996])

Platz	Befehl	Häufigkeit in %
1	LOAD	22
2	CONDITIONAL BRANCH	20
3	COMPARE	16
4	STORE	12
5	ADD	8
6	AND	6
7	SUB	5
8	MOVE REGISTER	4
9	CALL	1
10	RETURN	1
	Gesamt	96

Tabelle 4.2. Vergleich technologischer Eigenschaften vom ersten Mikroprozessor bis zu einem hypothetischem Prozessor des Jahres 2011

	INTEL 4004	Pentium 4	AMD Opteron	Micro 2011
Jahr	1971	2000	2002	2011
Architektur	CISC	RISC	RISC	RISC
Strukturgröße	10μm	$0{,}18\mu$m	$0{,}13\mu$m	$0{,}07\mu$m
Anzahl Transistoren	$2{,}3{\cdot}10^3$	$42{\cdot}10^6$	$93 \cdot 10^6$	10^9
Chipfläche	12mm^2	$180\ \text{mm}^2$	$302\ \text{mm}^2$	$2.090\ \text{mm}^2$
Taktfrequenz	0,75 MHz	1,3 GHz	2 GHz	10 GHz

5. CISC-Prozessoren

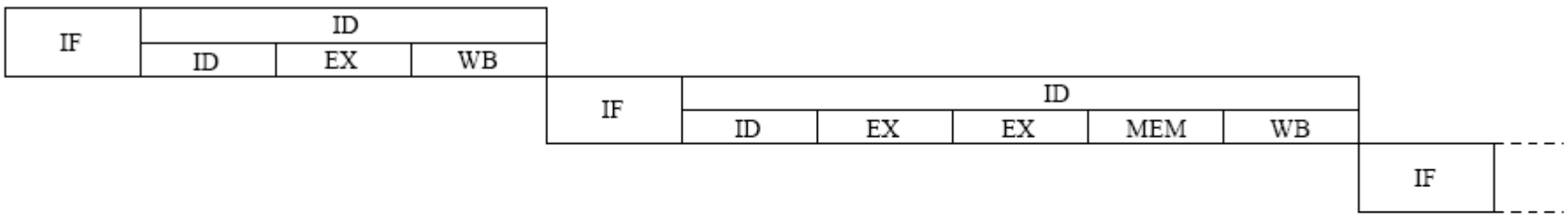


Abb. 5.1. Zeitlicher Ablauf von drei Maschinenbefehlen bei einem CISC–Prozessor

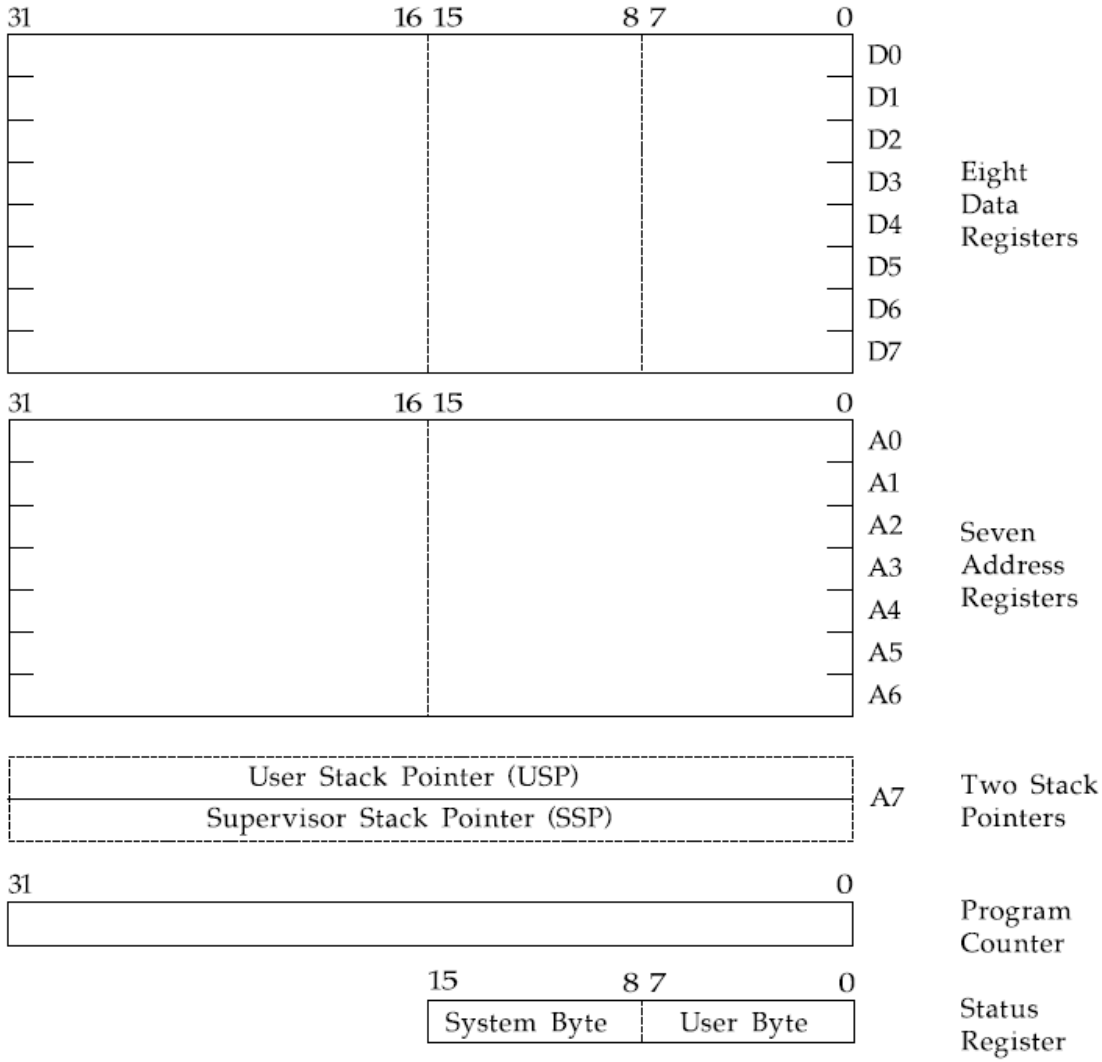


Abb. 5.2. Programmiermodell des Motorola 68000

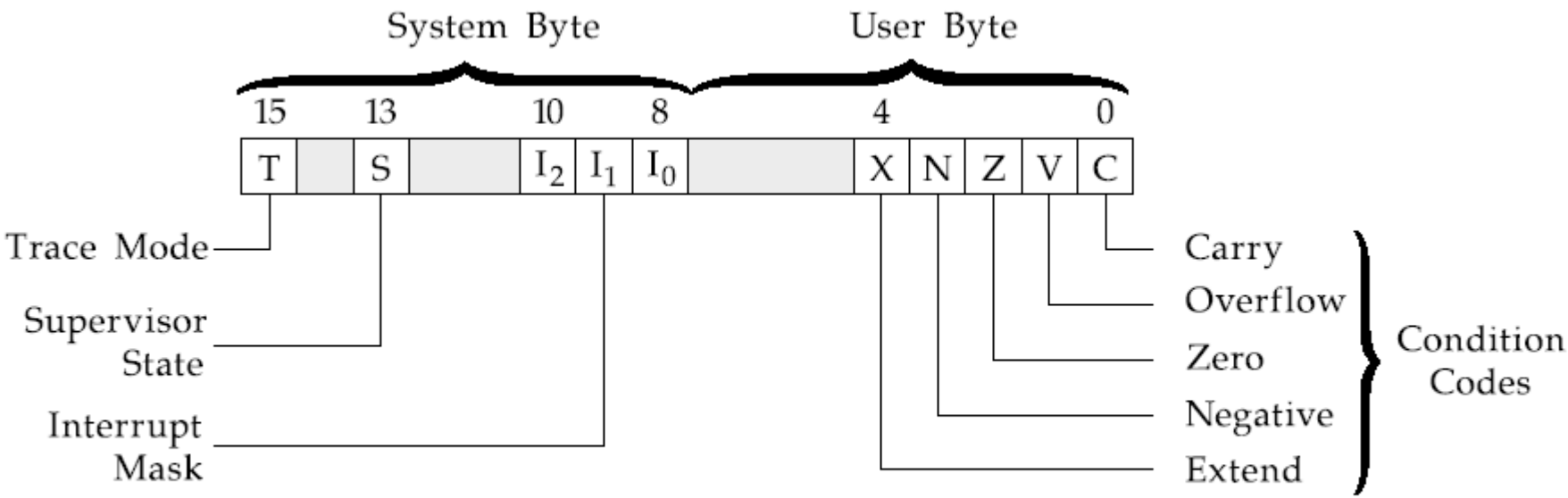


Abb. 5.3. Statusregister des Motorola 68000

Tabelle 5.1. Addressierungsarten des 68000

Adressierungsart	Adresserzeugung	Syntax
Register direkt		
Daten–Register direkt	EA = Dn	Dn
Adress–Register direkt	EA = An	An
Register indirekt		
Adress–Register indirekt	EA = (An)	(An)
Adress–Register indirekt mit Pre–Dekrement	An – N → An, EA = (An)	–(An)
Adress–Register indirekt mit Post–Inkrement	EA = (An), An + N → An	(An)+
Adress–Register indirekt mit Offset	EA = (An) + d_{16}	d(An)
Adress–Register indirekt mit Index und Offset	EA = (An) + (X_i) + d_8	d(An, X_i)
Absolute Adressierung		
Absolut kurz	EA = (Nächstes Wort)	hhhh
Absolut lang	EA = (Nächstes und übernächstes Wort)	hhhh.hhhh
Unmittelbare Adressierung		
Unmittelbar	Daten = Nächstes Wort/Nächste Wörter	# hhhh
Unmittelbar schnell	Befehlsinherente Daten	# hh
PC–relative Adressierung		
PC–relativ mit Offset	EA = (PC) + d_{16}	d
PC–relativ mit Index und Offset	EA = (PC) + (X_i) + d_8	d(X_i)
Implizite Adressierung		
Implizite Register	EA = SR, PC, USP, SSP	

Tabelle 5.2. Abbkürzungen in Tabelle 5.1

EA = Effektive Adresse	SP = aktiver Systemstapelzeiger
An = Adress–Register	d_8 = 8 Bit Offset
Dn = Daten–Register	d_{16} = 16 Bit Offset
X_i = Index–Register, Adress–/Daten–Register	h = hexadezimales Digit
SR = Statusregister	N = 1 für Byte, 2 für Wort, 4 für Langwort
PC = Programmzähler	(An) = indirekte Adressierung mit An
SSP = Supervisor–Stapelzeiger	
USP = User–Stapelzeiger	b → a = a wird durch b ersetzt

Folgende Befehlsklassen werden unterschieden:

- Datentransfer
- Integer–Arithmetik
- BCD–Arithmetik
- Logische Operationen
- Schieben und Rotieren
- Bit Manipulation
- Programmsteuerung
- Systemsteuerung

Tabelle 5.3. Befehle zum Datentransfer

Befehl	Beschreibung
EXG	Exchange Registers
LEA	Load Effective Address
LINK	Link Stack
MOVE	Move Data
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MOVEQ	Move Quick
PEA	Push Effective Address
SWAP	Swap Data Register Halves
UNLK	Unlink Stack

Tabelle 5.4. Befehle zur Integer–Arithmetik

Befehl	Beschreibung
ADD	Add
CLR	Clear Operand
CMP	Compare
DIVS	Signed Divide
DIVU	Unsigned Divide
EXT	Sign Extend
MULS	Signed Multiply
MULU	Unsigned Multiply
NEG	Negate
NEGX	Negate with Extend
SUB	Subtract
SUBX	Subtract with Extend
TAS	Test and Set an Operand
TST	Test an Operand

Tabelle 5.5. Befehle zur BCD–Arithmetik

Befehl	Beschreibung
ABCD	Add Decimal with Extend
SBCD	Subtract Decimal with Extend
NBCD	Negate Decimal with Extend

Tabelle 5.6. Logische Befehle werden bitweise angewandt.

Befehl	Beschreibung
AND	AND logical
OR	Inclusive OR logical
EOR	Exclusive OR logical
NOT	Logical Complement

Tabelle 5.7. Befehle zum Schieben und Rotieren.

Befehl	Beschreibung
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
LSL	Logical Shift Left
LSR	Logical Shift Right
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend

Tabelle 5.8. Befehle zur Bit-Manipulation.

Befehl	Beschreibung
BTST	Test a Bit and Change
BSET	Test a Bit and Set
BCLR	Test a Bit and Clear
BCHG	Test a Bit and Change

Tabelle 5.9. Condition Codes des 68000.

CC	Carry Clear	LS	Low or Same
CS	Carry Set	LT	Less Than
EQ	Equal	MI	Minus
F	Never True	NE	Not Equal
GE	Greater Equal	PL	Plus
GT	Greater Than	T	Always True
HI	High	VC	No Overflow
LE	Less or Equal	VS	Overflow

Tabelle 5.10. Verzweigungsbefehle des 68000.

Befehl	Beschreibung
Bcc	Branch Conditionally
DBcc	Test Condition, Decrement, and Branch
Scc	Set According to Condition
BRA	Branch Always
BSR	Branch to Subroutine
JMP	Jump
JSR	Jump to Subroutine
RTR	Return and Restore Condition Codes
RTS	Return from Subroutine

Tabelle 5.11. Befehle zur Systemsteuerung

Befehl	Beschreibung
ANDI to SR	AND Immediate to Status Register
EORI to SR	Exclusive OR Immediate to Status Register
MOVE from/to SR	Move Status Register
MOVE from/to USP	Move User Stack Pointer
ORI to SR	OR Immediate to Status Register
RESET	Reset External Devices
RTE	Return from Exception
STOP	Load Status Register and Stop
CHK	Check Register Against Bounds
TRAP	Trap
TRAPV	Trap on Overflow
MOVE to CCR	Move to Condition Codes
ANDI to CCR	AND Immediate to Condition Codes
EORI to CCR	Exclusive OR Immediate to Condition Codes
ORI to CCR	OR Immediate to Condition Codes

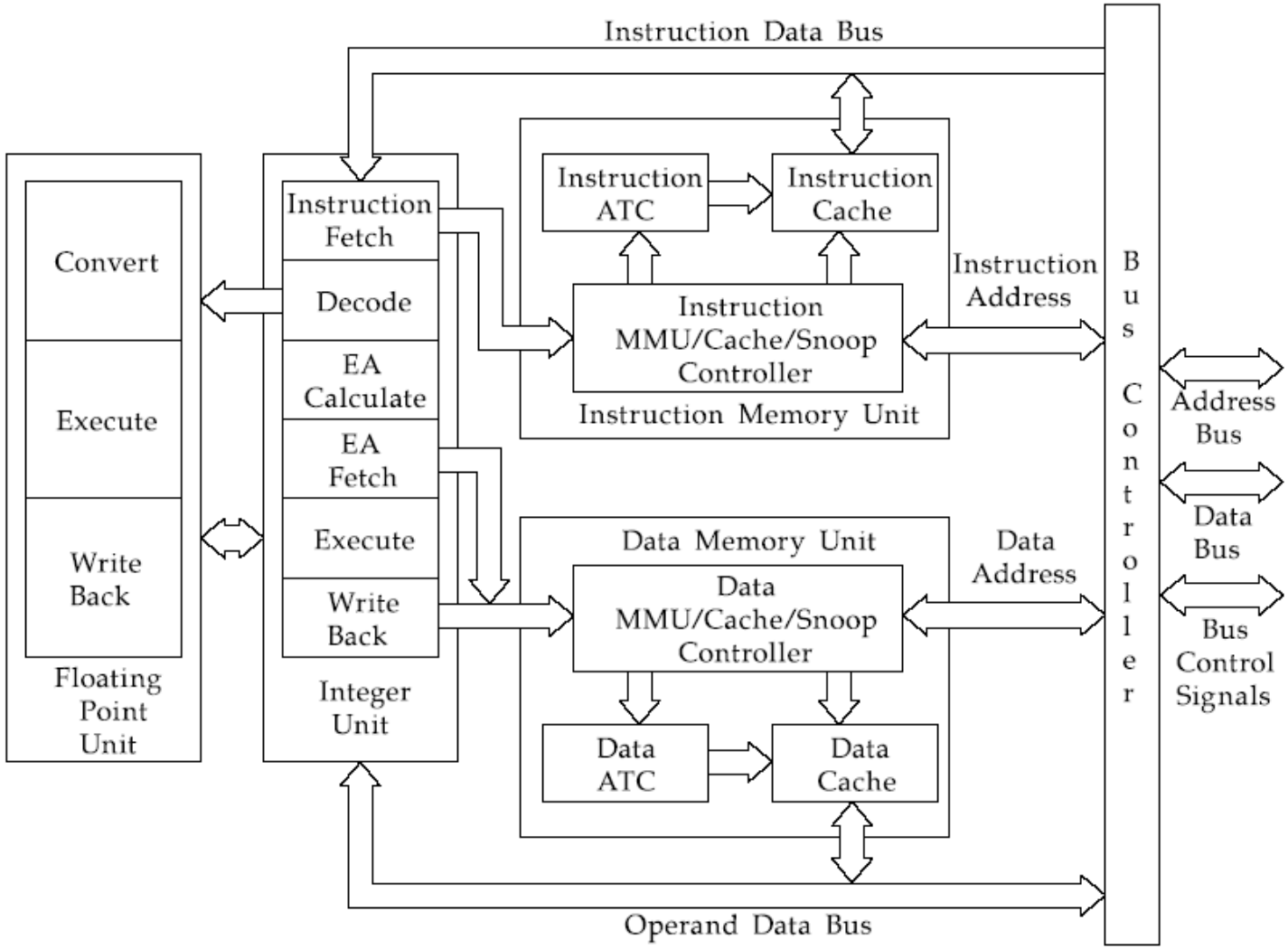


Abb. 5.4. Aufbau des Motorola 68040

6. RISC-Prozessoren

6.1.2 RISC–Definition

D. Tabak gibt die wohl eindeutigste Definition für eine RISC–Architektur [*Tabak*, 1995]. Danach soll ein RISC–Prozessor möglichst viele der folgenden neun Kriterien erfüllen:

1. Mindestens 80% der Befehle werden in einem Taktzyklus ausgeführt.
2. Alle Befehle werden mit einem Maschinenwort codiert.
3. Maximal 128 Maschinenbefehle.
4. Maximal 4 Befehlsformate.
5. Maximal 4 Adressierungsarten.
6. Speicherzugriffe ausschließlich über LOAD/STORE–Befehle.
7. Register–Register Architektur.
8. Festverdrahtete Ablaufsteuerung (nicht mikroprogrammiert).
9. Mindestens 32 Prozessorregister.

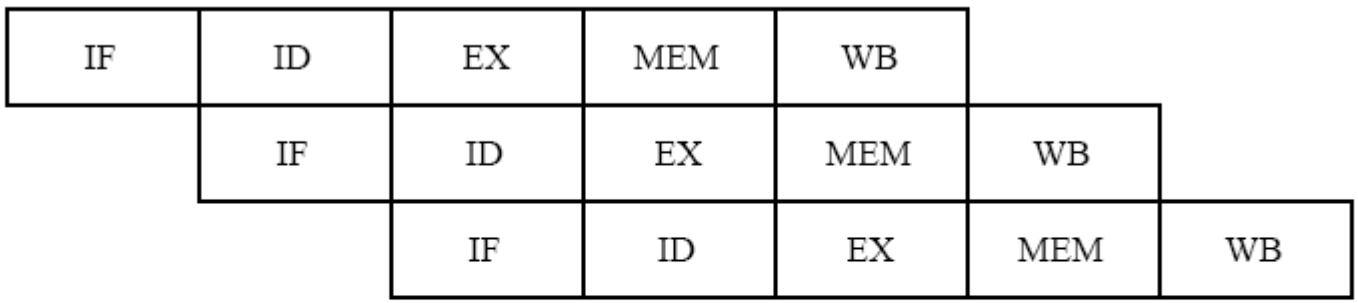


Abb. 6.1. Zeitlicher Ablauf der Teilschritte bei konfliktfreiem Befehlspipelining (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory Access, WB = Write Back)

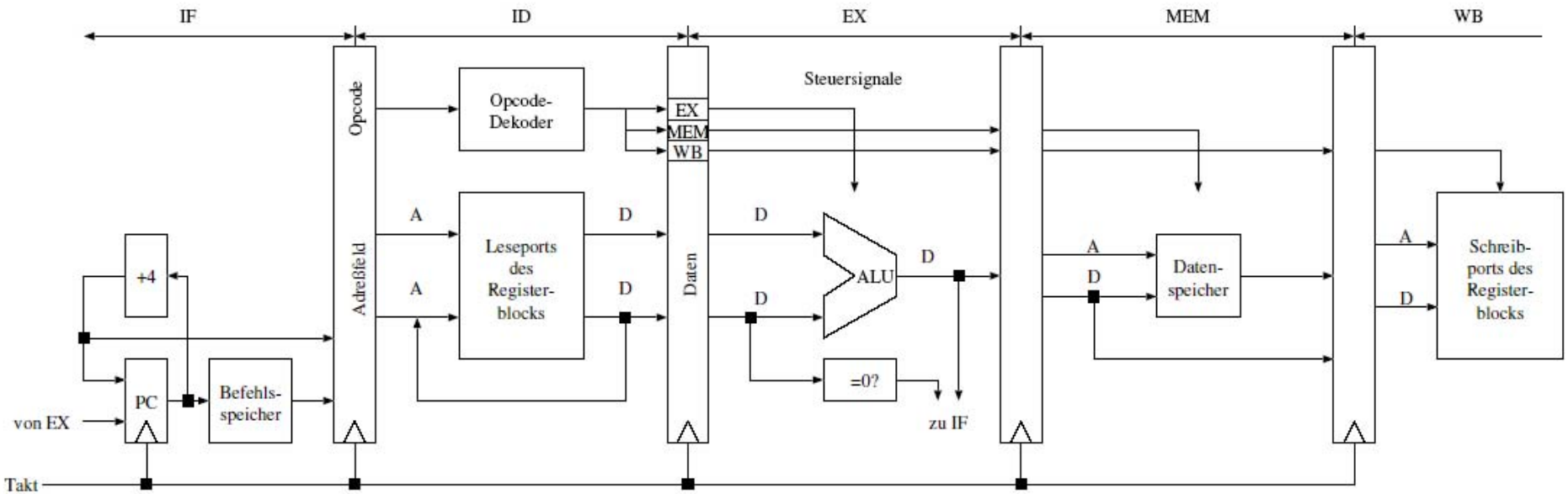


Abb. 6.2. Schematischer Aufbau eines RISC–Prozessors mit fünfstufigem Befehlspipelining und getrenntem Befehls– und Datenspeicher (Instruction/Data Memory IM/DM)

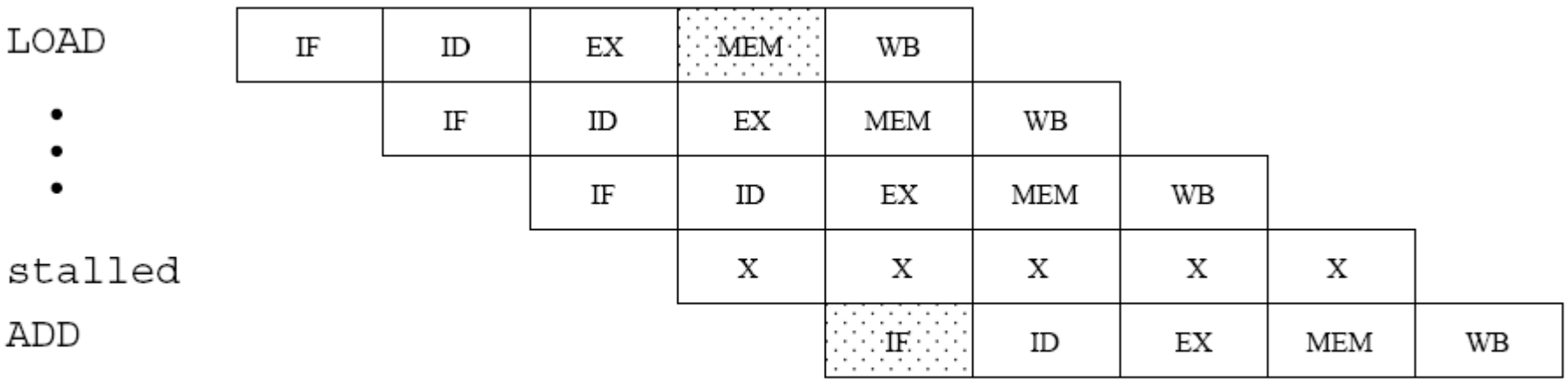


Abb. 6.3. Beispiel für die Behebung eines strukturellen Konflikts durch Anhalten der Pipelineverarbeitung

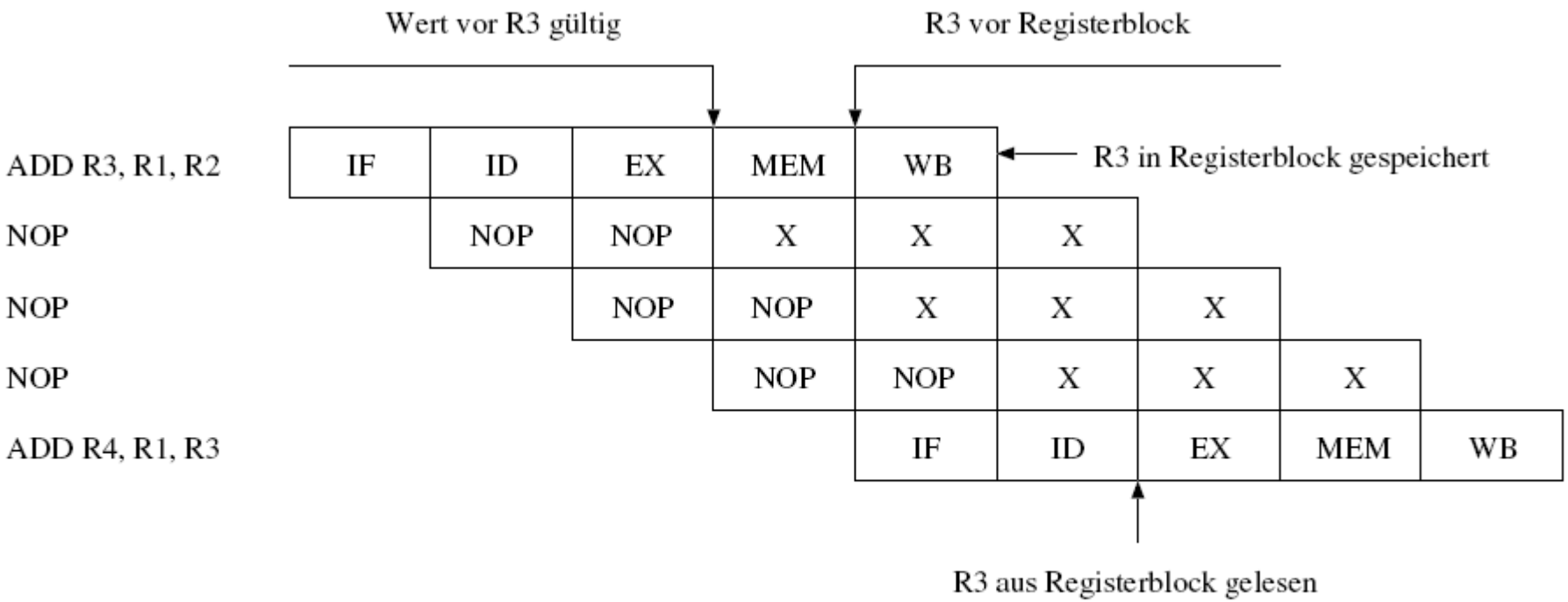


Abb. 6.4. Beispiel für die Behebung eines RAW Datenflusskonflikts durch Einfügen von NOPs

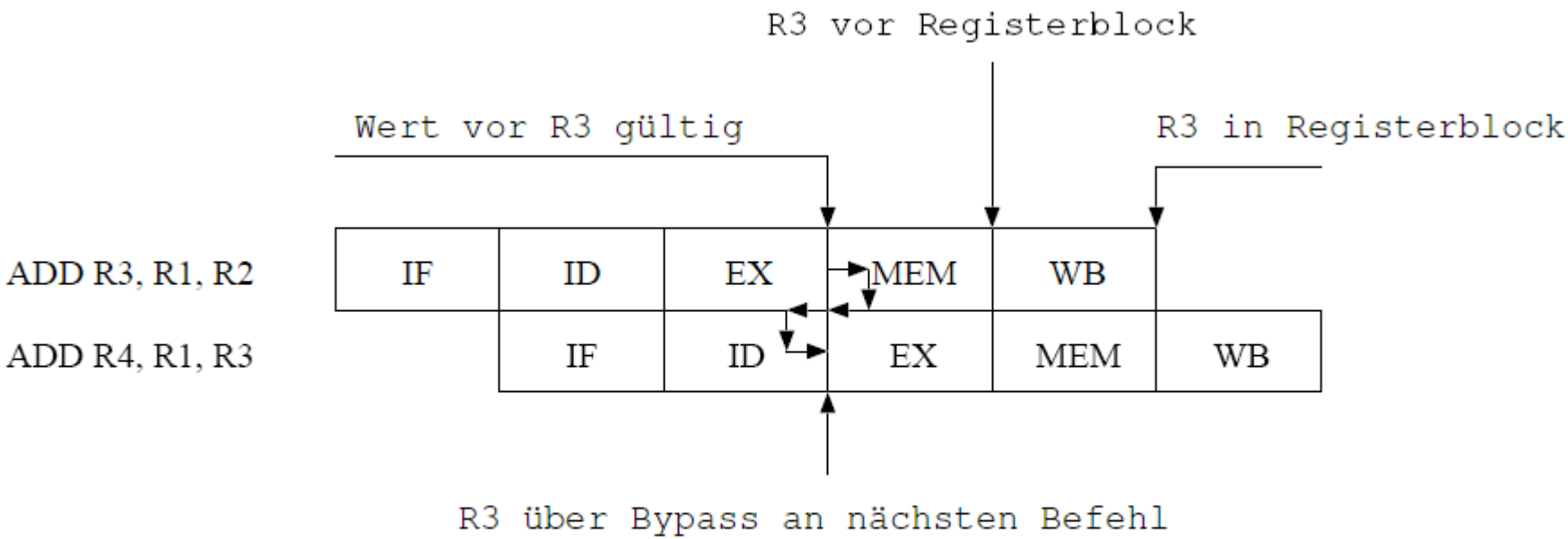


Abb. 6.5. Beseitigung des Datenflusskonflikts aus Abb. 6.4 mit Hilfe einer Bypass Hardware, die den Registerblock umgeht.

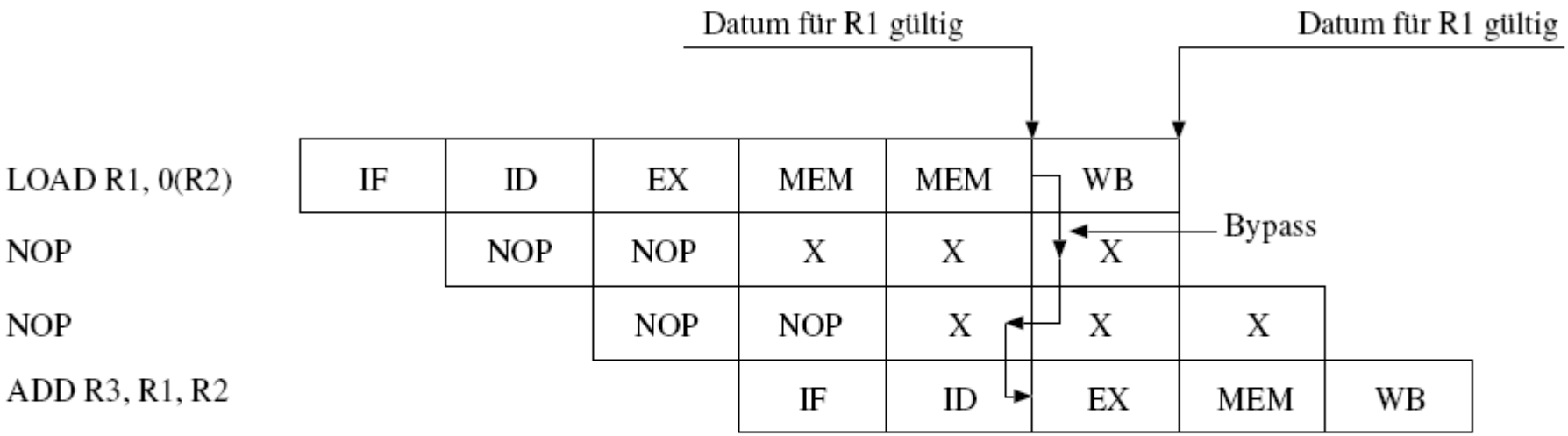


Abb. 6.6. Lesezugriff auf den Hauptspeicher, der zwei Taktzyklen erfordert. Der ADD–Befehl muss wegen der Datenabhängigkeit mit dem LOAD–Befehl ebenfalls um zwei Taktzyklen verzögert werden. Damit zwischen LOAD– und ADD–Befehl kein struktureller Pipelinekonflikt besteht, wird ein getrennter Daten– und Befehlsspeicher verwendet.

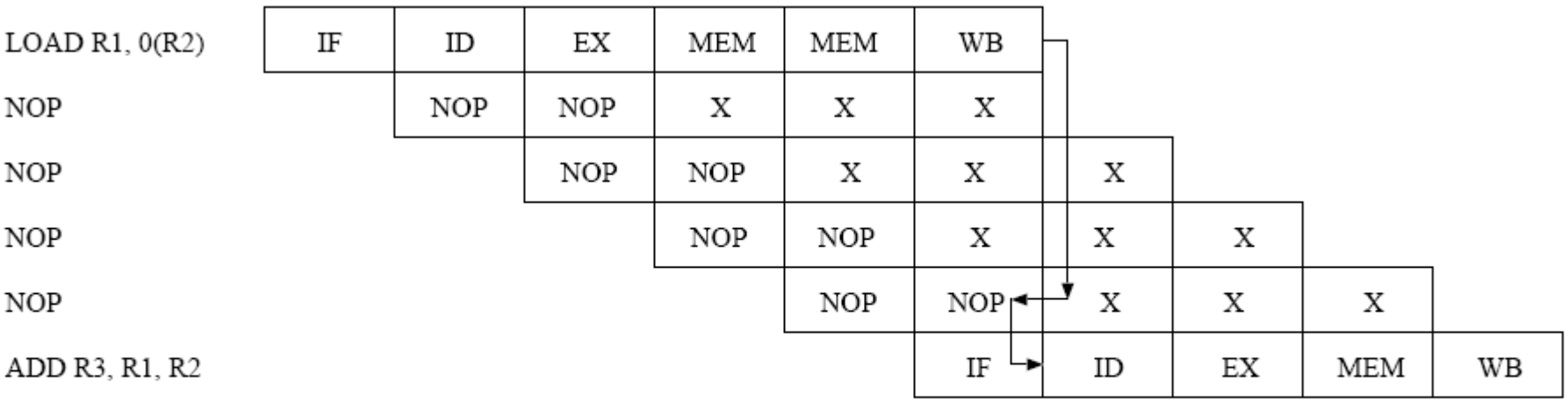


Abb. 6.7. Delayed Load aus Abb. 6.6, falls *kein* Bypass vorhanden wäre.

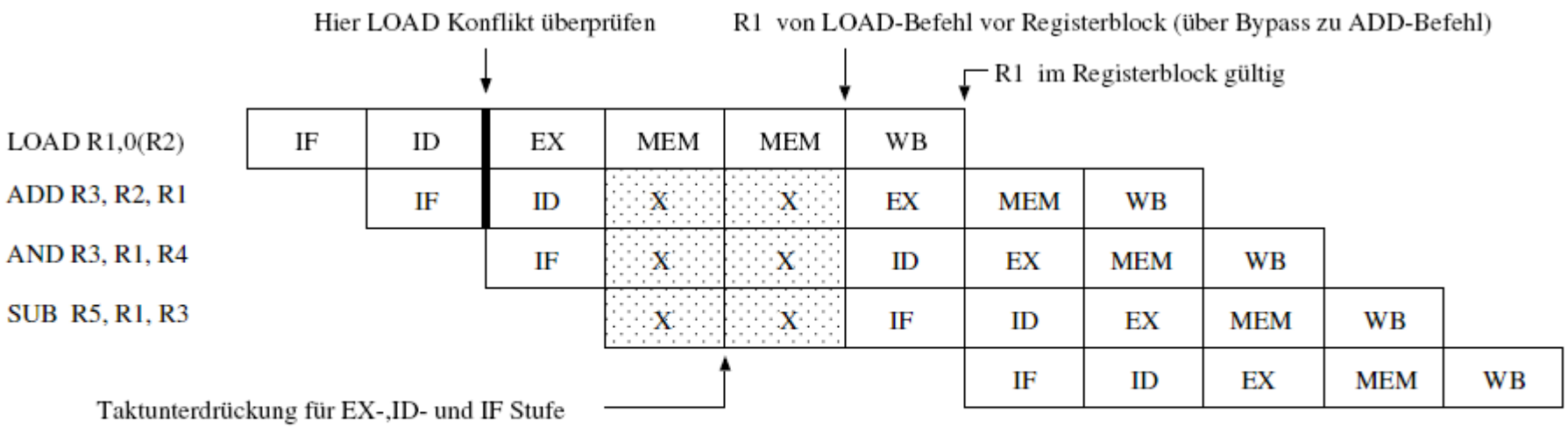


Abb. 6.8. Behebung eines dynamischen Laufzeitkonflikt mittels LOAD Interlock Hardware

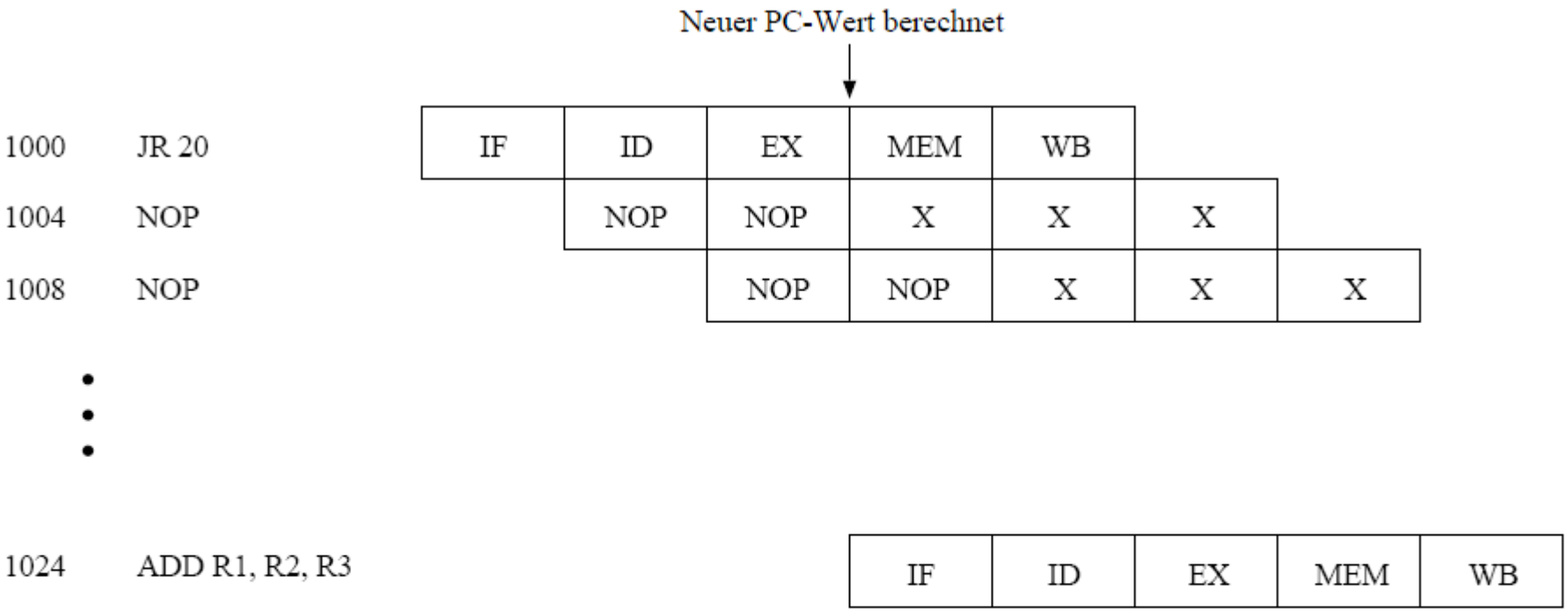


Abb. 6.9. Beispiel für die Behebung eines Steuerflusskonflikts durch Einfügen von NOPs. JR 20 sei ein Sprungbefehl (Jump Relative), der relativ zum aktuellen *PC*–Wert (hier 1004) mit der im Adressfeld angegeben Verschiebung (hexadezimal 20) verzweigt.

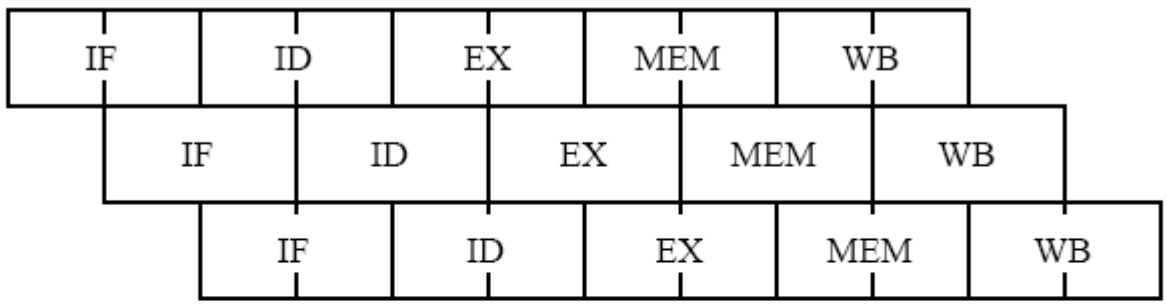


Abb. 6.10. Prinzip des Superpipelining: Durch eine feinere Aufteilung der Pipelineschritte kann die Tiefe der Pipeline vergrößert werden. Gleichzeitig erhöht sich auch die Taktrate.

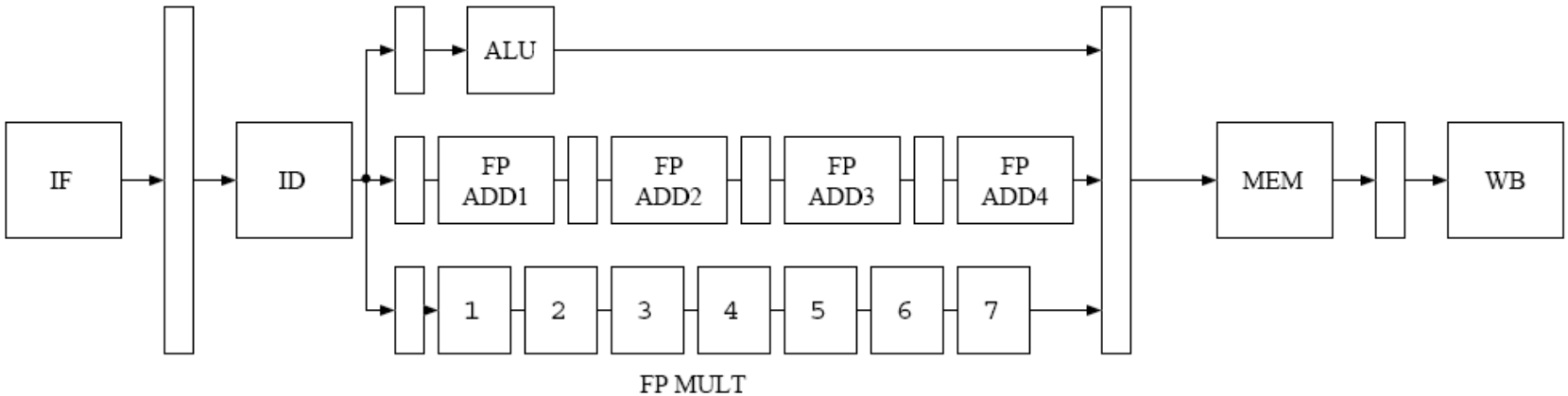


Abb. 6.11. RISC–Prozessor mit zwei arithmetischen Pipelines für Gleitkommaoperationen. Die Pipelineregister der Multiplikationseinheit wurden der Übersicht halber weggelassen.

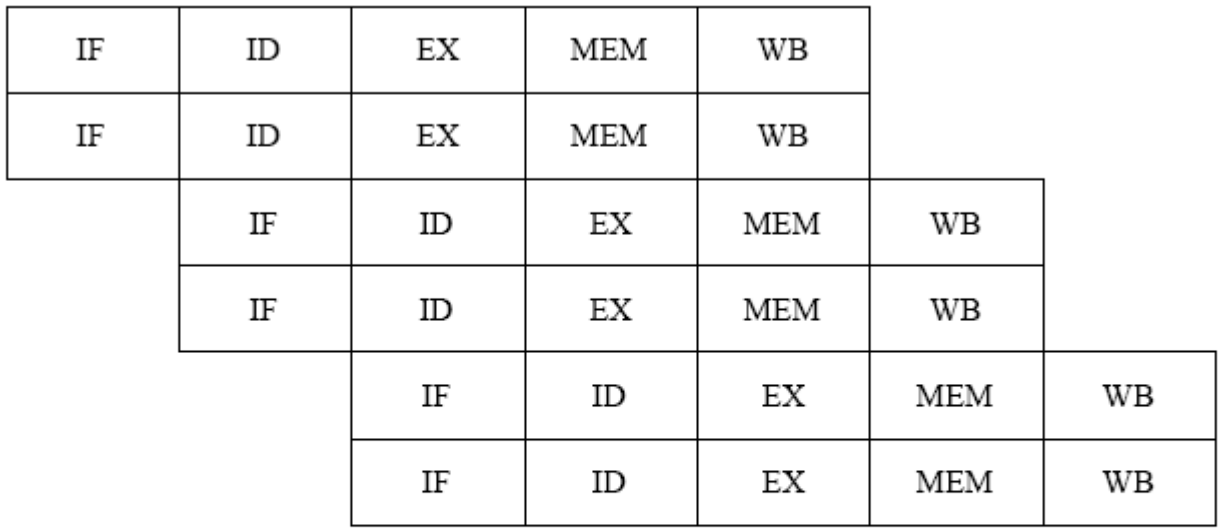


Abb. 6.12. Befehlsverarbeitung bei einem superskalaren RISC–Prozessor. Weil gleichzeitig zwei Befehle bearbeitet werden, ist eine hohe Speicherbandbreite erforderlich.

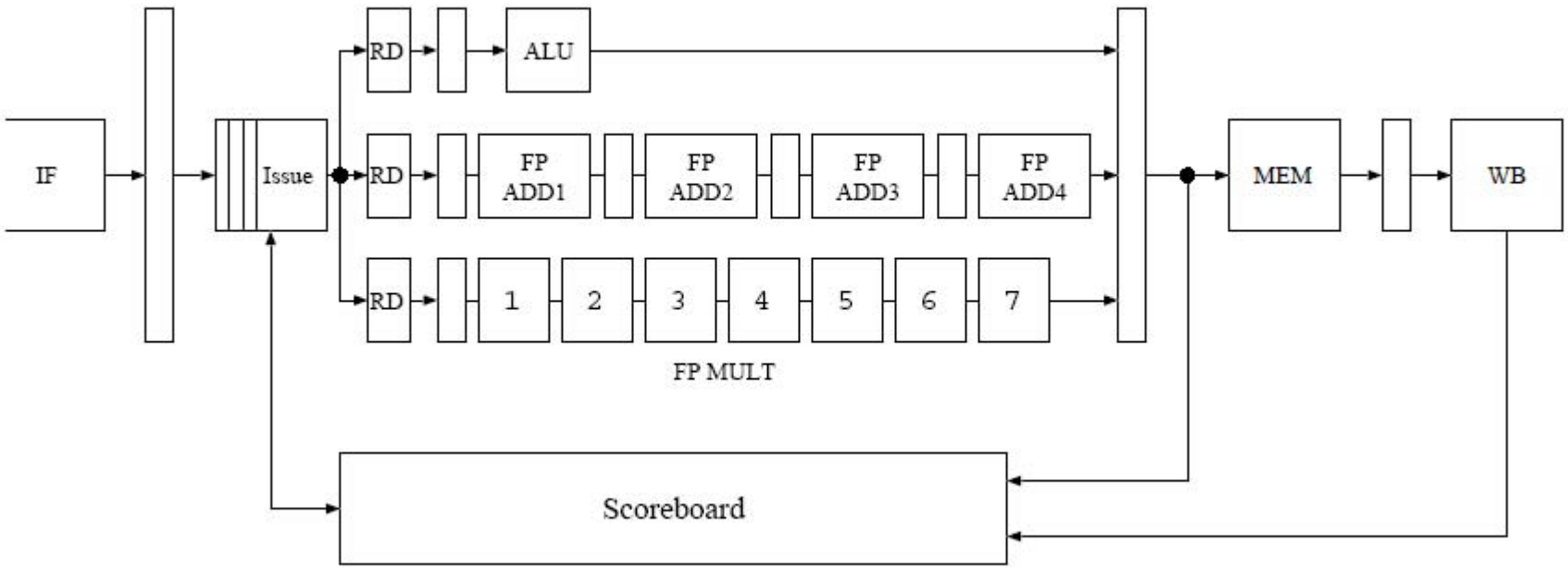


Abb. 6.13. Dynamisches Befehlsscheduling mit Hilfe eines Scoreboards

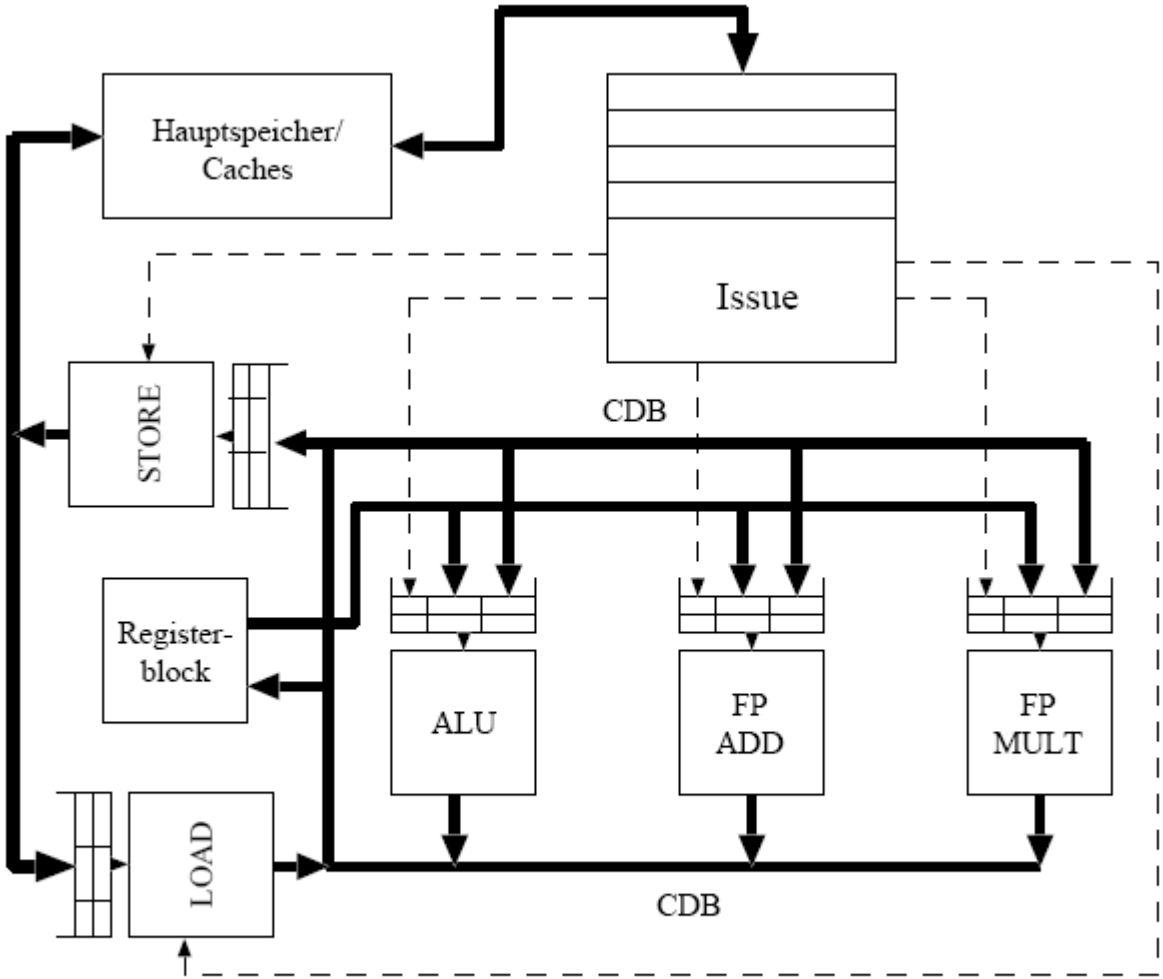


Abb. 6.14. Schematische Darstellung eines superskalaren Prozessors mit Reservierungsstationen

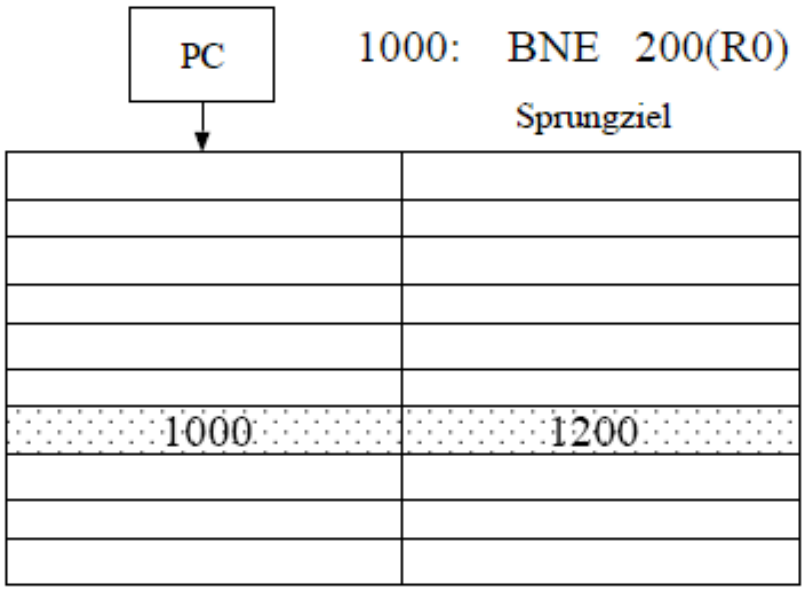


Abb. 6.15. Sprungzielspeicher zur Minimierung von Steuerflusskonflikten. Für den angegebenen Beispielbefehl wurde ein Sprungziel gefunden.

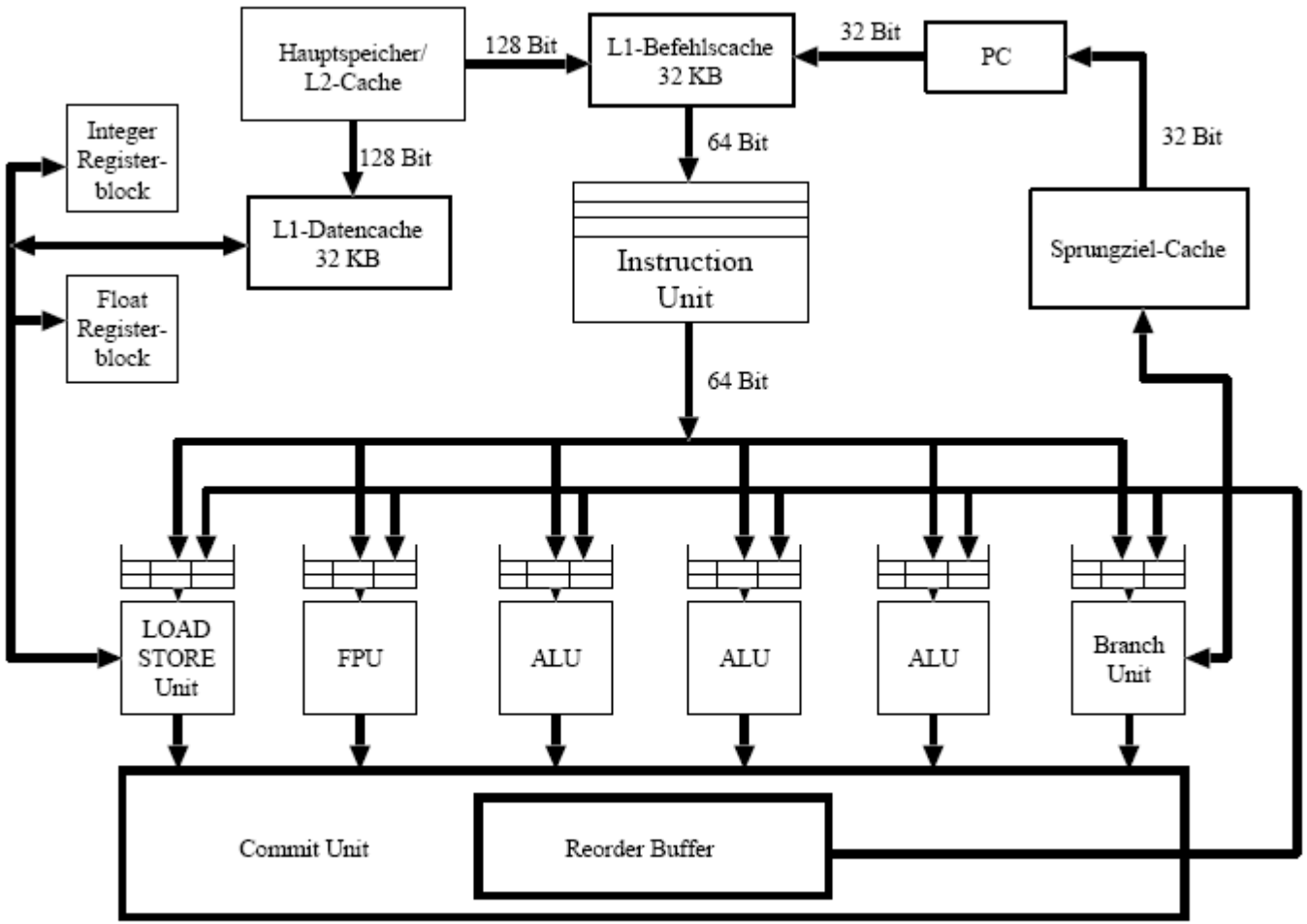


Abb. 6.16. Schematischer Aufbau des superskalaren PowerPC 620 RISC–Prozessors

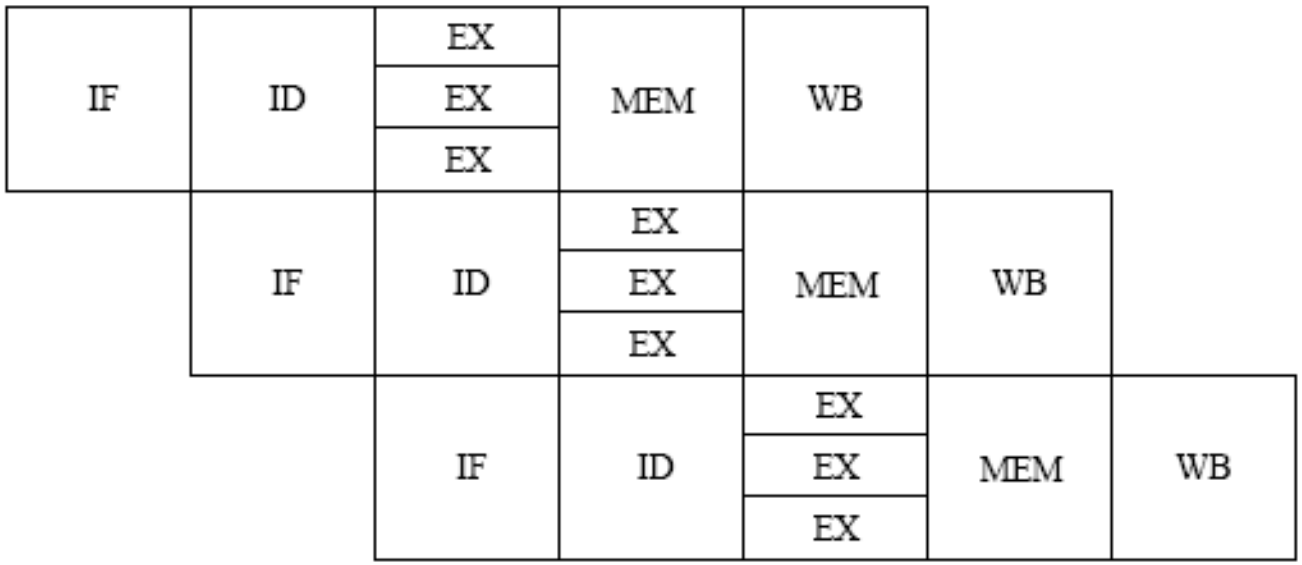


Abb. 6.17. Befehlsabarbeitung bei einem VLIW–Prozessor

7. Kommunikation

1. metallische Schichten auf Mikrochips (bis 200 MBit/s)
2. geätzte Leiterbahnen auf Platinen (bis 100 MBit/s)
3. Flachbandkabel (bis 10 MBit/s)

} Komponentenebene

4. symmetrische Kabel (1–5 MBit/s)
5. Koaxialkabel (10–100 MBit/s)
6. Lichtwellenleiter (1 GBit/s)
7. Satellitenverbindung (64 KBit/s)

} Systemebene

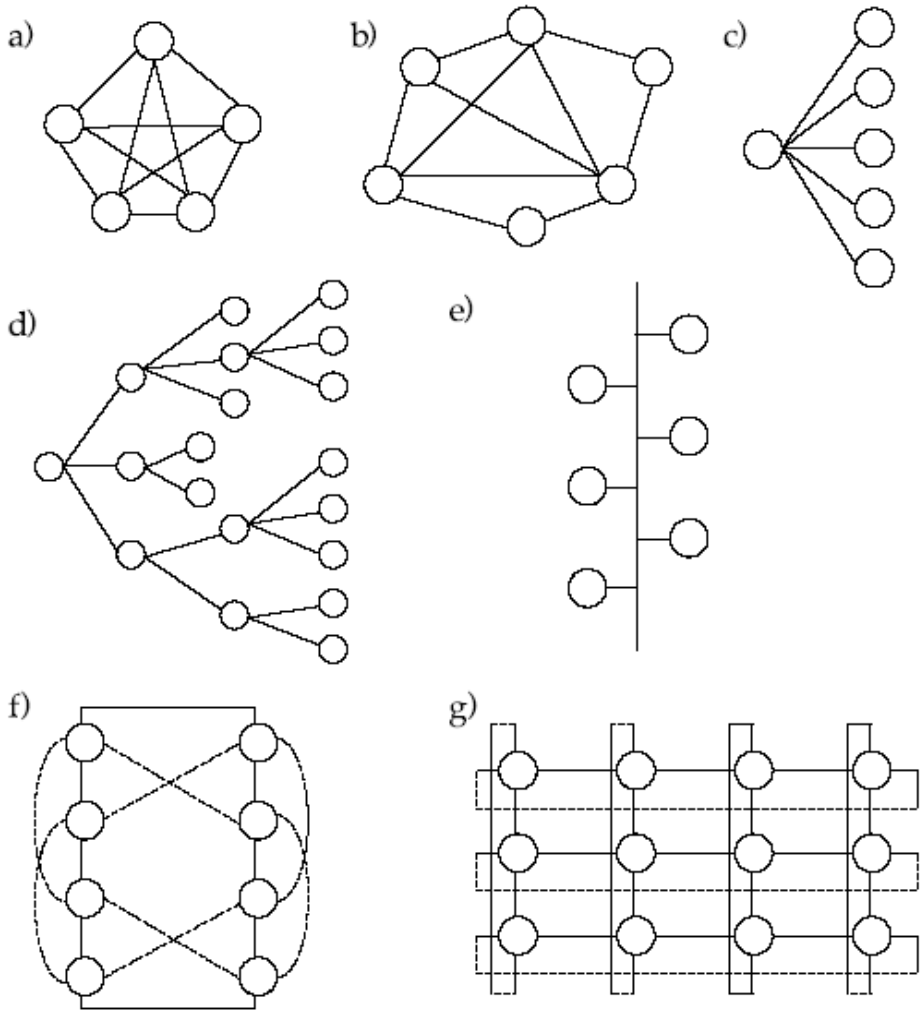


Abb. 7.1. Verschiedene Topologien: a) Vollständiger Graph b) Vermaschtes Netz c) Stern d) Baum e) Bus f) Ring (gestrichelt: Verzopfung) g) Gitter (gestrichelt: Torus)

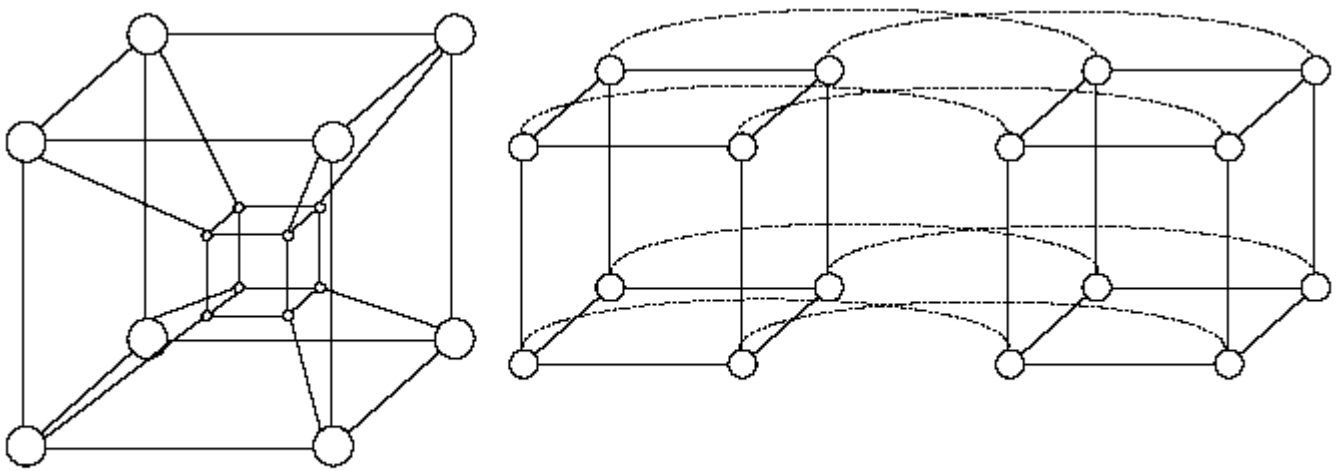


Abb. 7.2. Vierdimensionaler Hypercube in zwei verschiedenen Darstellungen

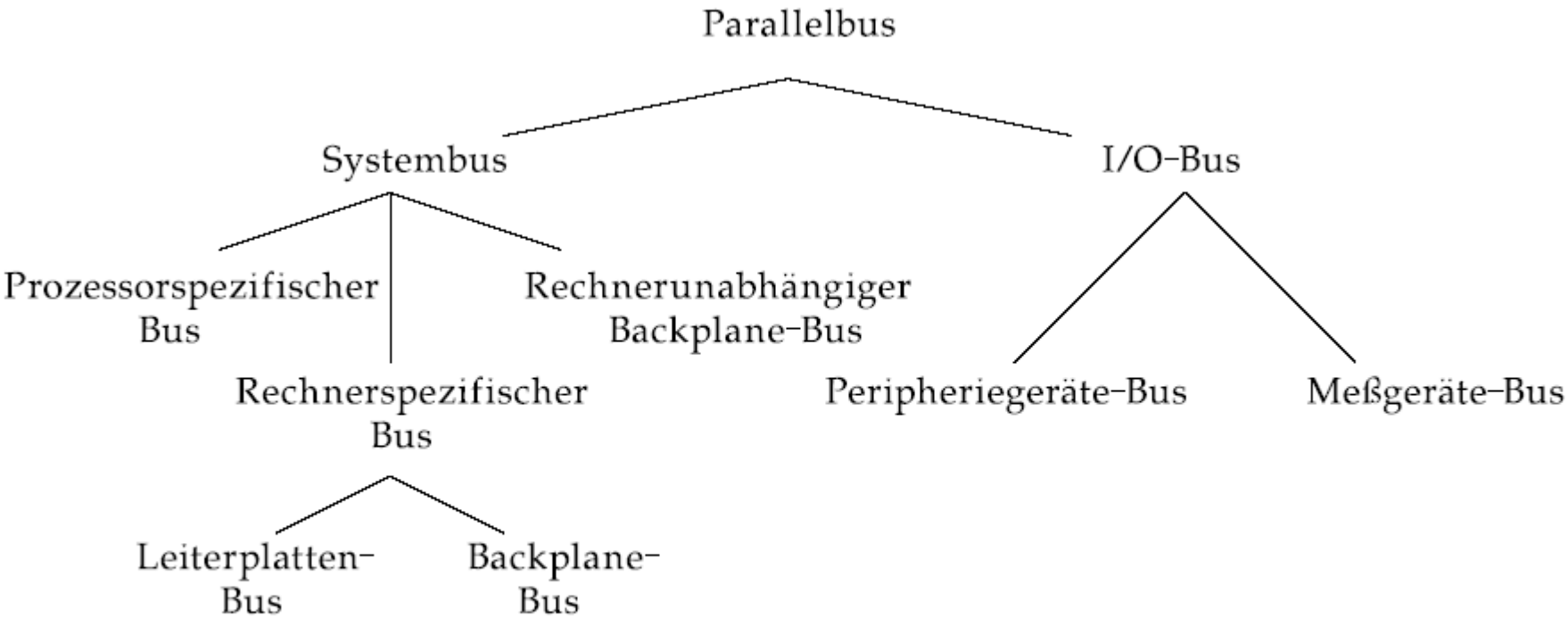


Abb. 7.3. Übersicht über verschiedene Arten von Parallelbussen

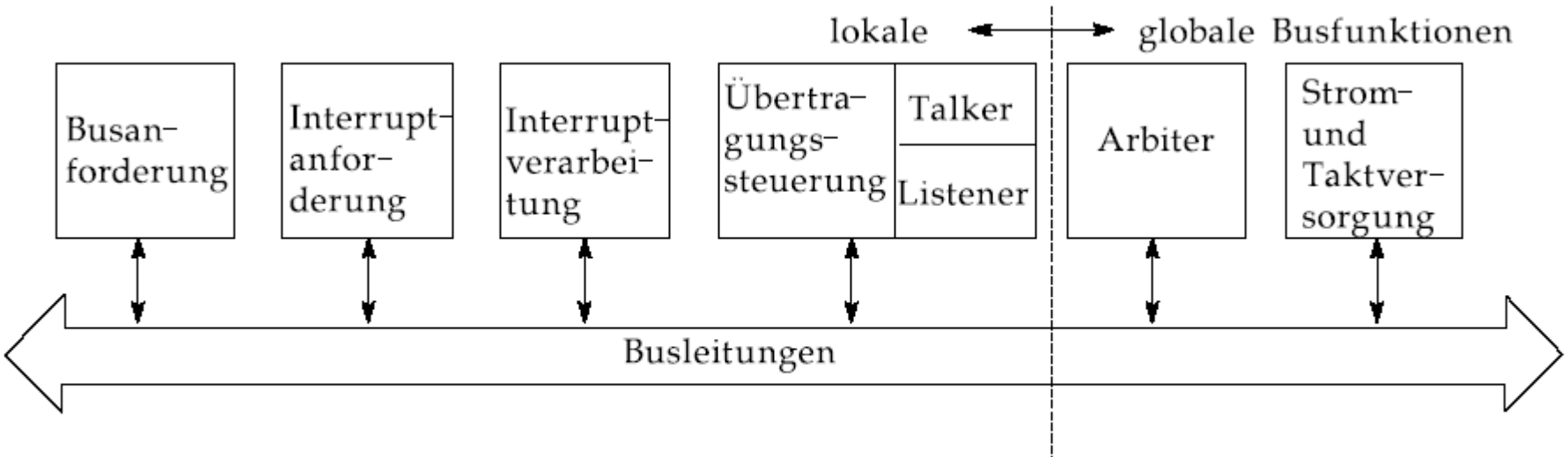


Abb. 7.4. Komponenten eines Businterfaces und globale Busfunktionen

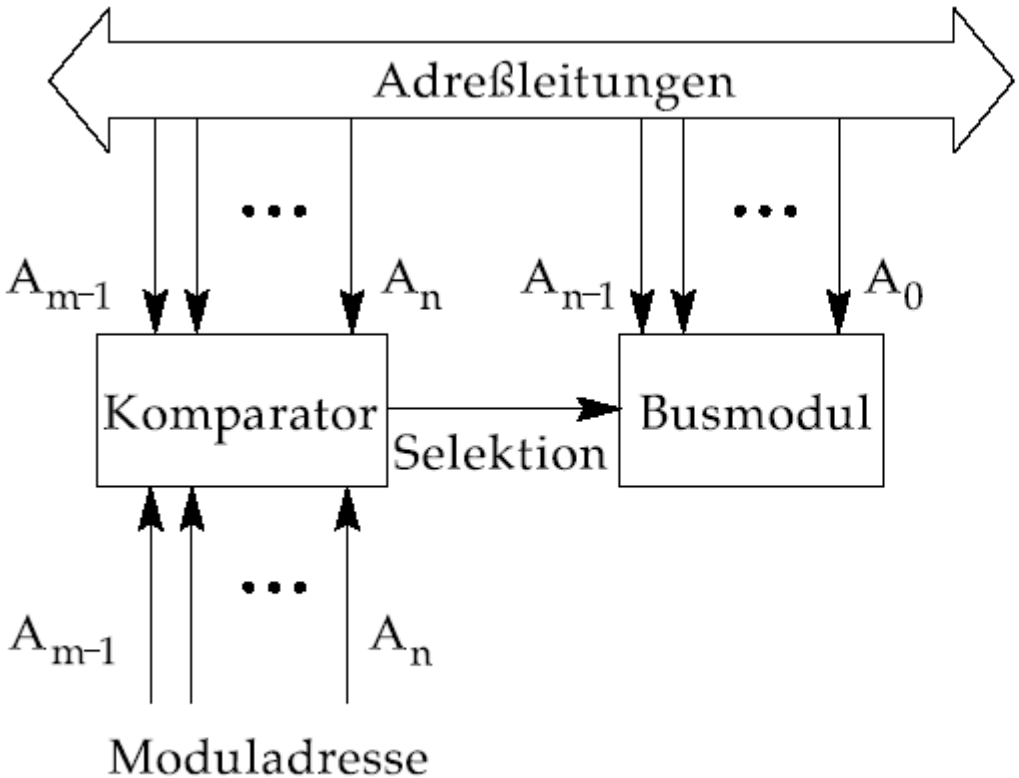


Abb. 7.5. Selektion eines Busmoduls mit 2^n großem Adressbereich.

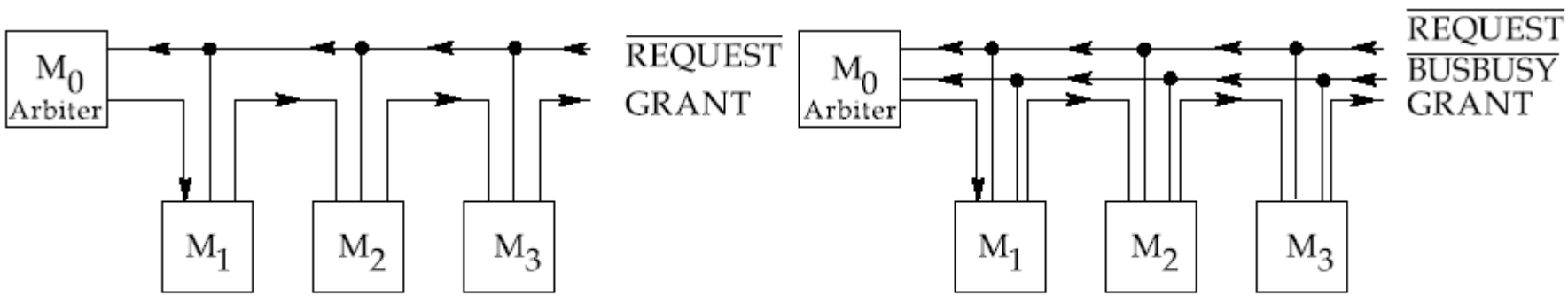


Abb. 7.6. Aufbau einer zentralen Daisy–Chain Busarbitrierung (links: 2–Draht Protokoll; rechts: 3–Draht Protokoll)

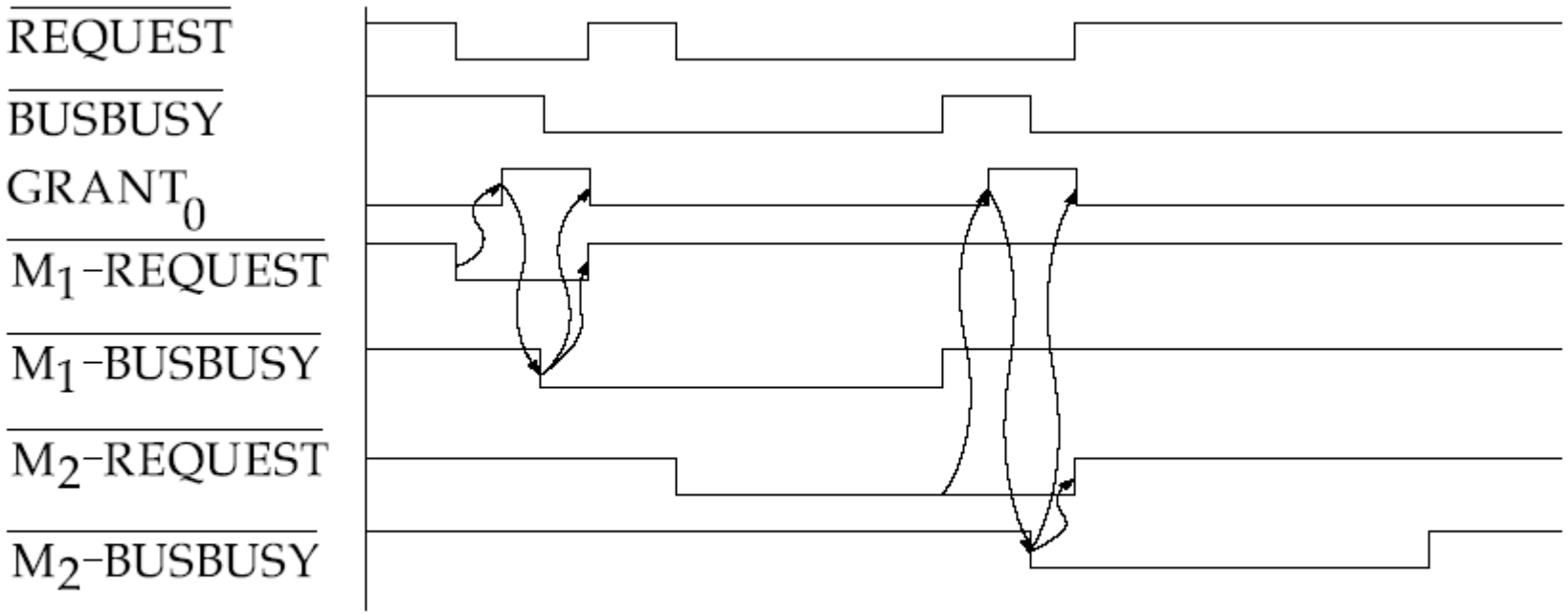


Abb. 7.7. Zeitdiagramm der Bus–Arbitrierung mit dem Drei–Draht Protokoll

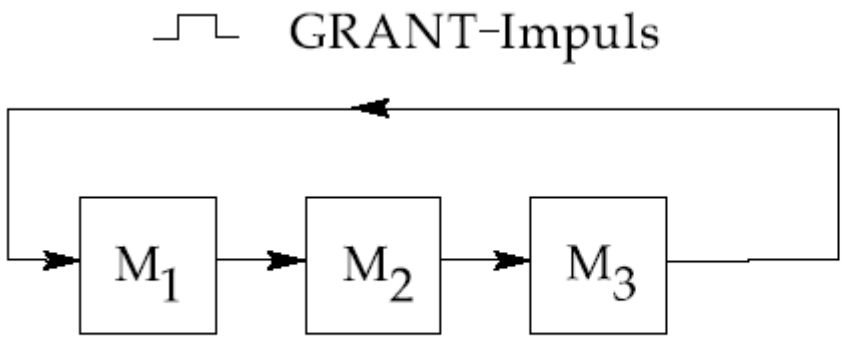


Abb. 7.8. Busarbitrierung mit dezentraler Daisy–Chain

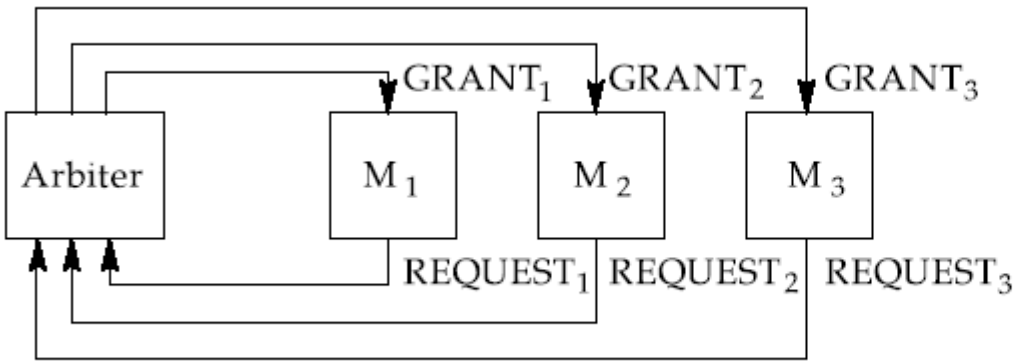


Abb. 7.9. Zentraler Arbiter mit Stichleitungen

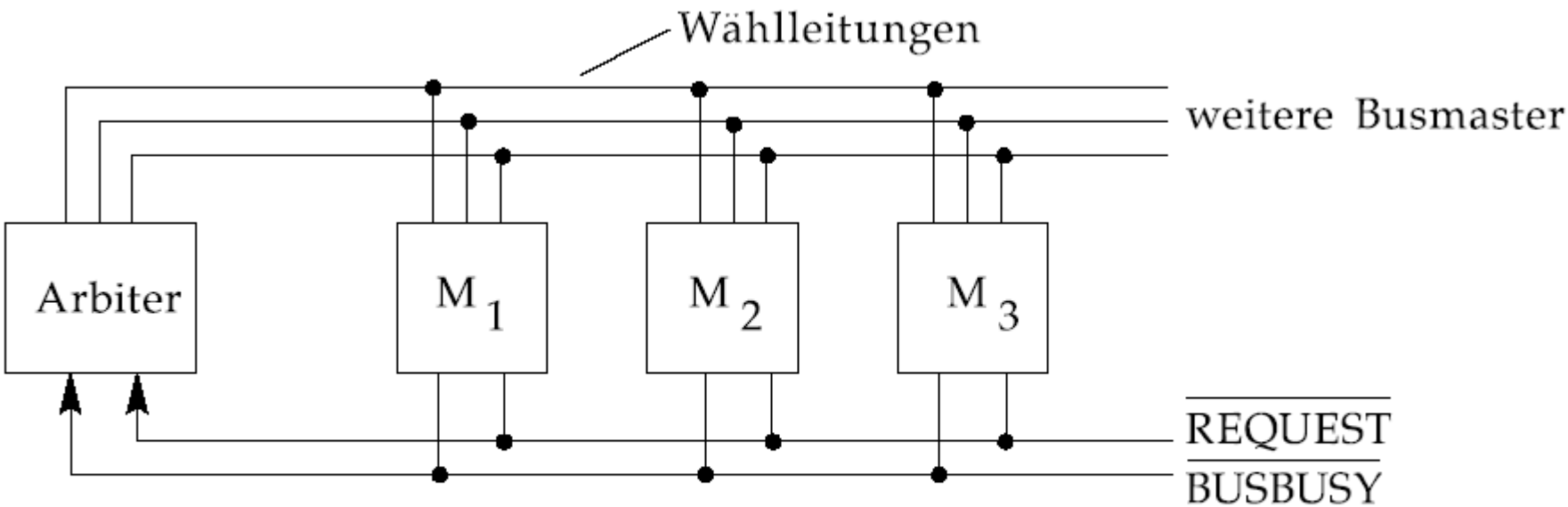


Abb. 7.10. Zentraler Arbiter mit Polling

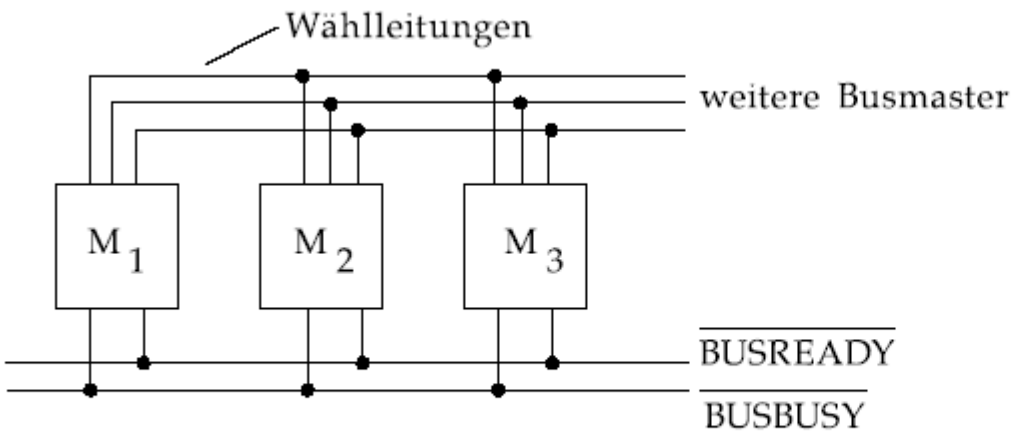


Abb. 7.11. Dezentrale Arbitrierung mit Polling

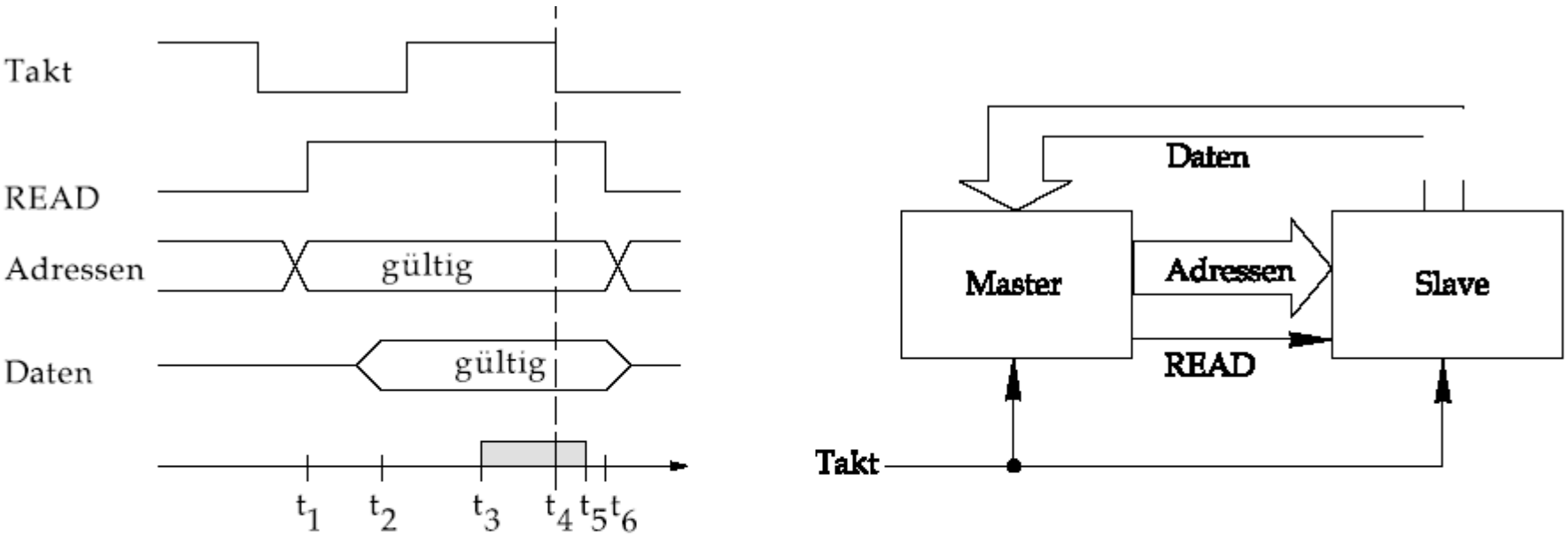


Abb. 7.12. Leseoperation bei synchroner Datenübertragung

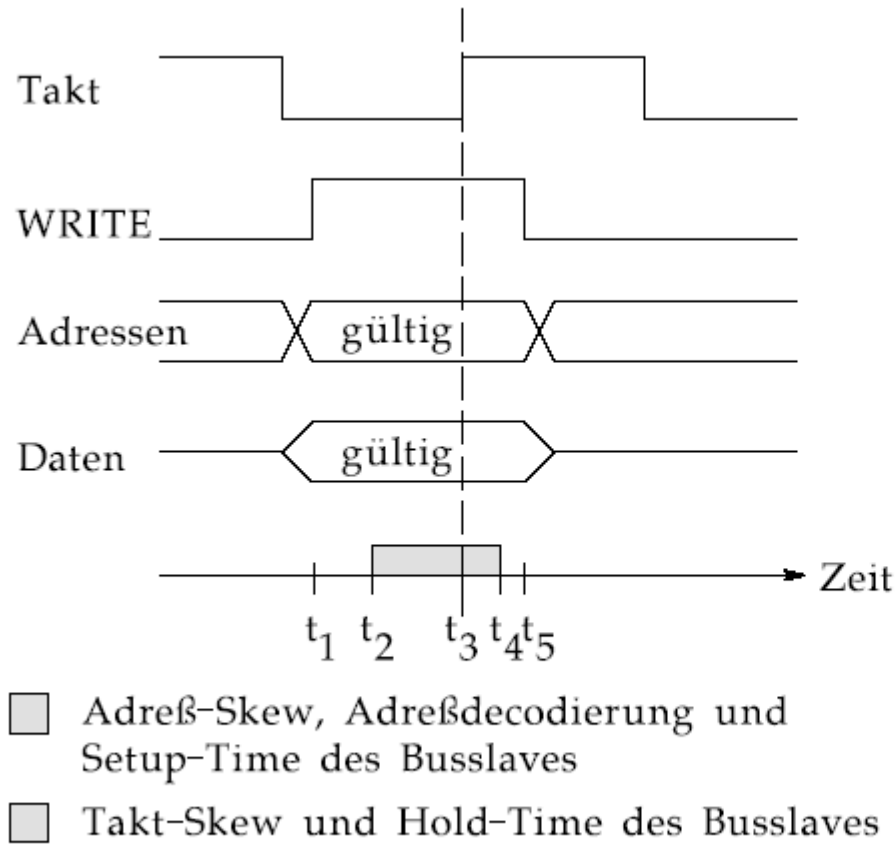


Abb. 7.13. Schreiboperation bei synchroner Datenübertragung

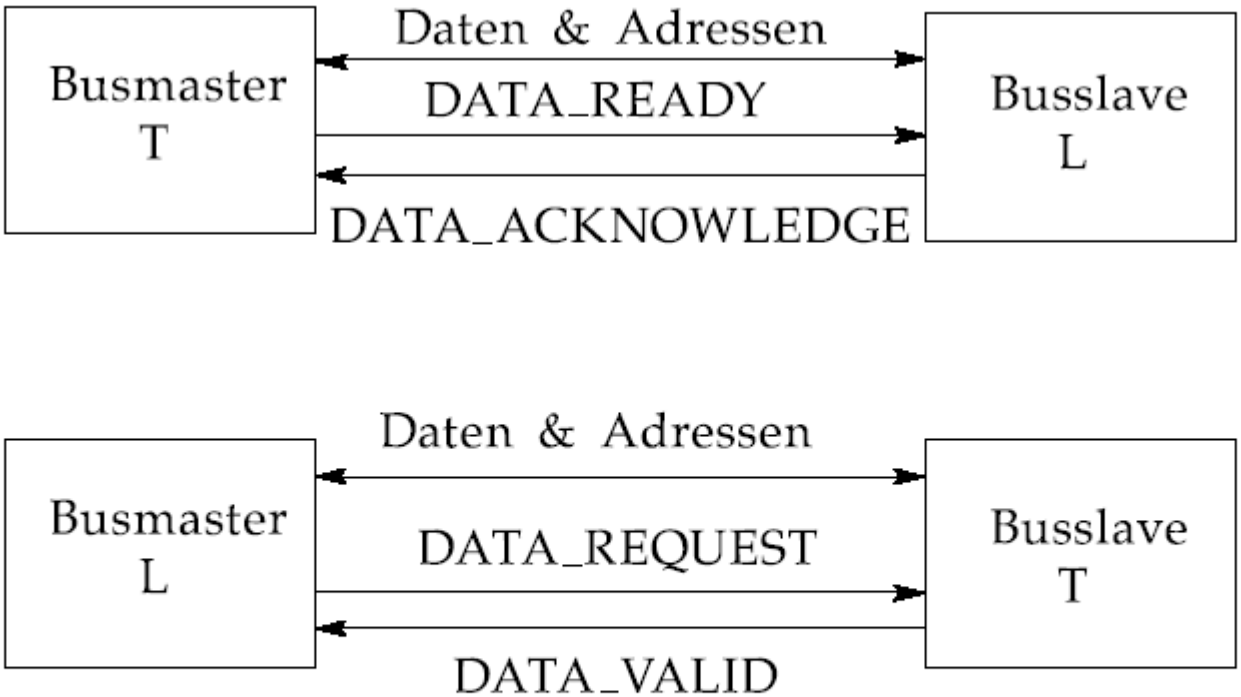


Abb. 7.14. Bedeutung der Handshake–Signale bei asynchroner Datenübertragung (oben: Talker–gesteuert; unten: Listener–gesteuert)

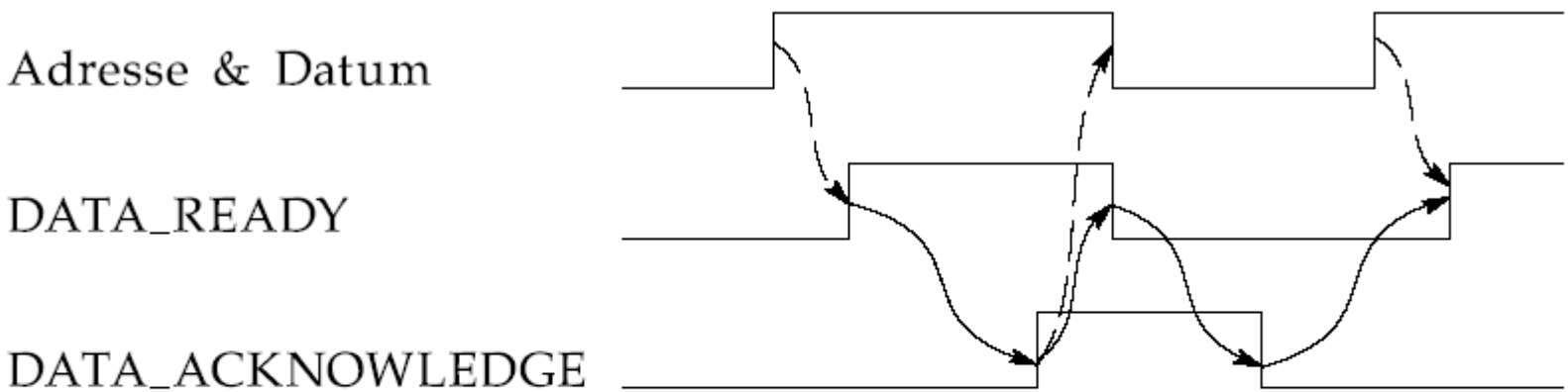


Abb. 7.15. Vollverschränktes (fully interlocked) Protokoll zur asynchronen Datenübertragung

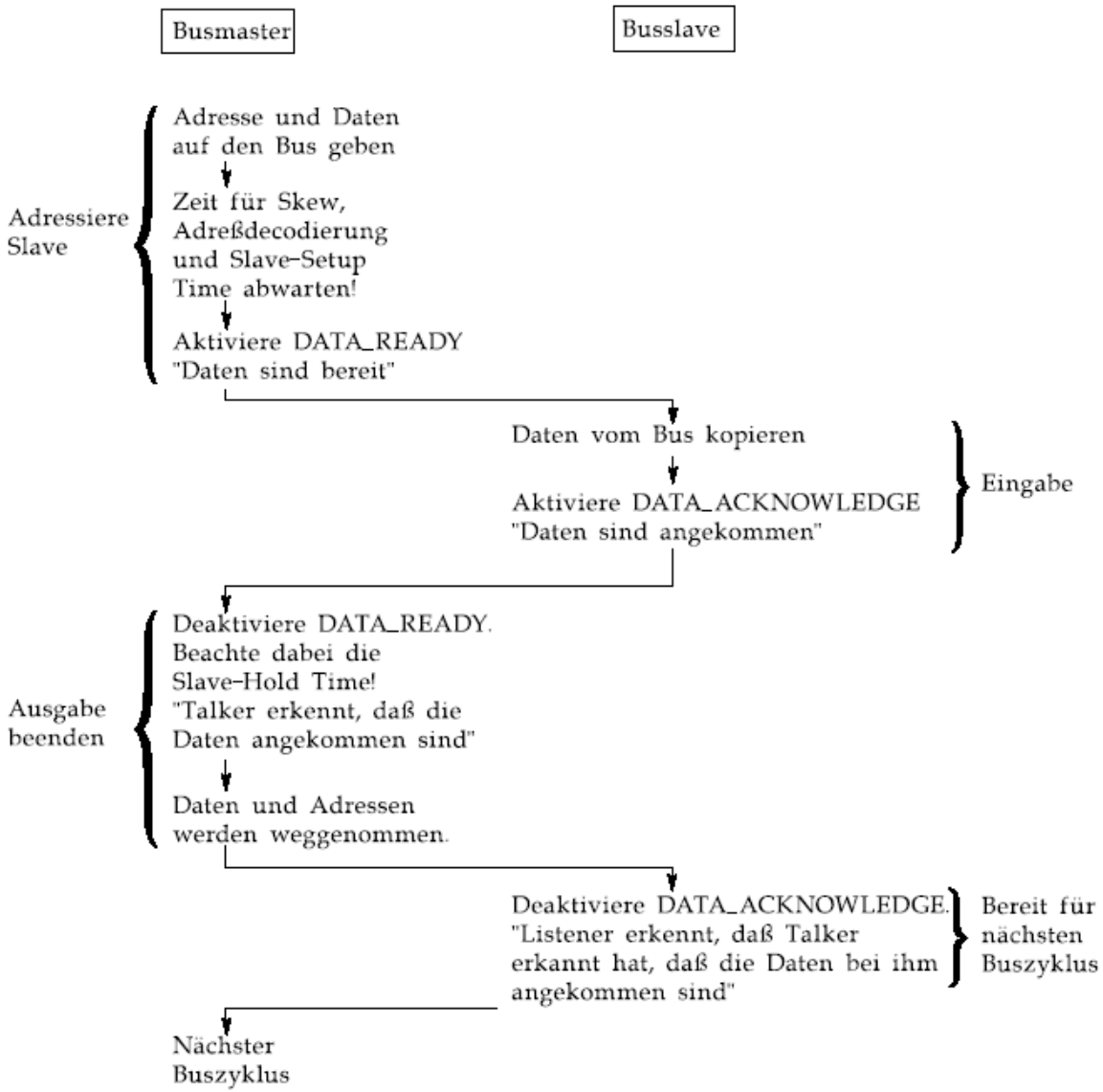


Abb. 7.16. Ablaufdiagramm der Interaktion von Busmaster und Busslave beim vollverschränkten asynchronen Protokoll.

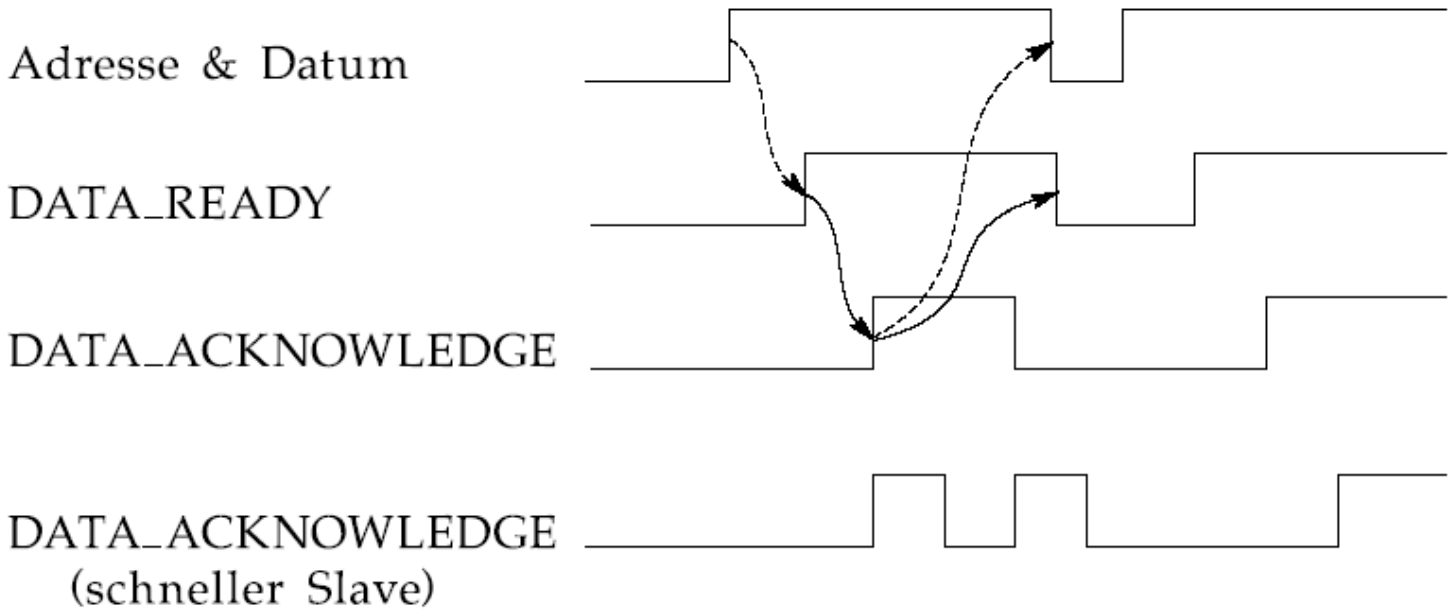


Abb. 7.17. Halbverschränktes (half interlocked) Protokoll zur asynchronen Datenübertragung. Unten: Datenvervielfachung bzw. Übertragungsfehler bei schnellem Busslave.

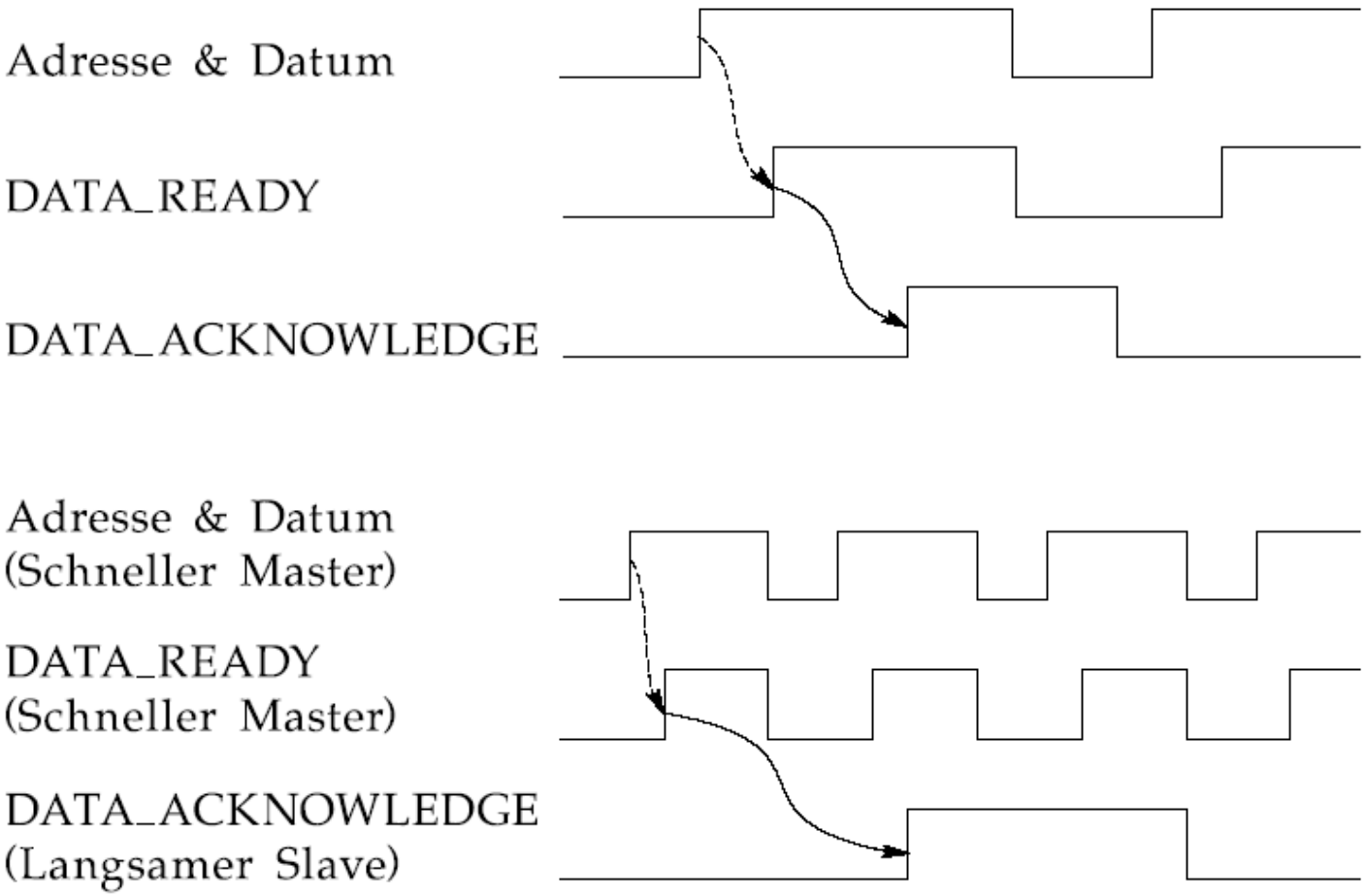


Abb. 7.18. Oben: Nichtverschränktes (non interlocked) Protokoll zur asynchronen Datenübertragung. Unten: Datenverlust bzw. Übertragungsfehler bei schnellem Busmaster.

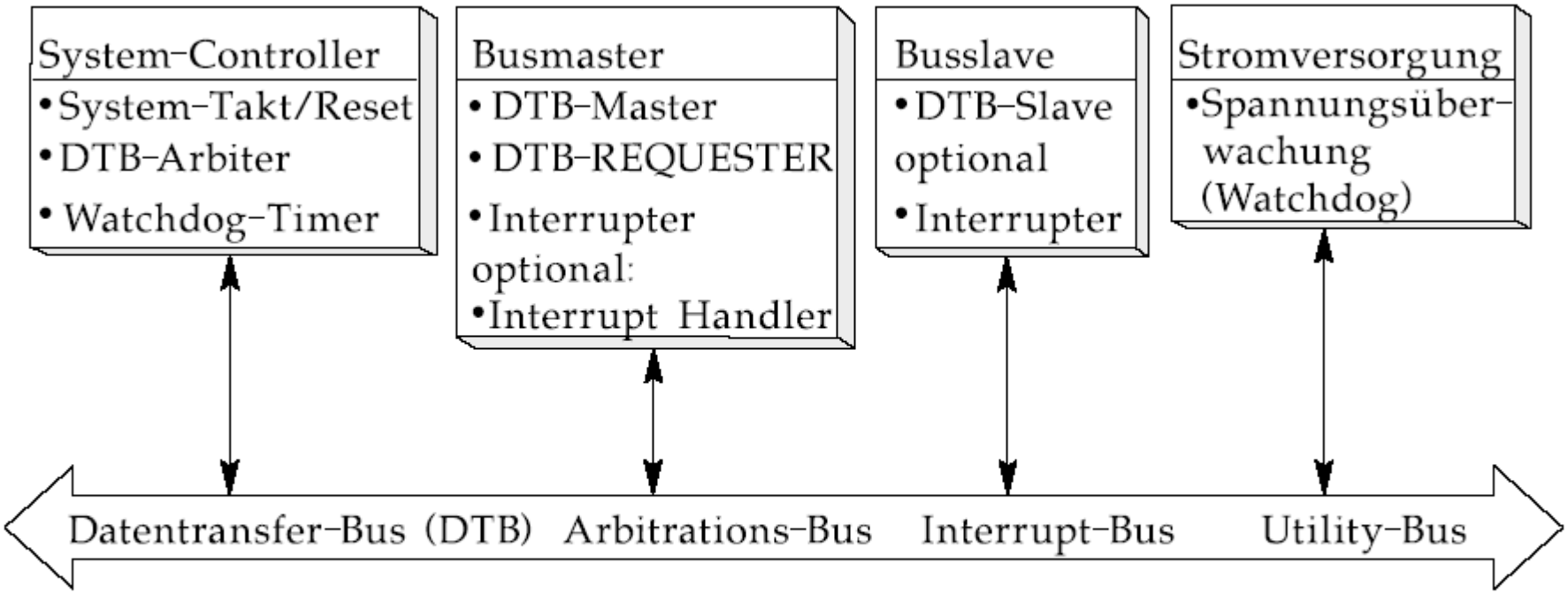


Abb. 7.19. Aufbau des VME–Bus und mögliche Busfunktionen der Module

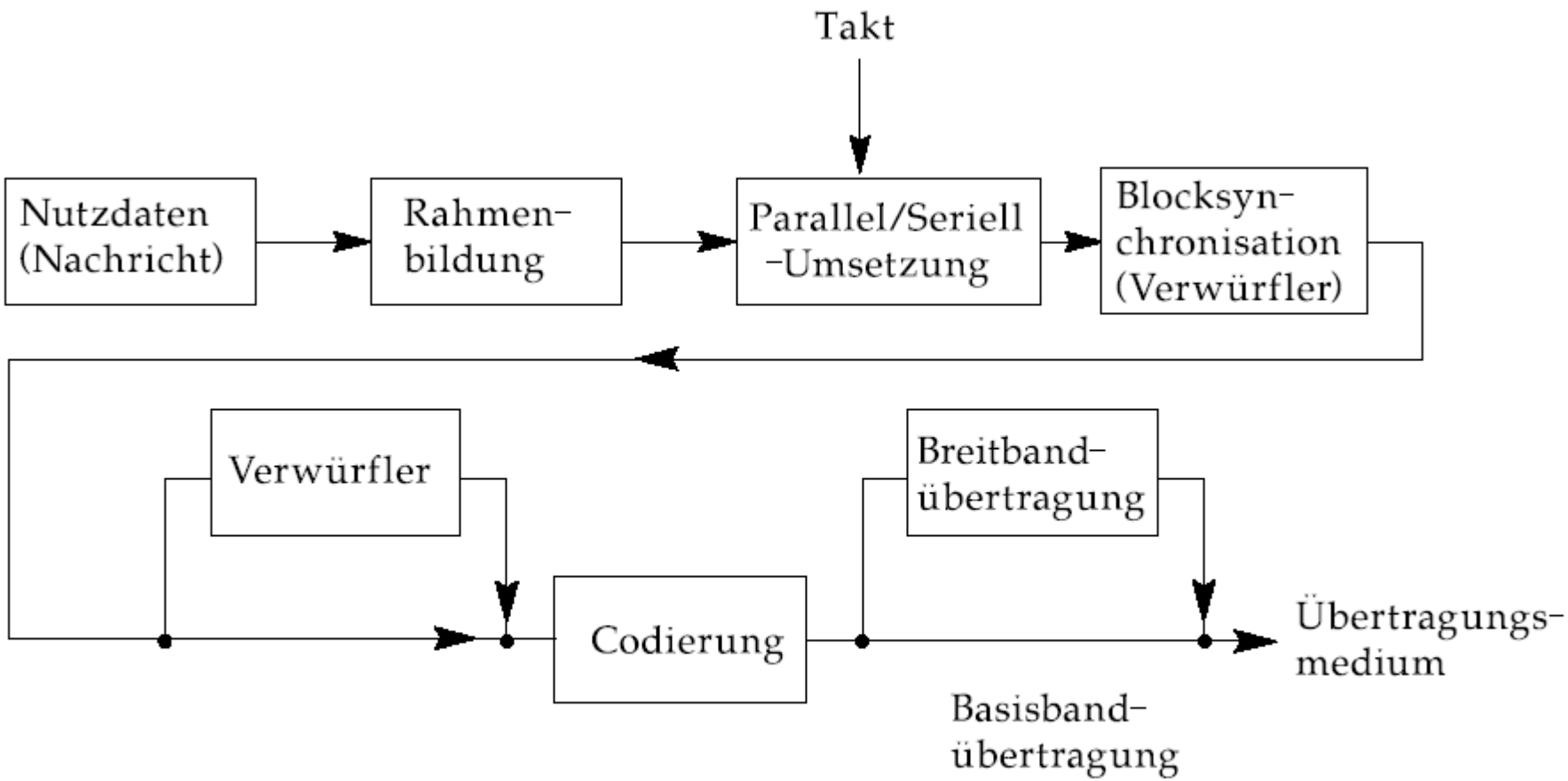


Abb. 7.20. Blockschaltbild der seriellen Übertragungskette im Sendeteil

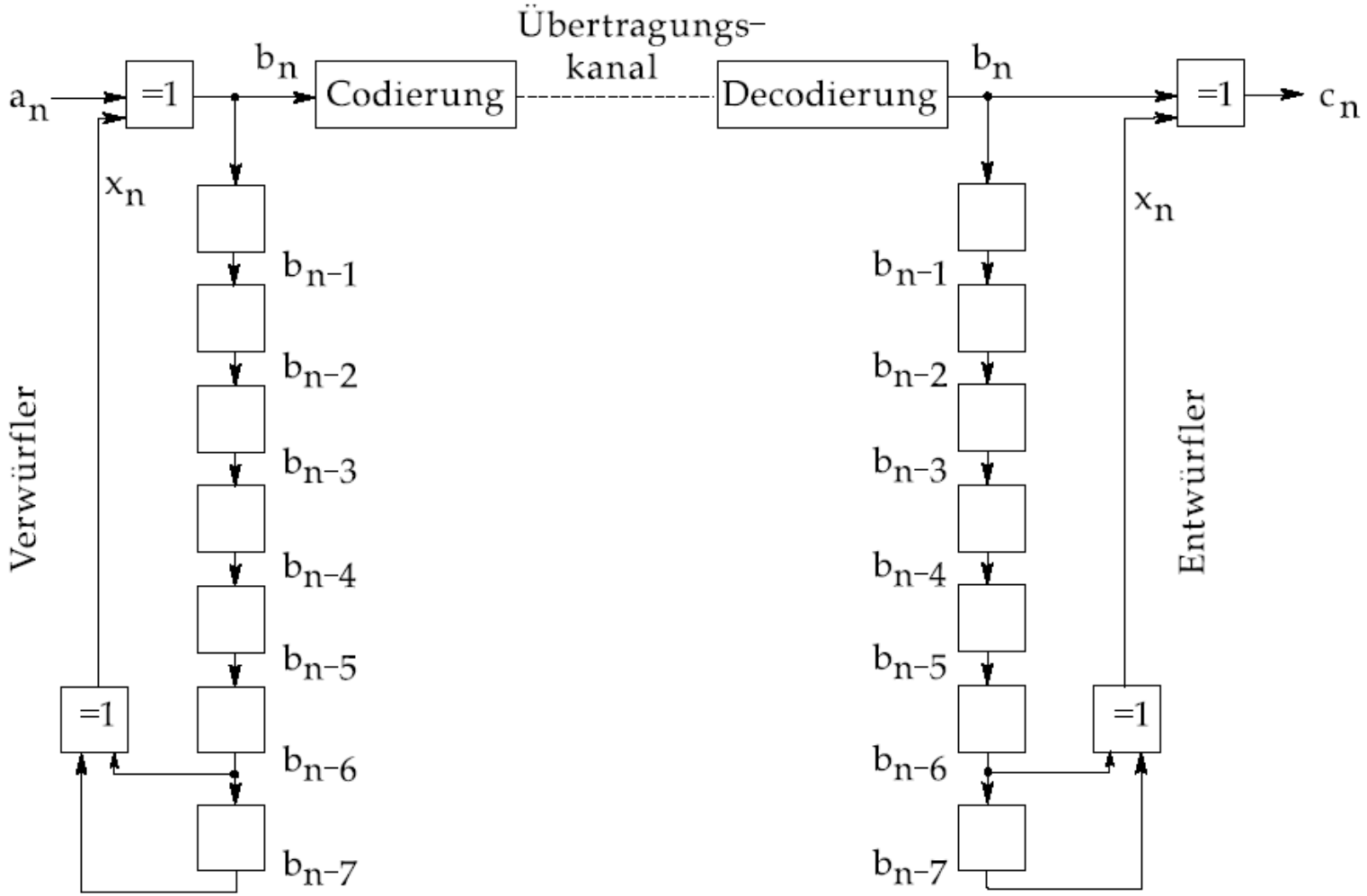
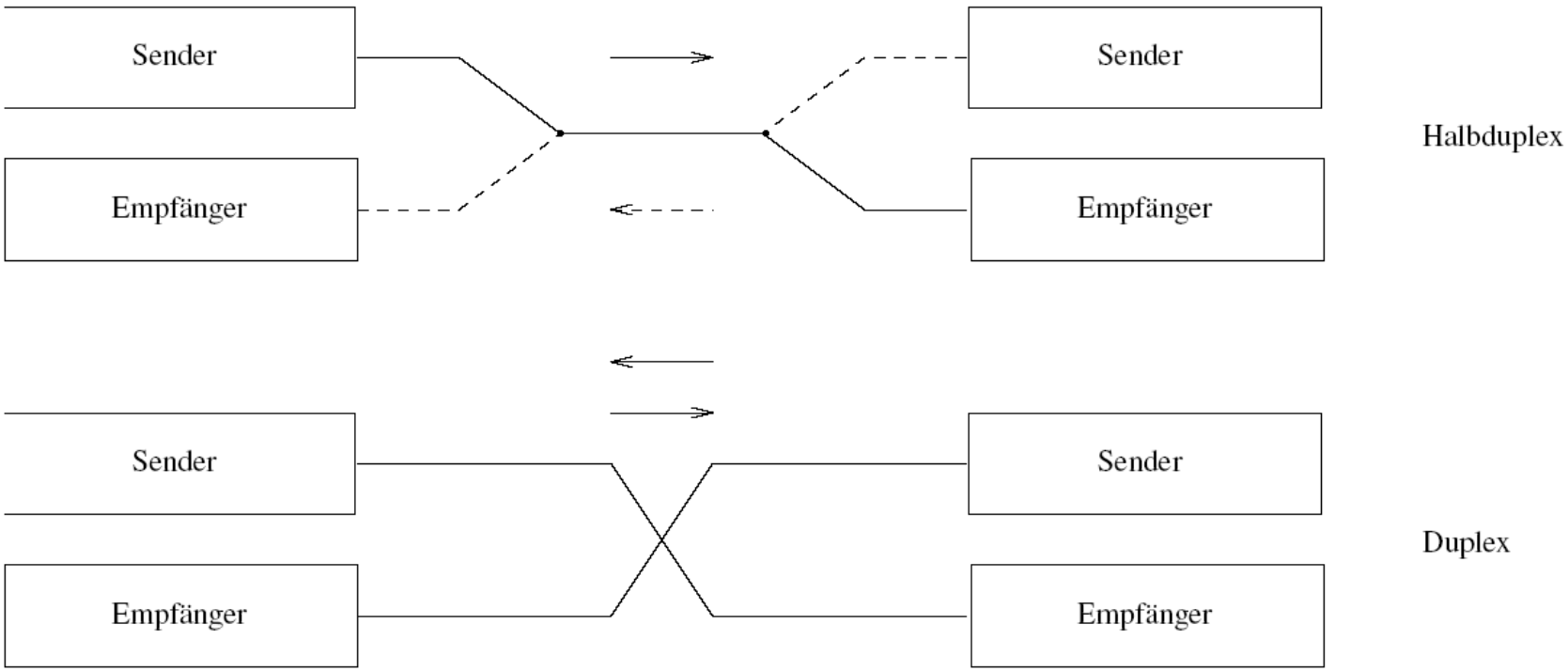


Abb. 7.21. Verwürfler und Entwürfler sind symmetrisch aufgebaut. Sie ermöglichen eine Synchronisierung des Empfängertaktes bei Verwendung einfacher Leitungscodes.



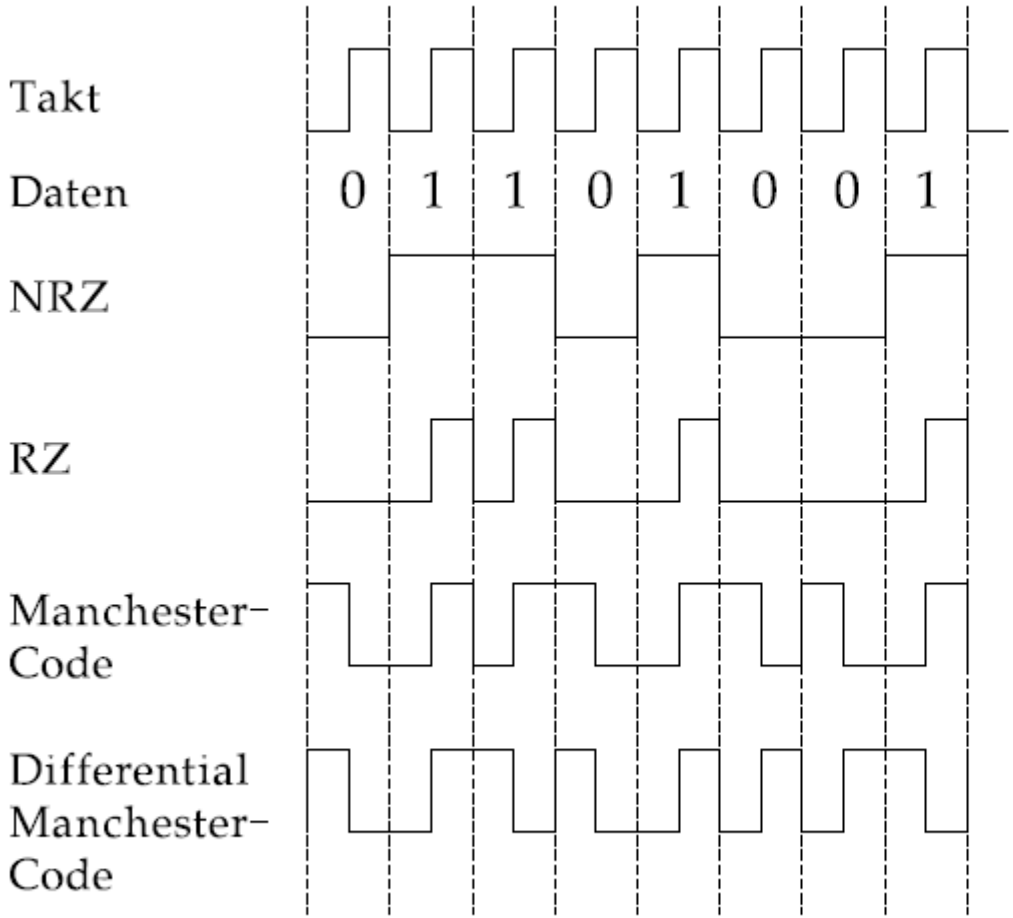


Abb. 7.23. Verschiedene binäre Leitungscodes

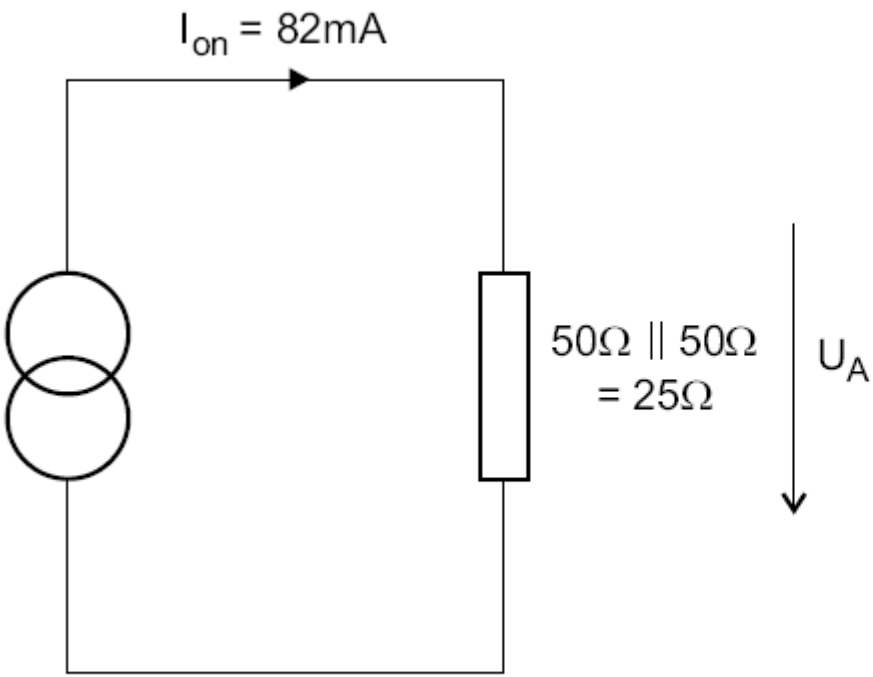


Abb. 7.24. DC–Schwellwert–Erkennung bei Überlagerung zweier Sender im Ethernet

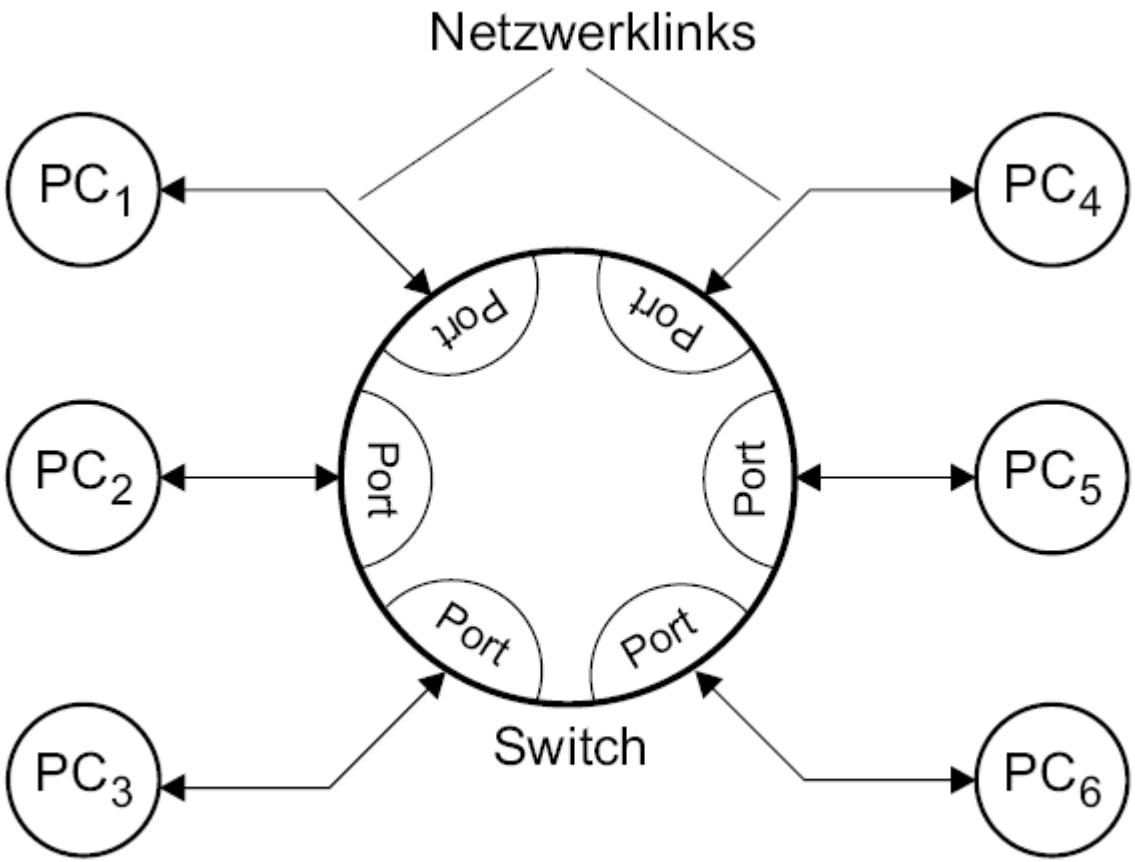


Abb. 7.25. Ethernet–Netzwerk mit Switch

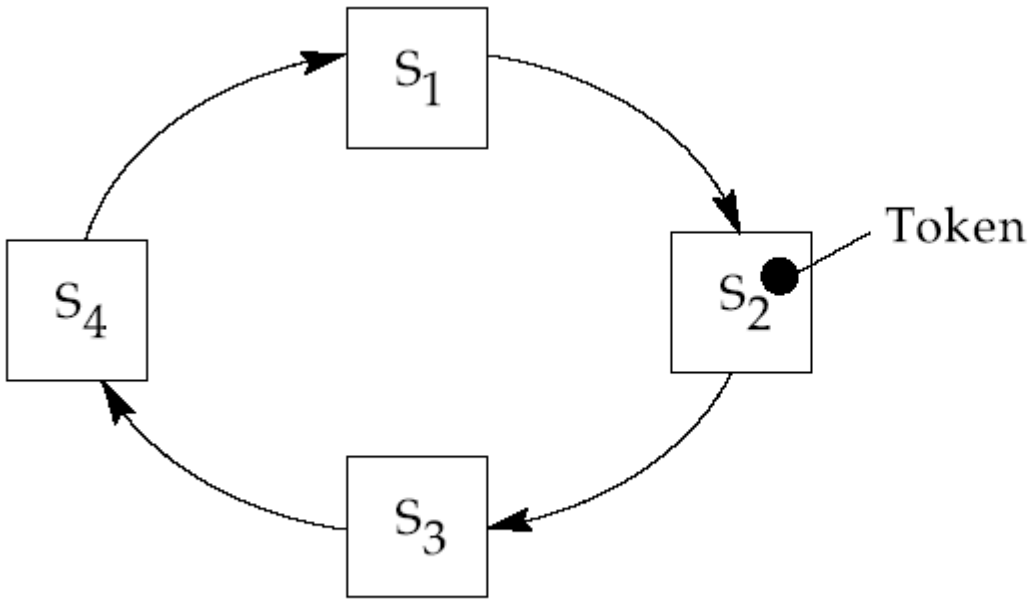


Abb. 7.26. Aufbau eines Token–Ring LAN

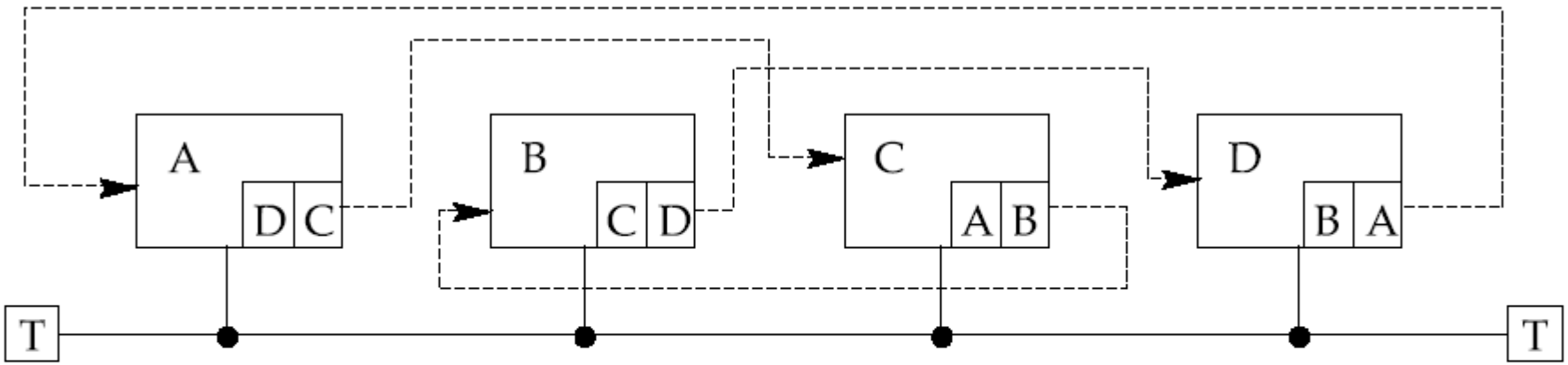


Abb. 7.27. Aufbau eines Token–Bus: logischer Ring durch Verweise auf Nachfolgestation (T = Abschlusswiderstände)

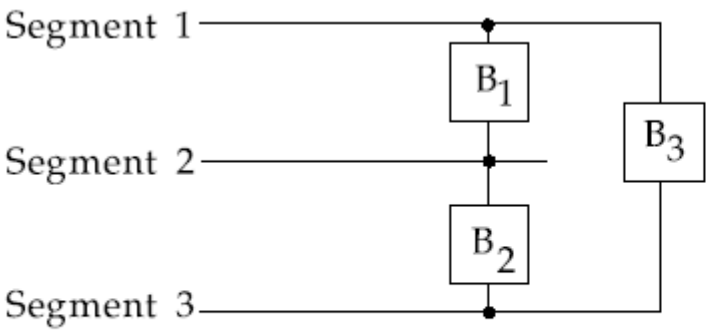


Abb. 7.28. Die redundante Brücke B_3 erhöht die Verfügbarkeit eines Ethernet–LANs. Stationen aus Segment 1 und Segment 3 können über B_3 verbunden werden, ohne den Netzbetrieb auf Segment 2 zu stören.

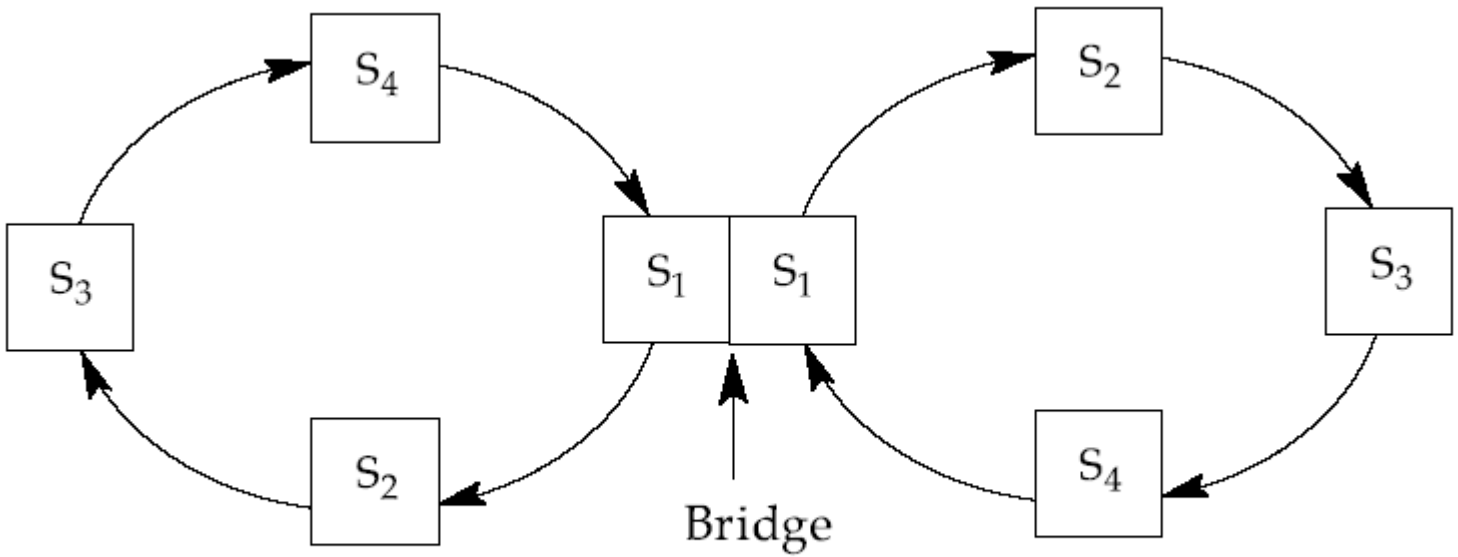


Abb. 7.29. Kopplung zweier Token–Ring LANs über eine Bridge

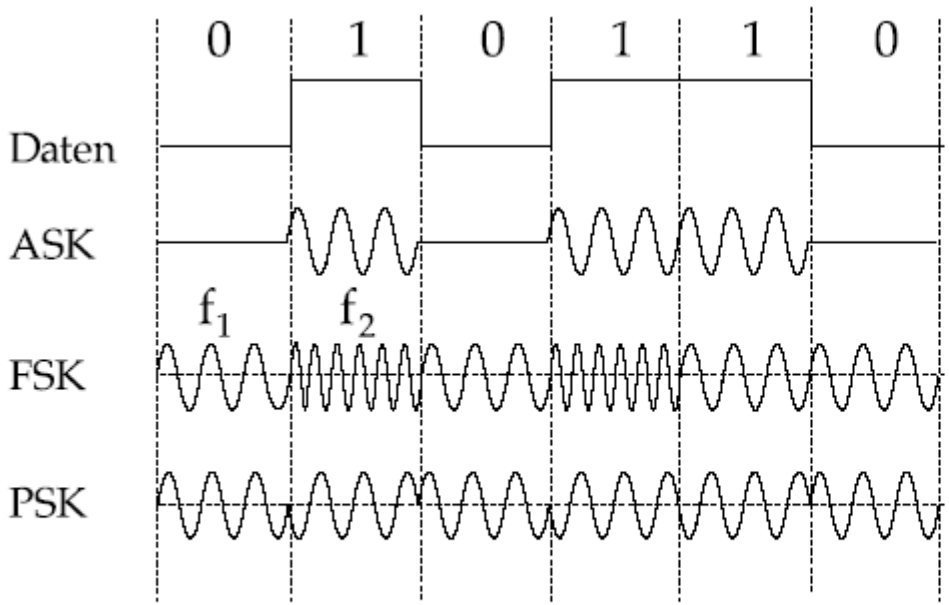


Abb. 7.30. Modulationsarten für die Breitbandübertragung: ASK = Amplitude Shift Keying, FSK = Frequency Shift Keying, PSK = Phase Shift Keying

									VRC
1. Zeichen	1	0	1	1	0	1	0	1	1
2. Zeichen	1	0	1	0	1	1	0	1	1
3. Zeichen	0	0	0	1	1	0	1	0	0
4. Zeichen	1	1	1	0	1	0	1	0	1
LRC	1	1	1	1	1	0	0	0	

Abb. 7.31. Übertragungssicherung mit gerader Quer– und Längsparität. Durch Kreuzsicherung können Einfachfehler (schattiert) behoben werden.

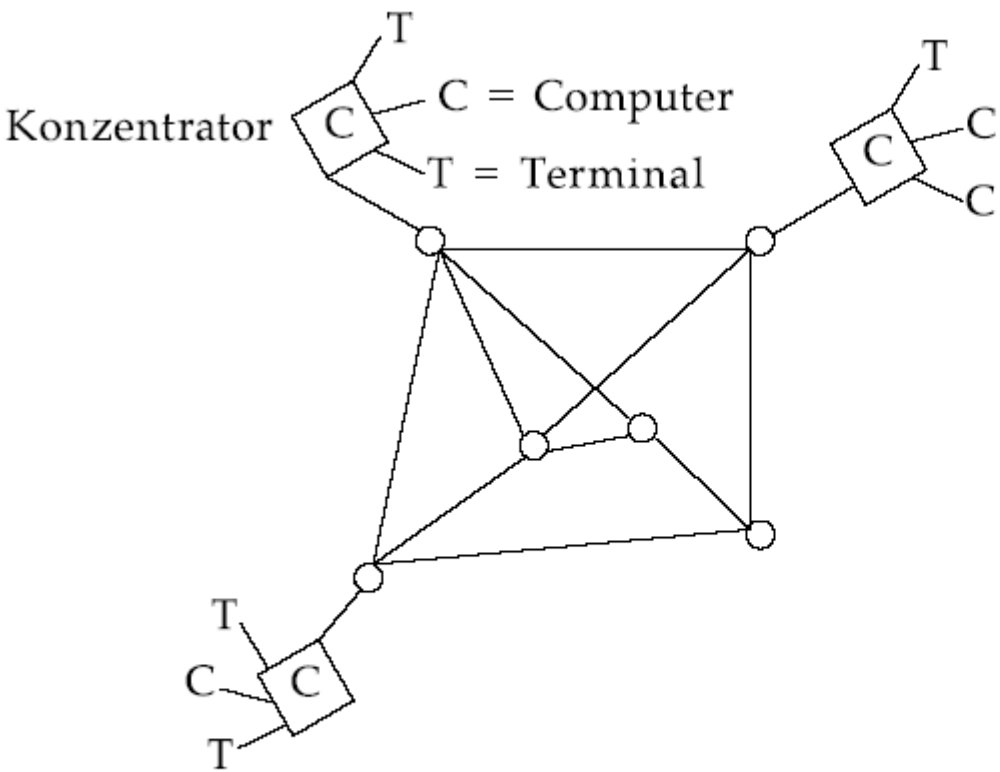


Abb. 7.32. Struktur eines Wide Area Networks (WAN)

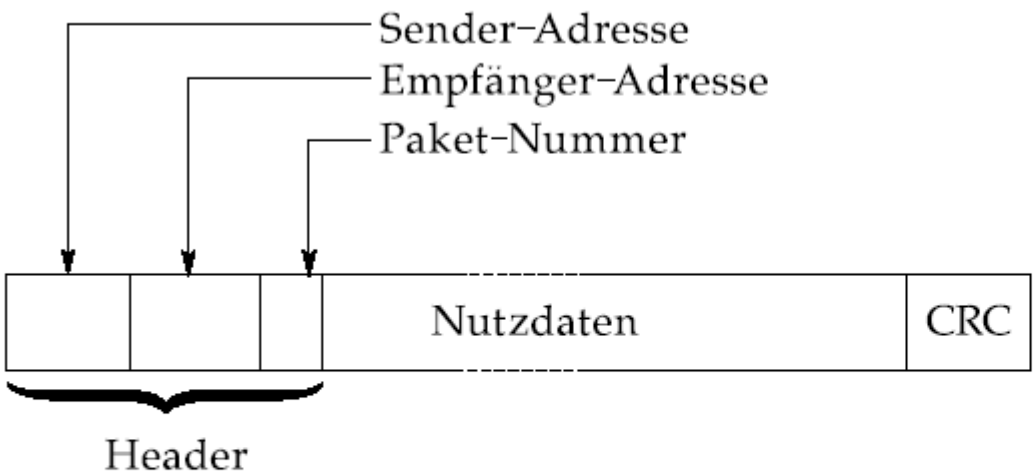


Abb. 7.33. Paketformat beim ARPANET

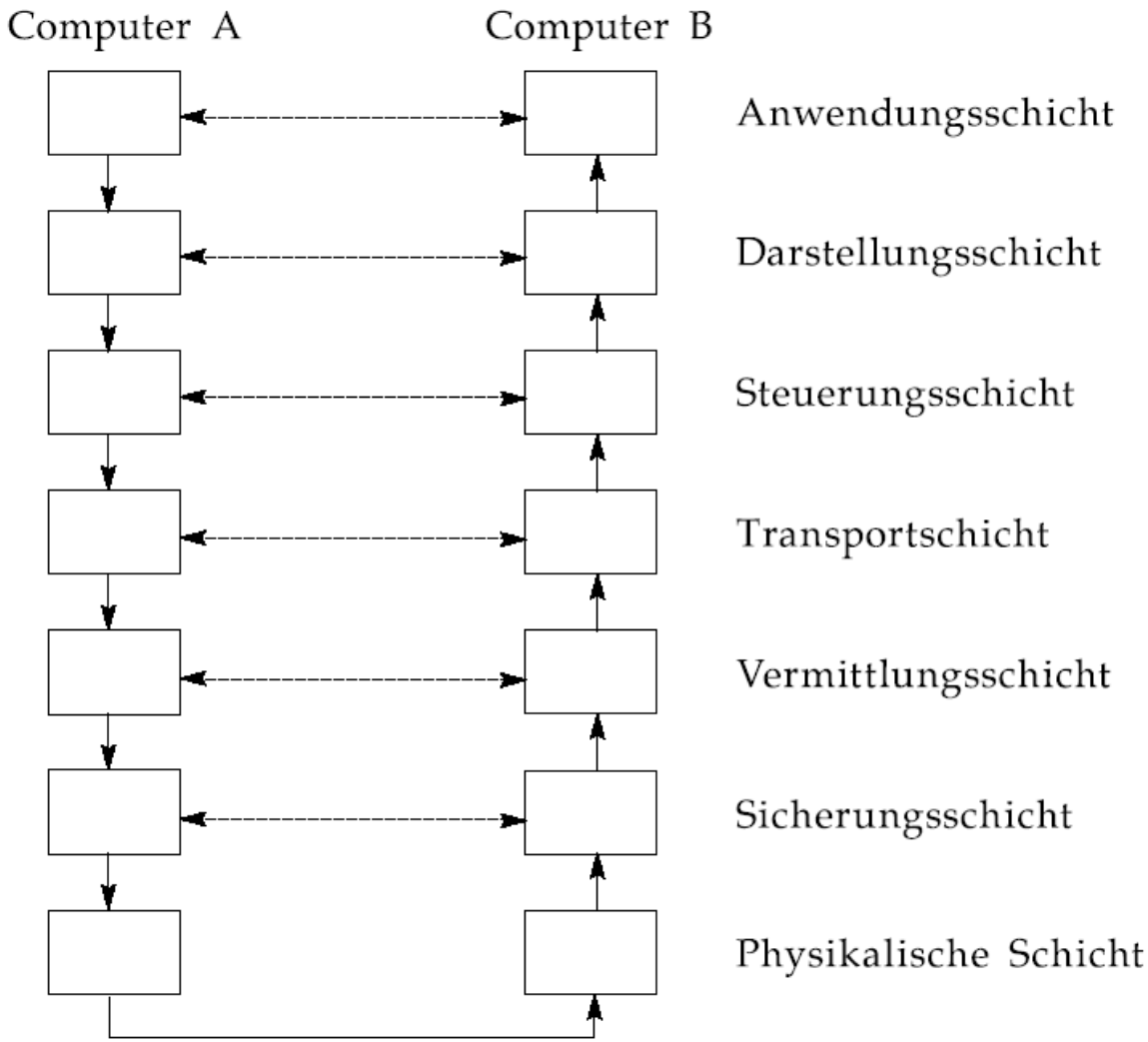


Abb. 7.34. Kommunikation zweier Computer im OSI–Modell

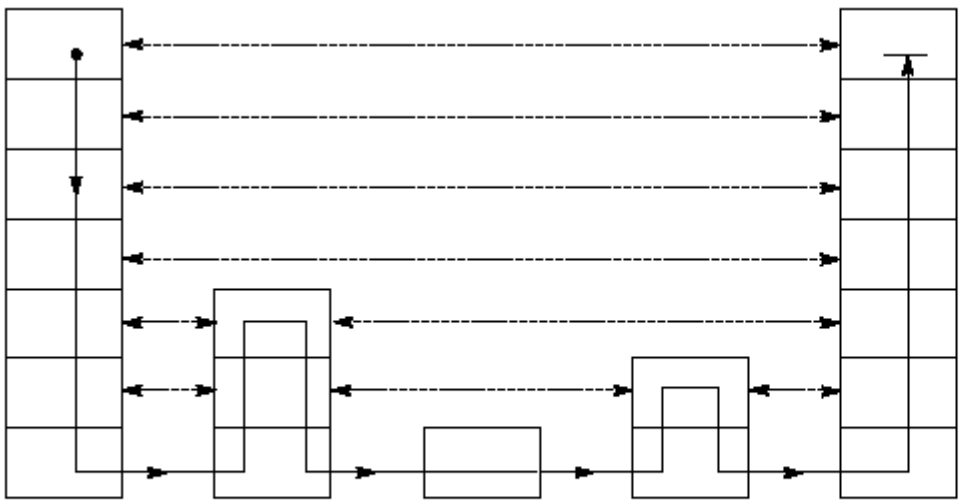


Abb. 7.35. Die Daten zwischen zwei Anwendungen können mehrere Netzwerkknoten (Relais–Stationen) durchlaufen.

8. Speicher

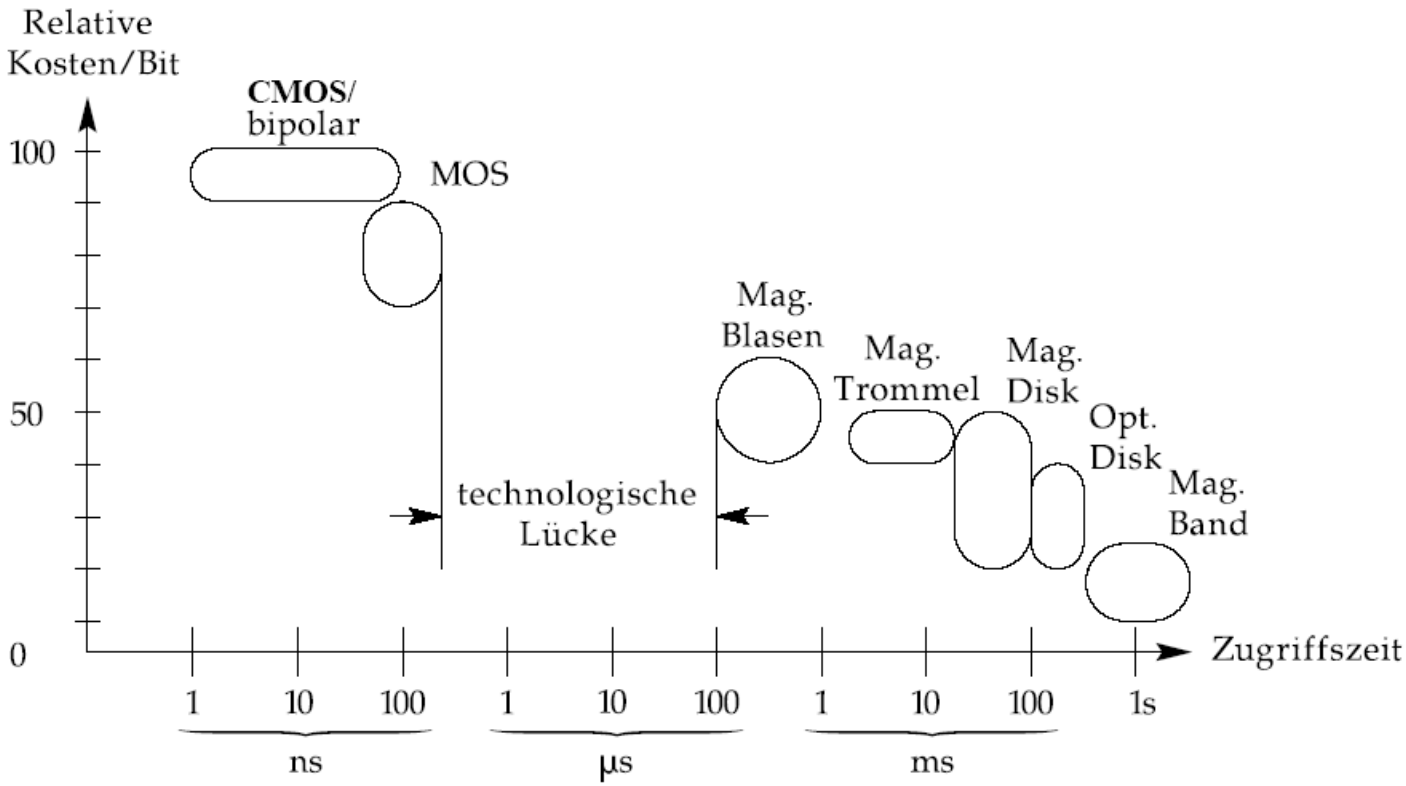


Abb. 8.1. Vergleich gebräuchlicher Speichertechnologien

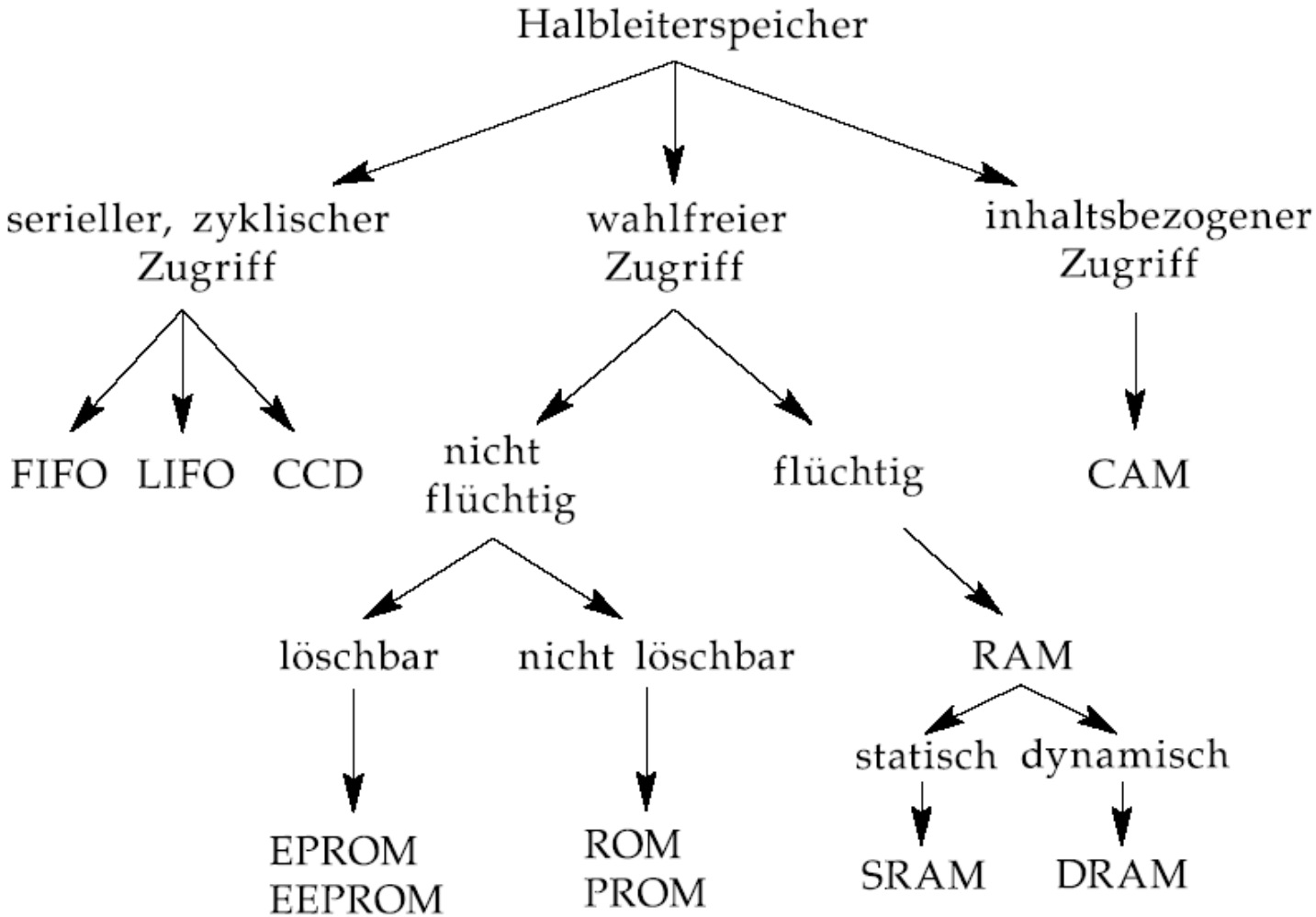


Abb. 8.2. Übersicht über Halbleiterspeicher

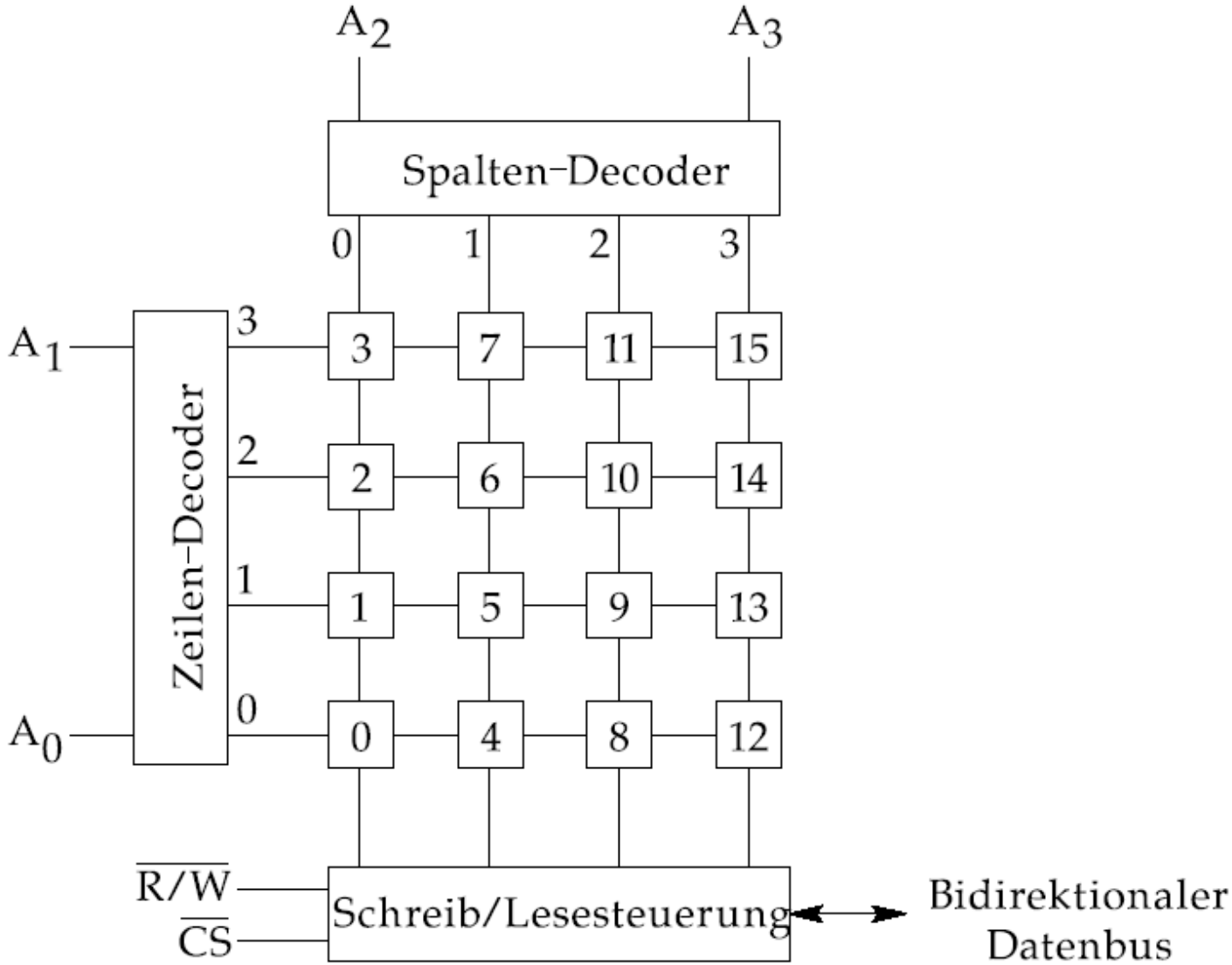


Abb. 8.3. Aufbau eines bitorientierten Halbleiterspeichers (16× 1)

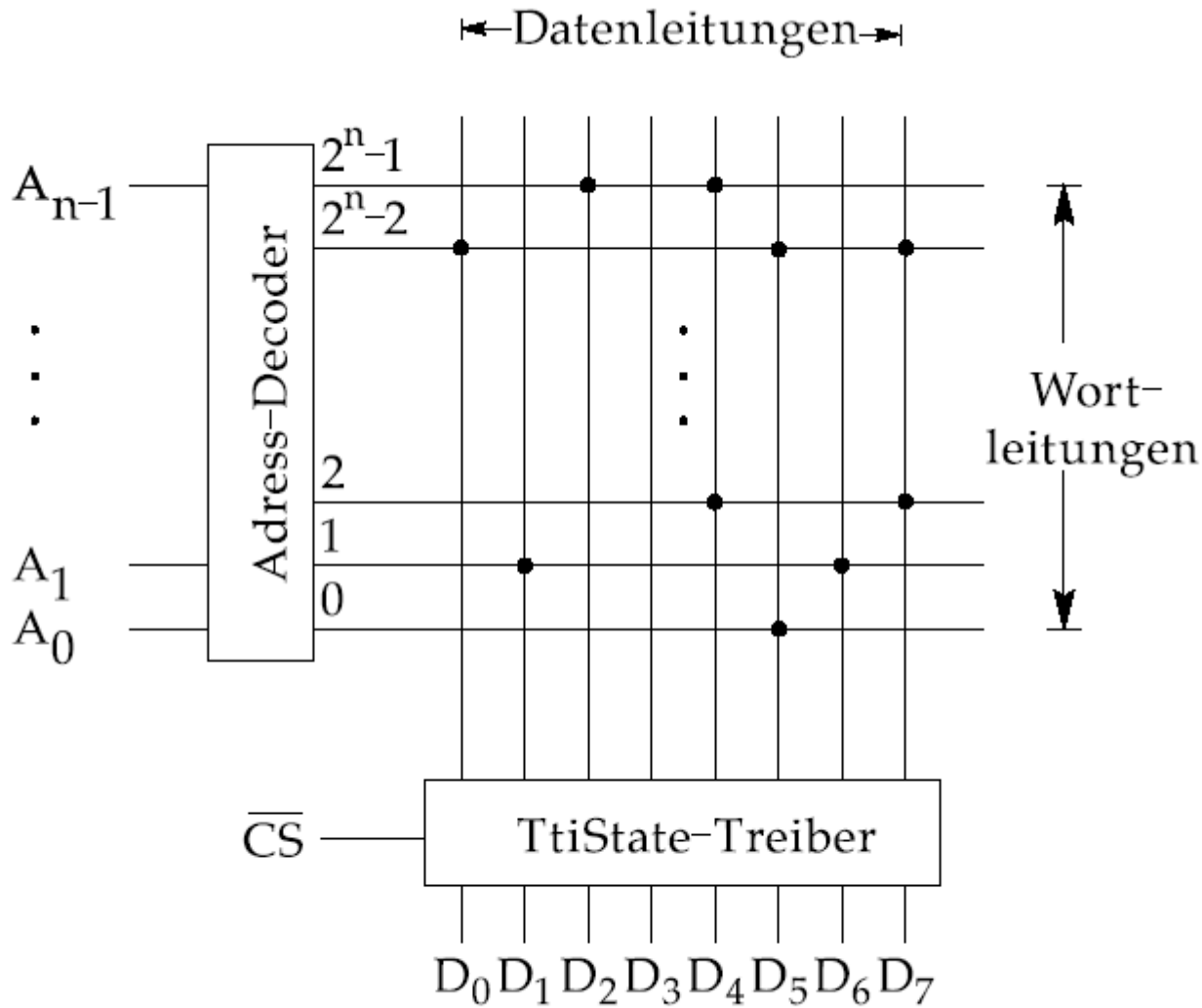


Abb. 8.4. Eindimensionale Adressierung bei hoher Wortbreite ($2^n \times 8$)

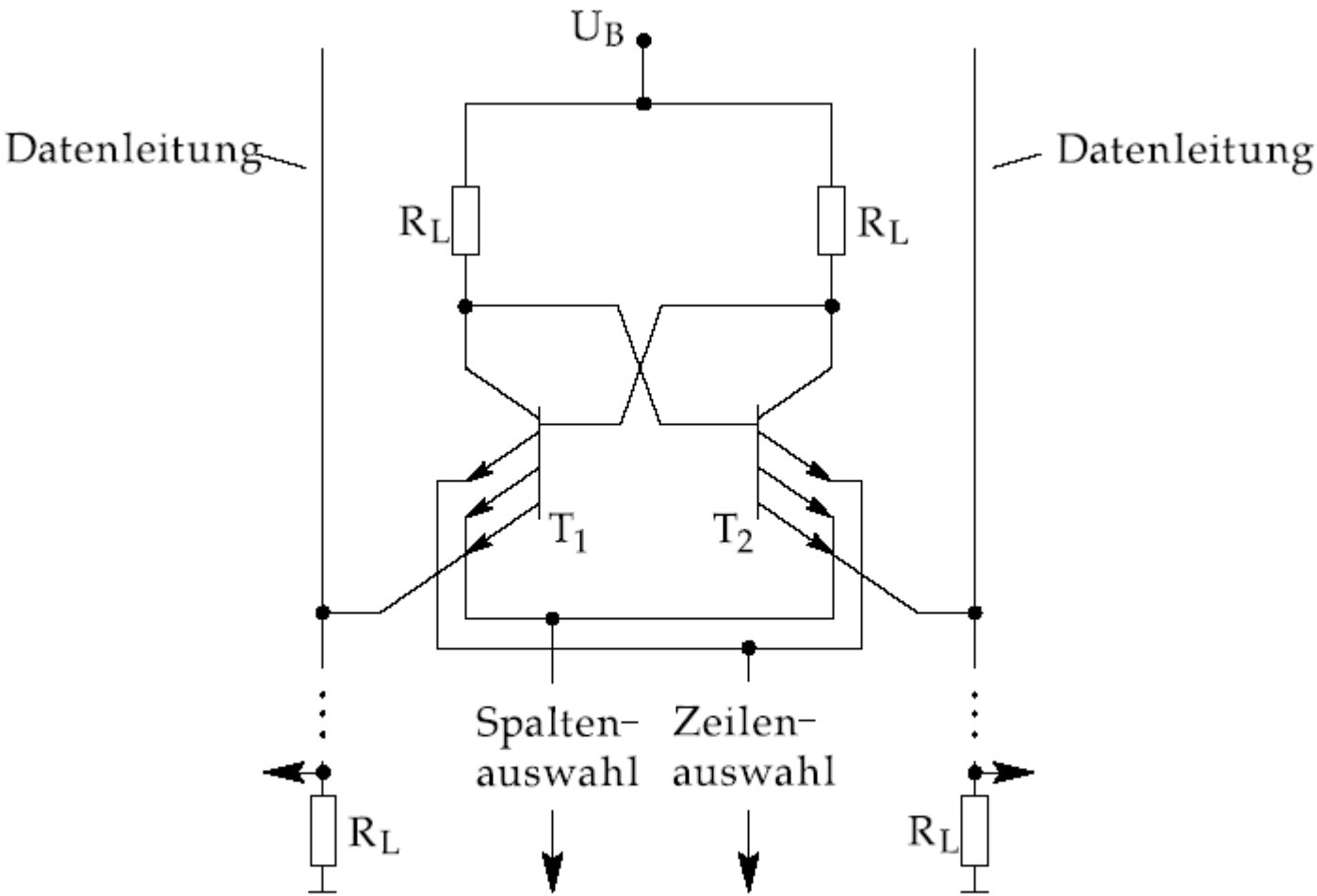


Abb. 8.5. Basis–Speicherzelle eines bipolaren RAMs

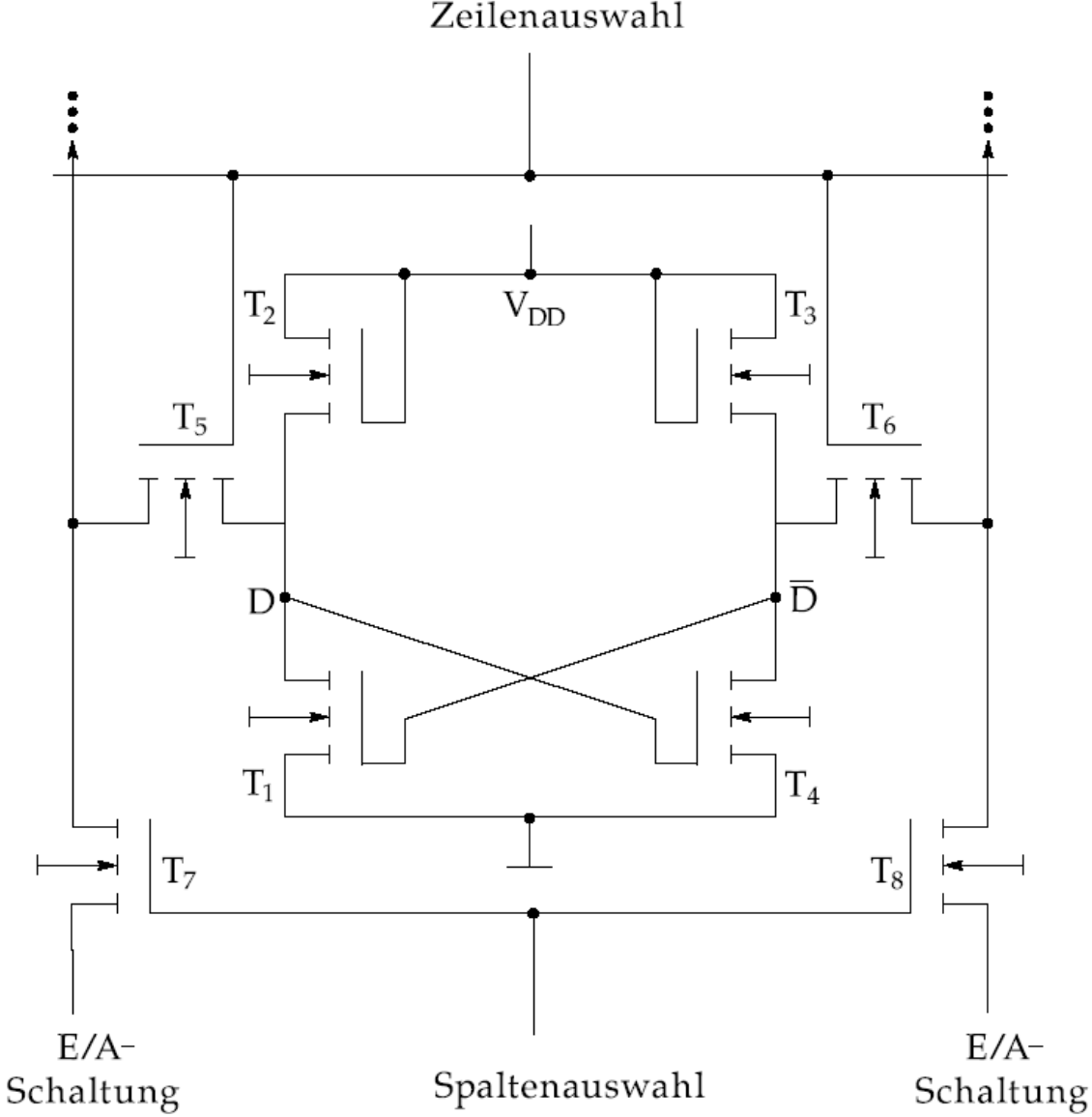


Abb. 8.6. Basis–Speicherzelle eines statischen NMOS–RAMs

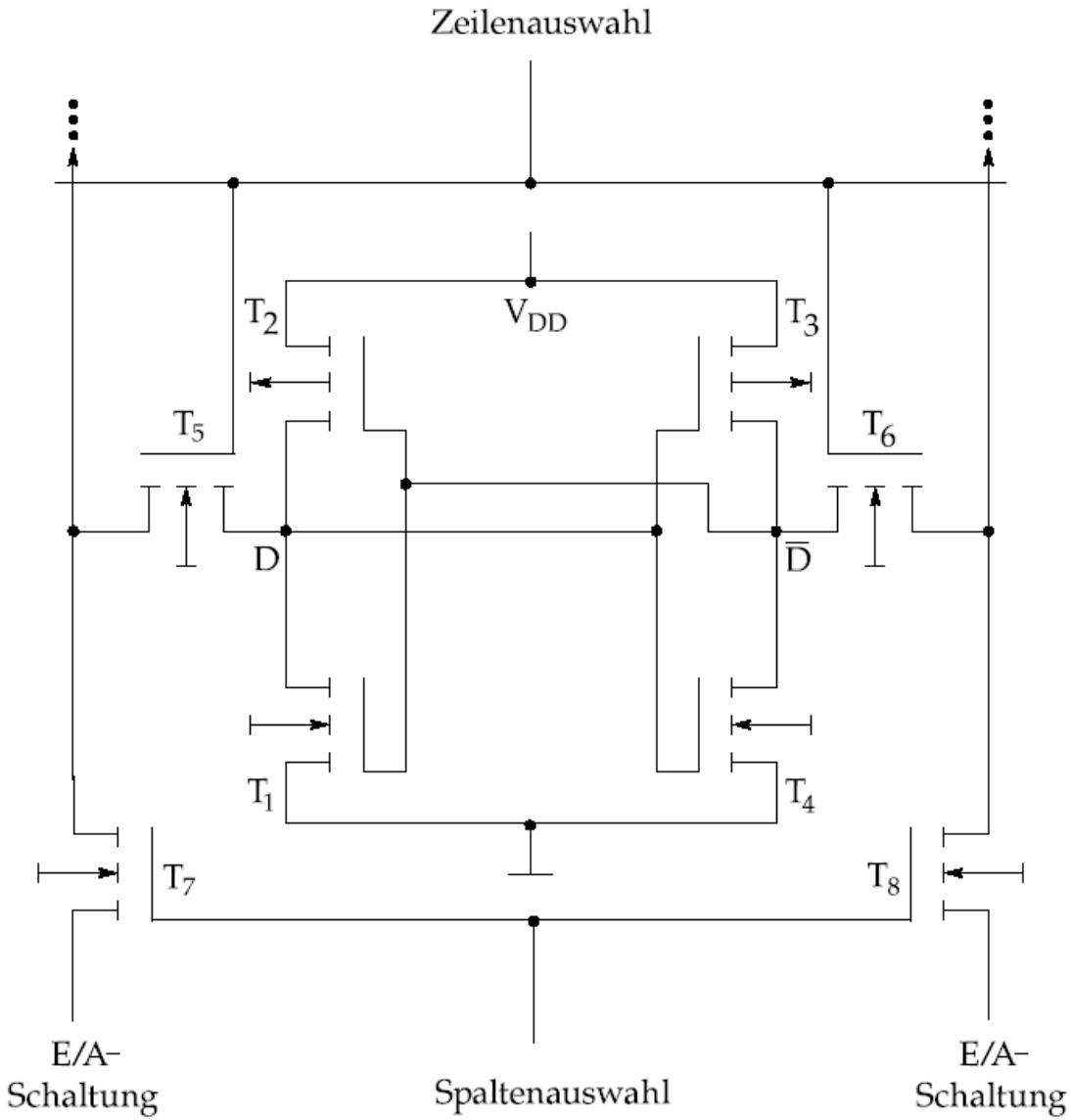


Abb. 8.7. Basis–Speicherzelle eines statischen CMOS–RAMs

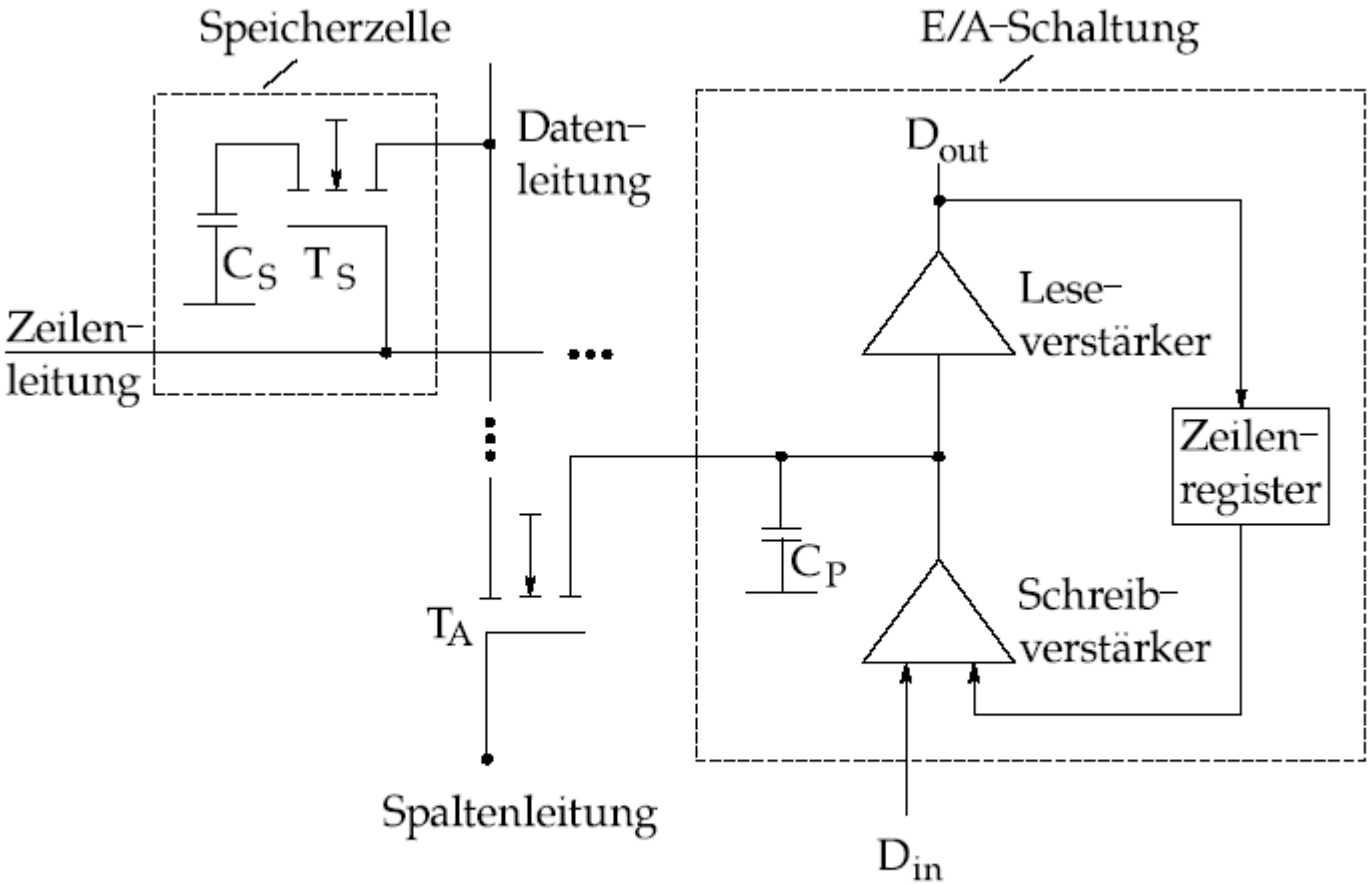


Abb. 8.8. Basis–Speicherzelle eines NMOS–DRAMs mit Ein–/Ausgabe Schaltung

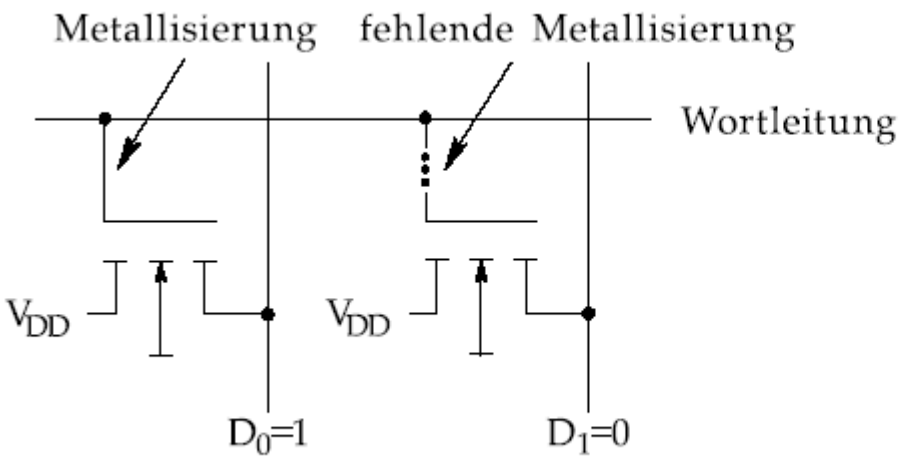


Abb. 8.9. Prinzip des maskenprogrammierbaren ROMs bei eindimensionaler Adressierung

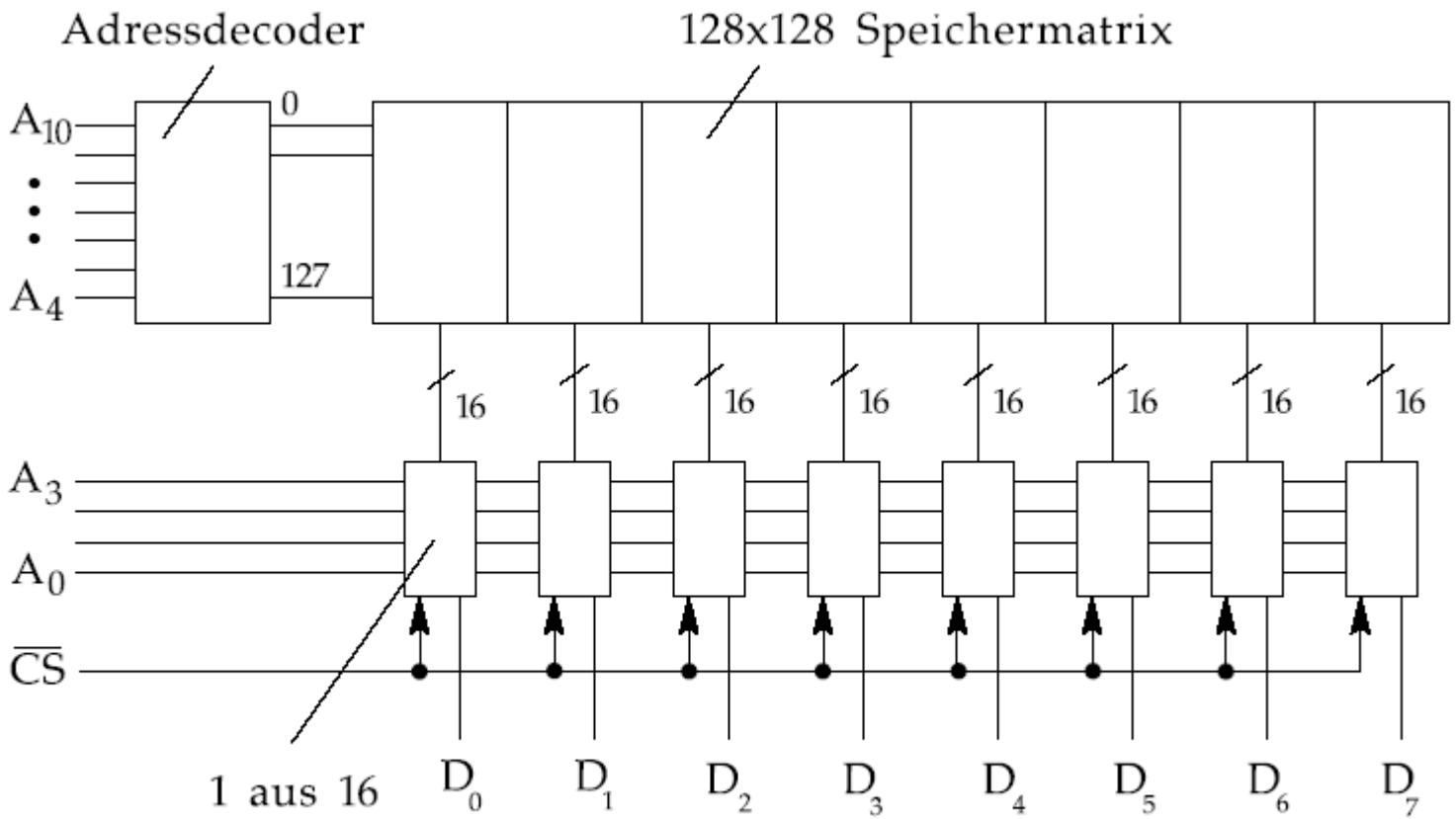


Abb. 8.10. Aufbau eines 2048×8 Bit organisierten Masken–ROMs (2716)

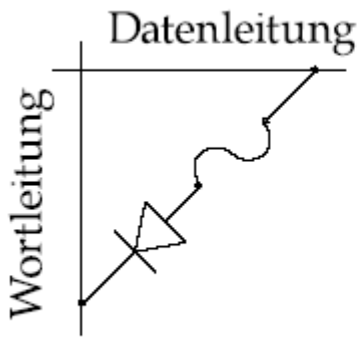


Abb. 8.11. Prinzip einer PROM–Speicherzelle

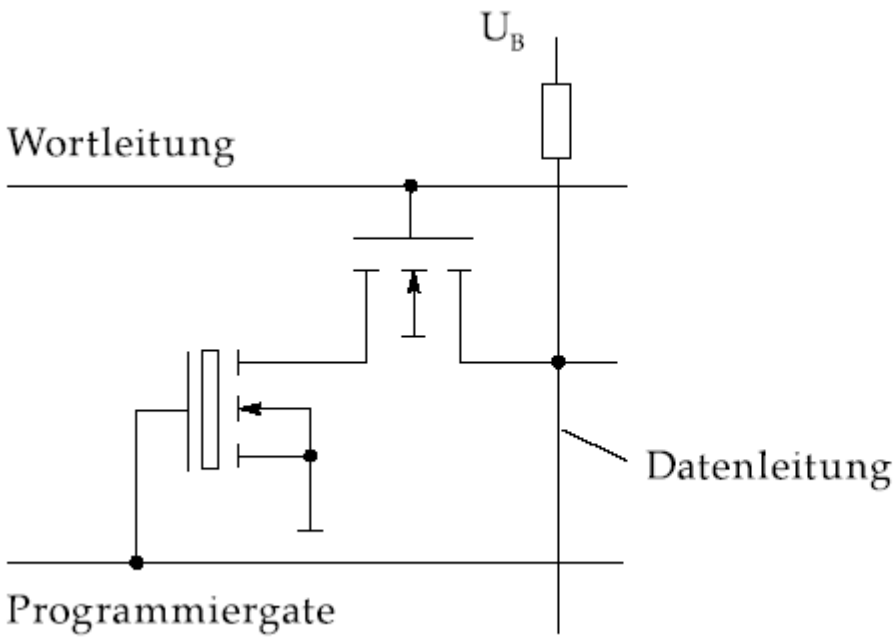


Abb. 8.12. Speicherzelle beim EEPROM 2816

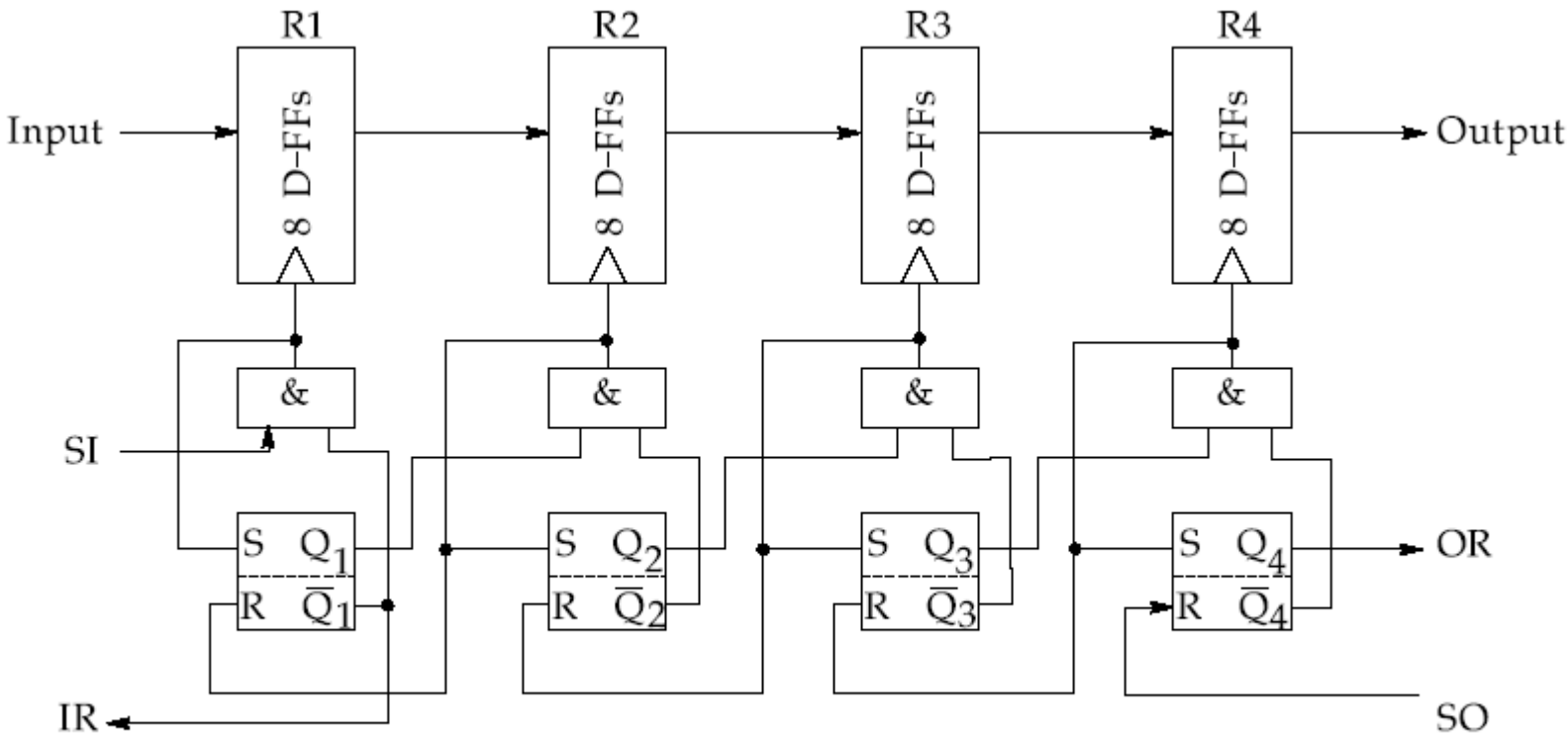


Abb. 8.13. FIFO–Speicher mit asynchroner Steuerung

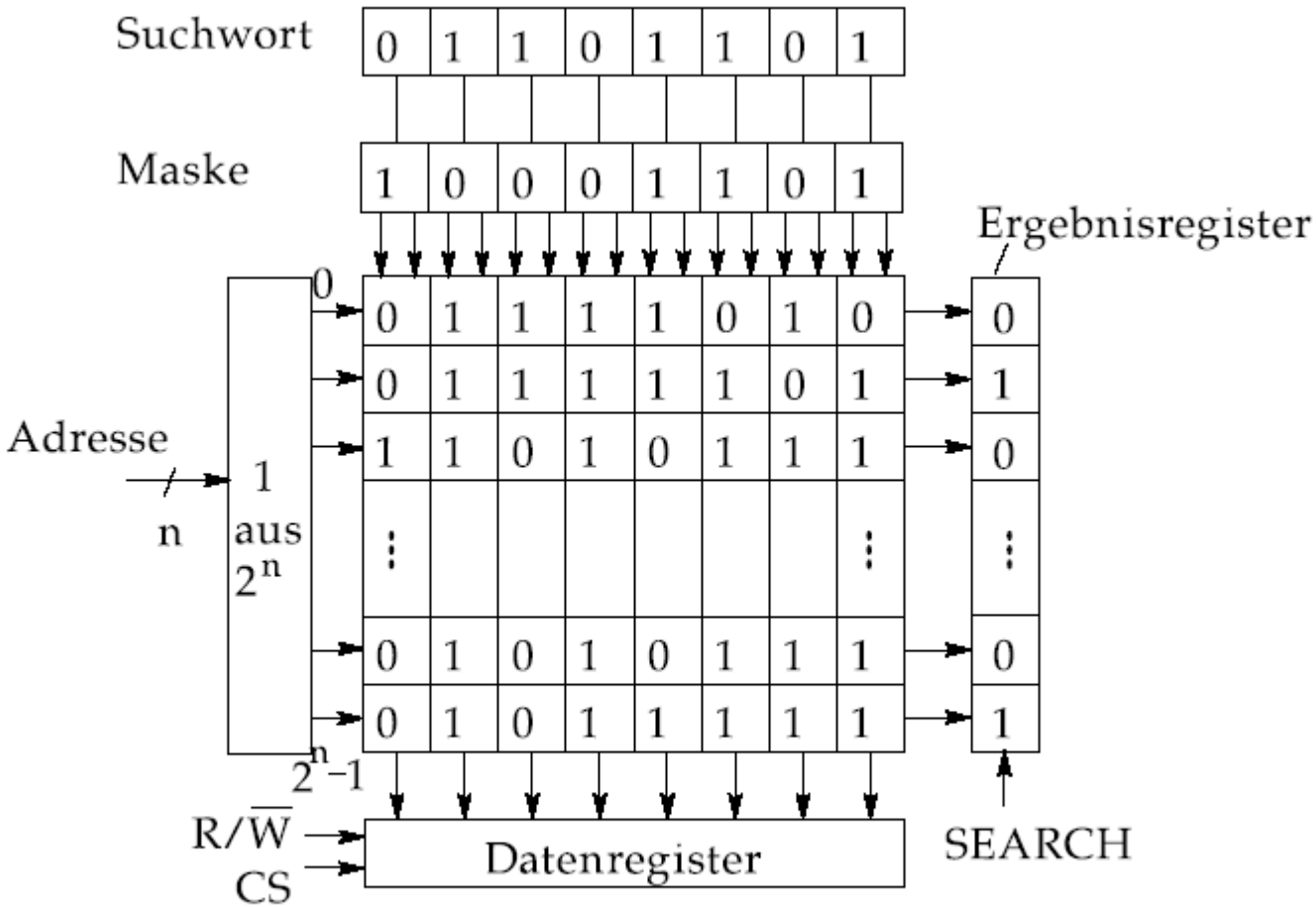


Abb. 8.14. Organisation und Arbeitsweise eines Assoziativspeichers

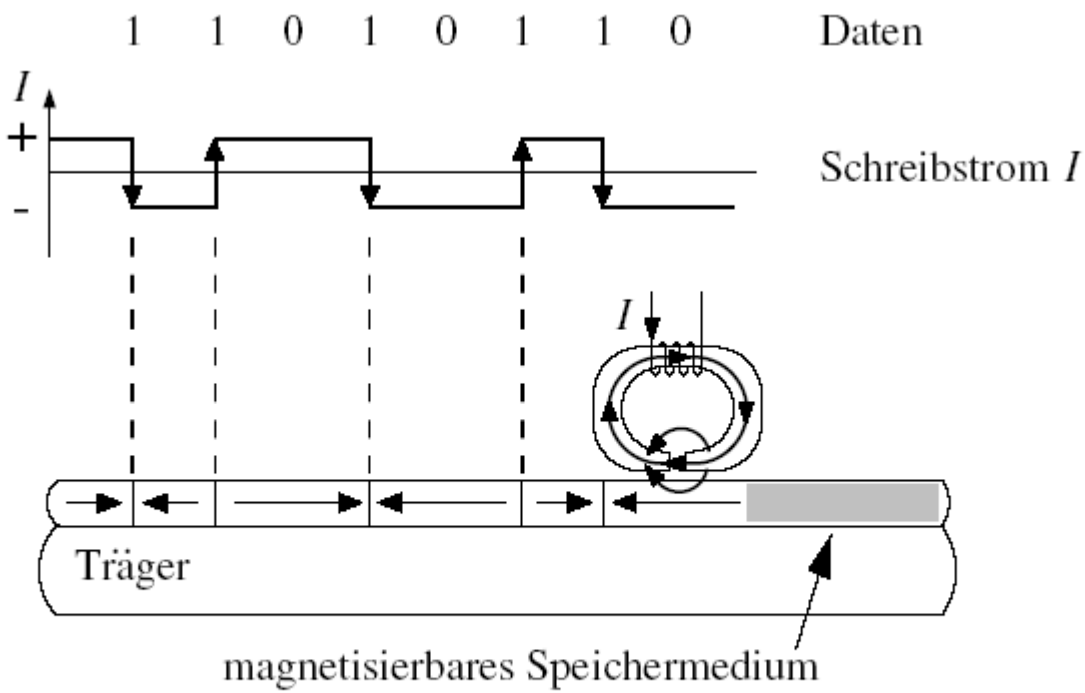


Abb. 8.15. Schreibvorgang bei einem magnetomotorischen Speichermedium.

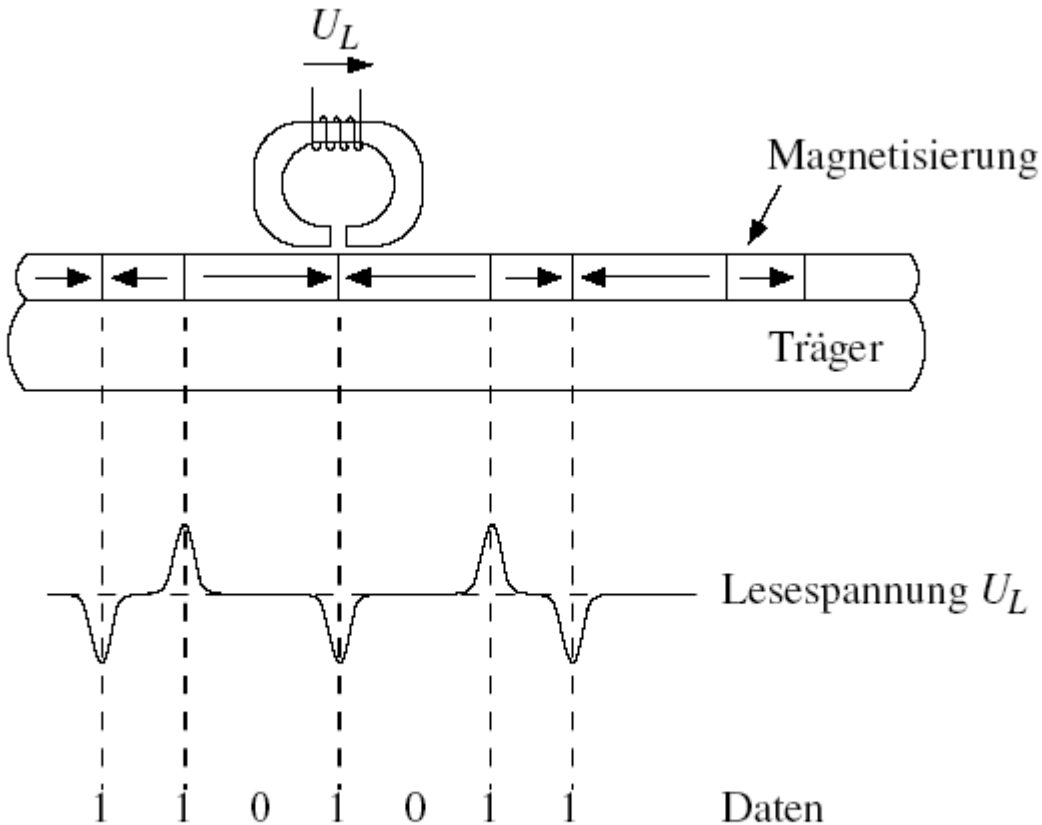


Abb. 8.16. Lesevorgang bei einem magnetomotorischen Speichermedium.

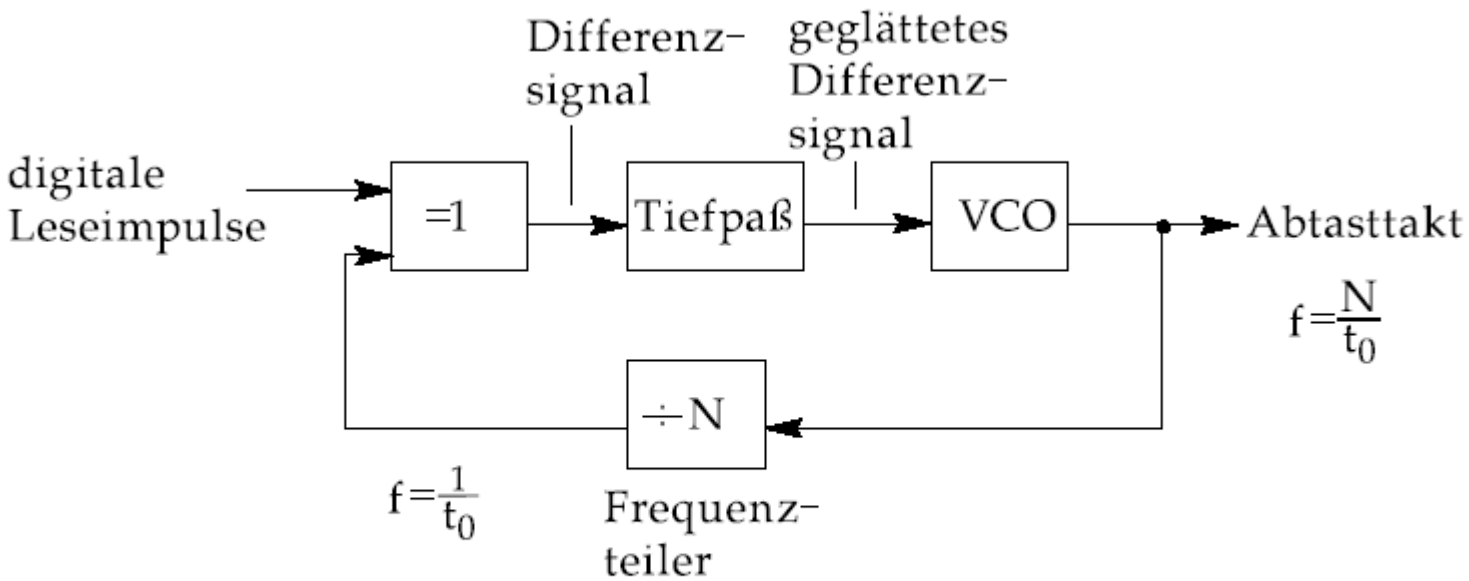


Abb. 8.17. Aufbau eines Phasenregelkreises (PLL) zur Gewinnung eines Abtasttaktes, der synchron zum Aufzeichnungstakt ist.

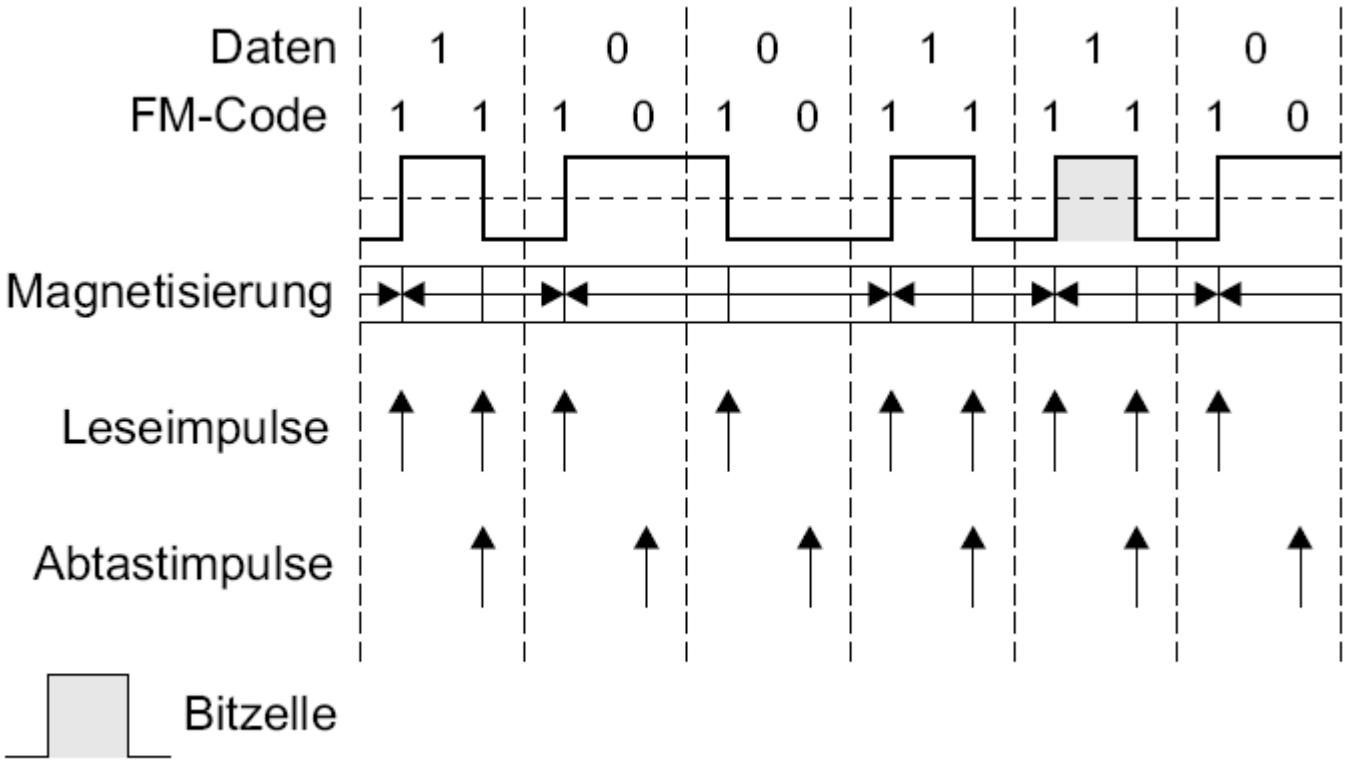


Abb. 8.18. Lesevorgang bei der FM–Codierung.

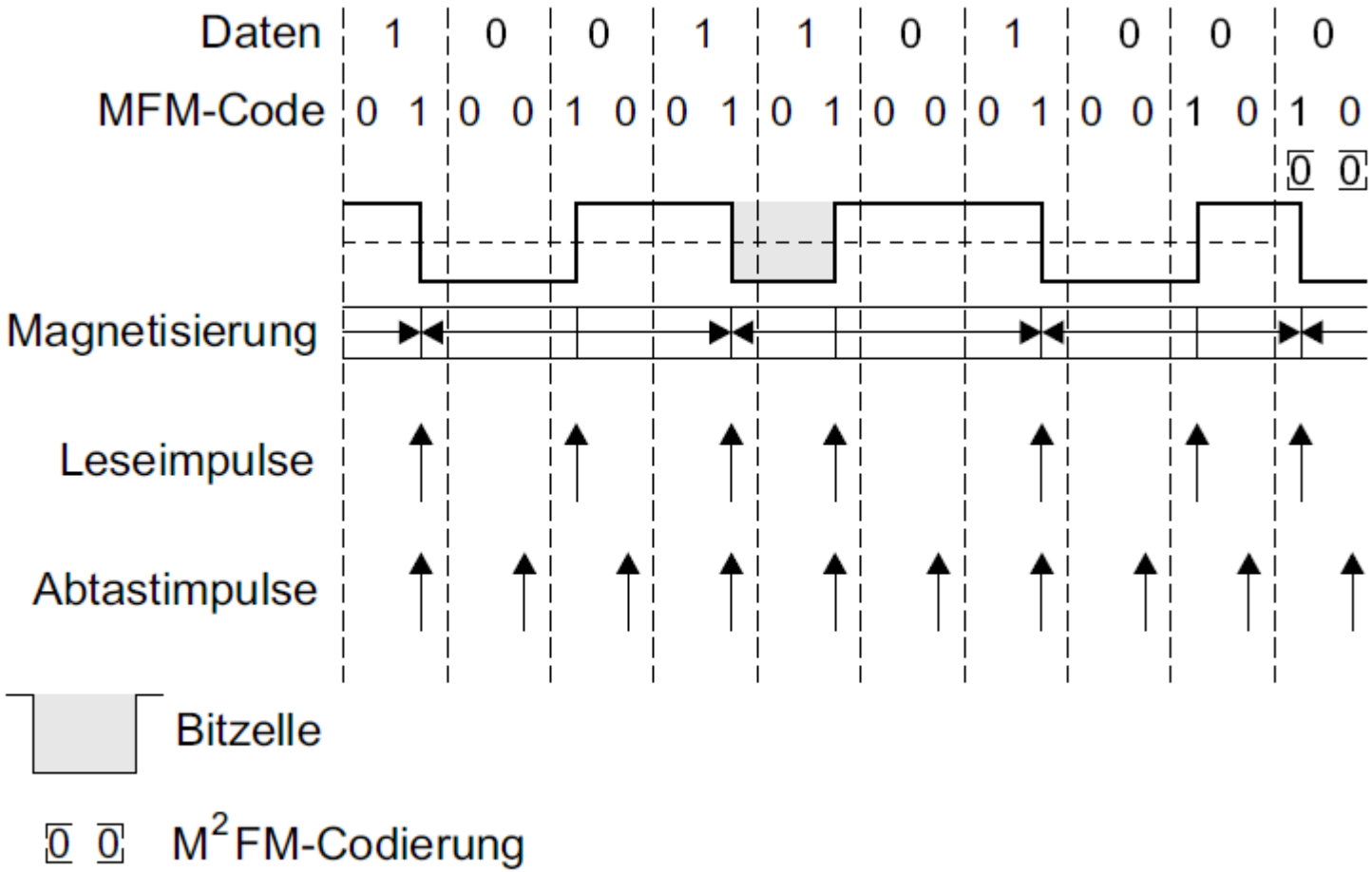


Abb. 8.19. Lesevorgang bei der MFM–Codierung.

Datenbits	Speichercode
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
0011	00001000

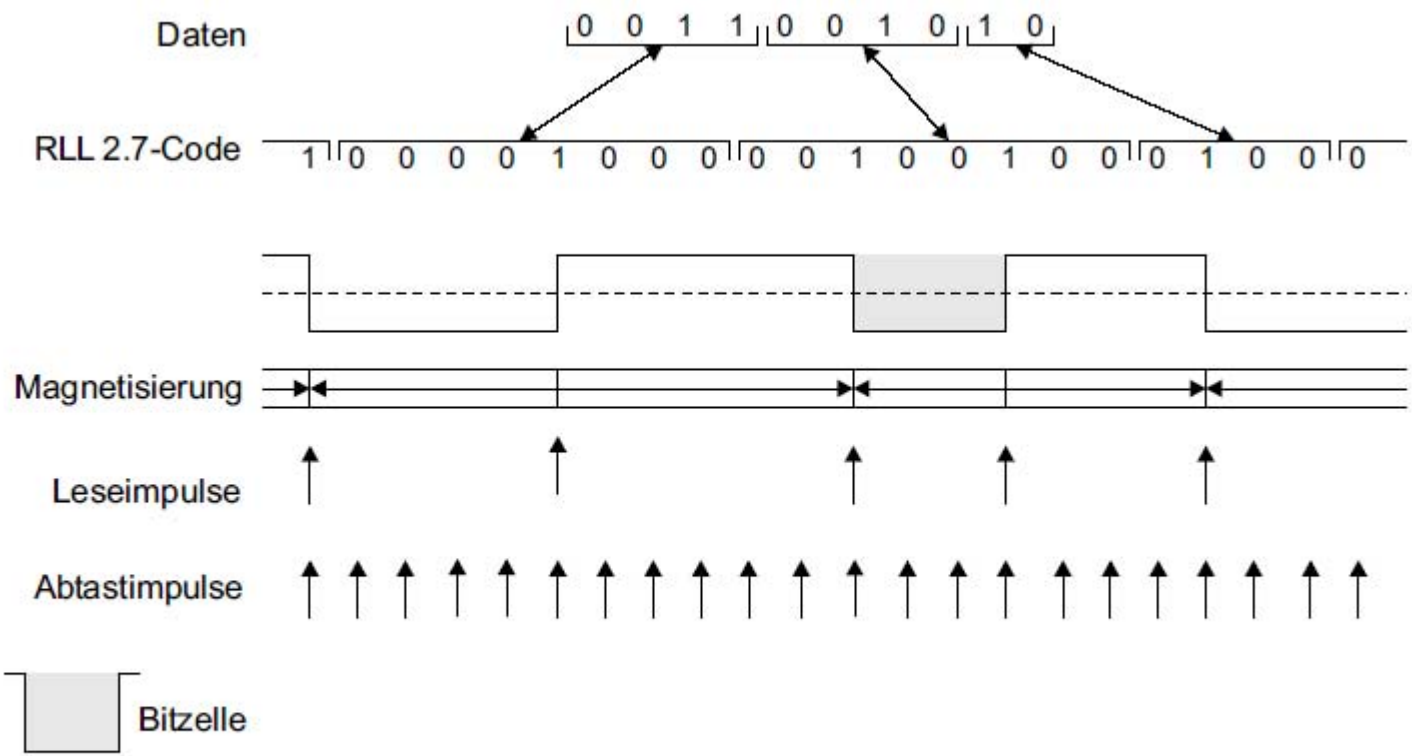


Abb. 8.20. Lesevorgang mit RLL 2.7–Codierung bei einer Festplatte.

	Flusswechsel/Datenbit
FM (SD)	1,5
MFM (DD)	0,75
RLL 2.7	0,43

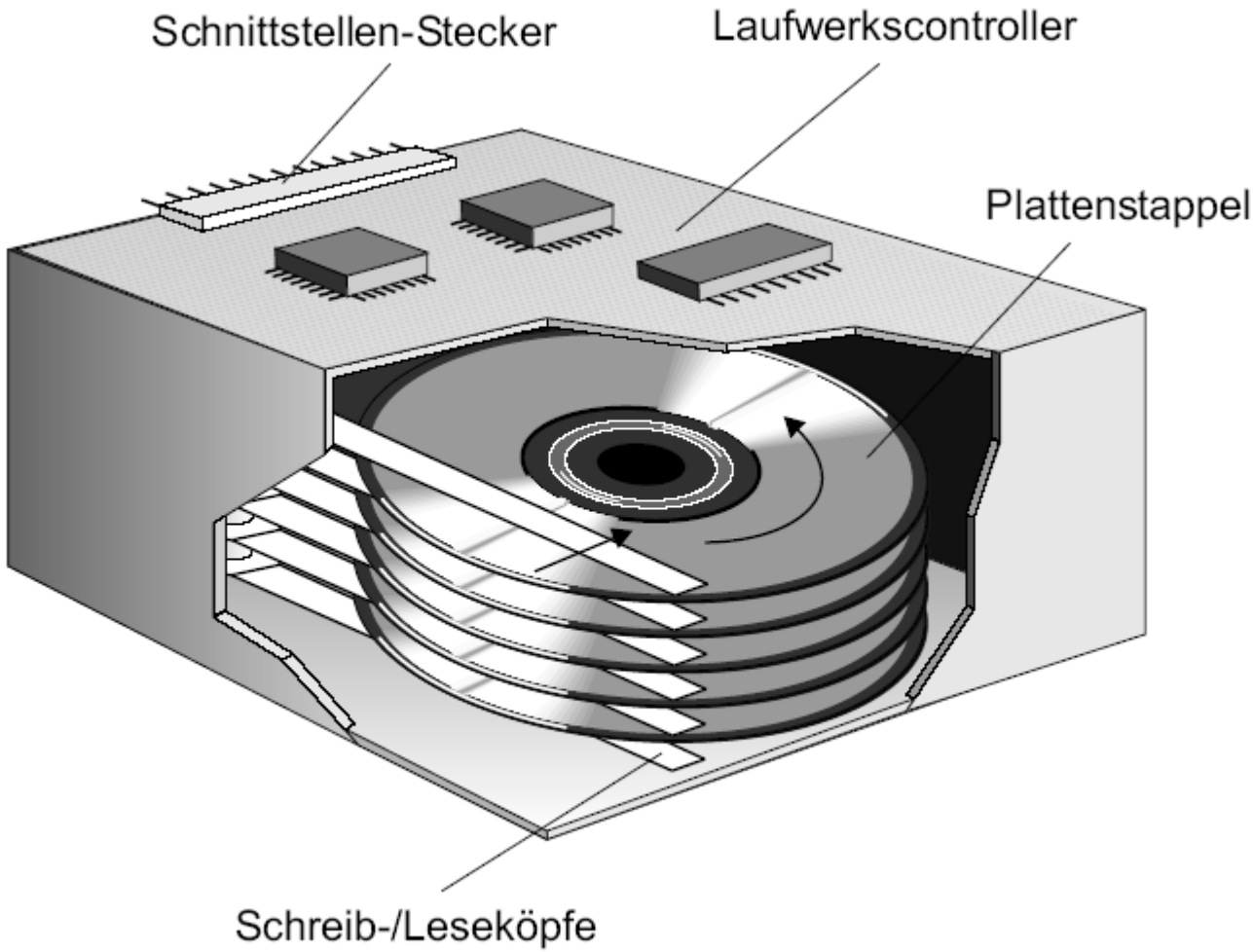


Abb. 8.21. Aufbau eines Festplattenlaufwerks.

Tabelle 8.1. Vergleich von Größeneinheiten im Dual– und Dezimalsystem.

Zeichen	Name	Wert Dual	Wert Dezimal
K	Kilo	2^{10}=1.024	1.000
M	Mega	2^{20}=1.048.576	1.000.000
G	Giga	2^{30}=1.073.741.824	1.000.000.000
T	Tera	2^{40}=1.099.511.627.766	1.000.000.000.000
P	Peta	2^{50}=1.125.899.906.842.624	1.000.000.000.000.000

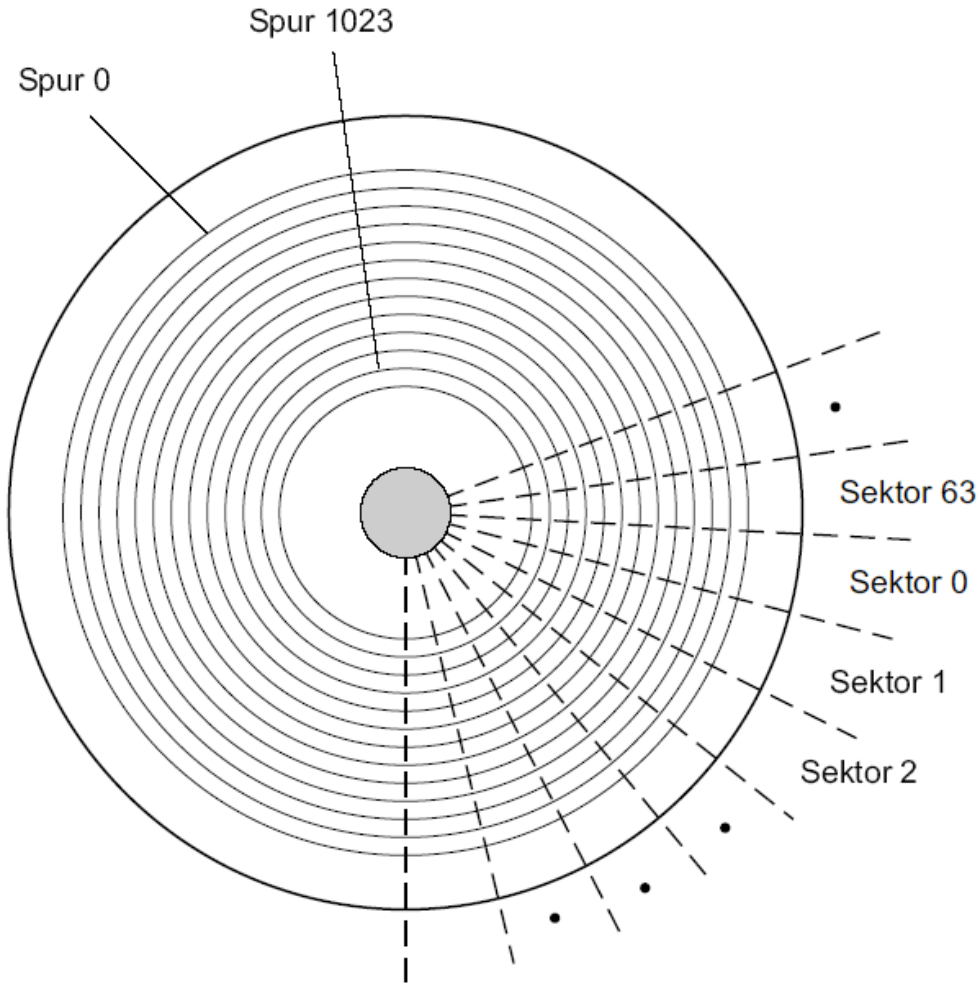


Abb. 8.22. Aufteilung der Plattenoberfläche.

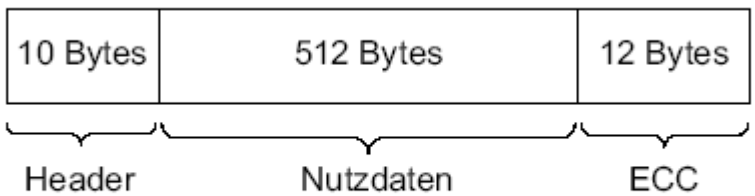


Abb. 8.23. Format eines Sektors bei Softsektorierung.

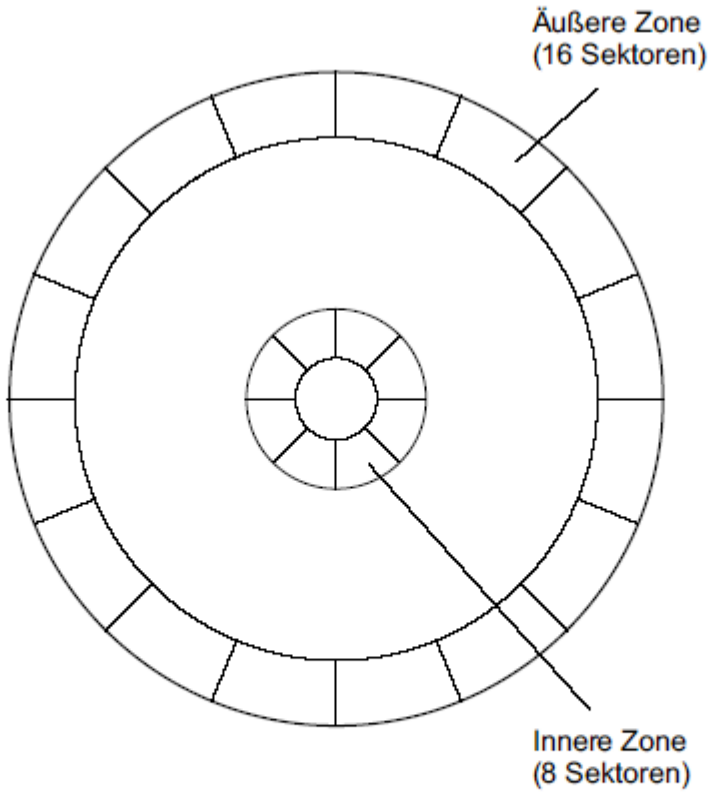


Abb. 8.24. Prinzip der Zonenaufzeichnung: Bei gleichbleibender Sektorlänge können auf den äußeren Spuren mehr Sektoren untergebracht werden.

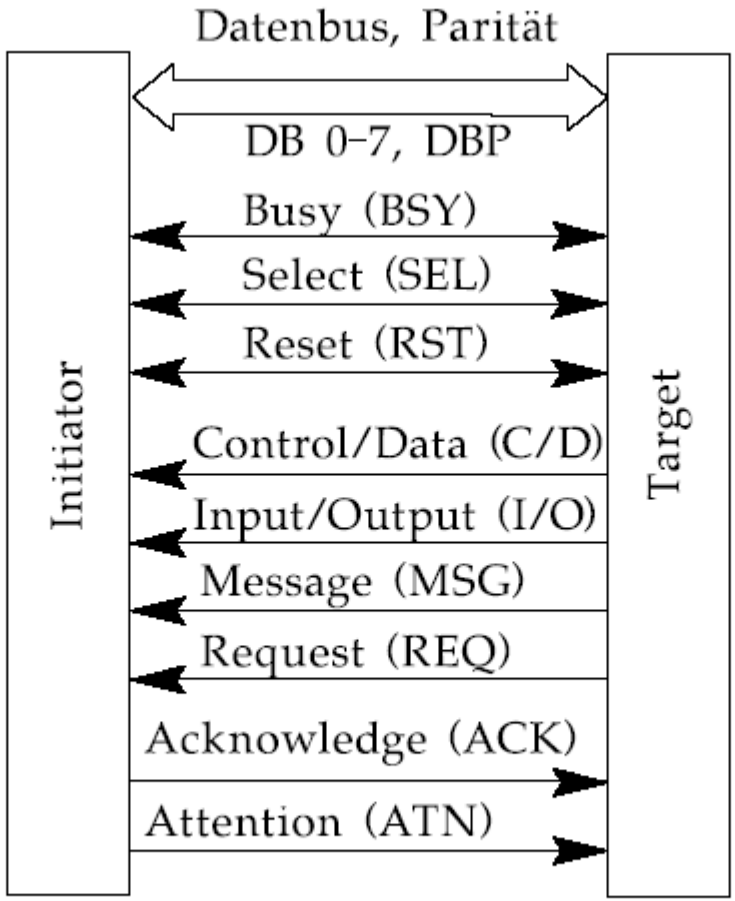


Abb. 8.25. Verbindung zwischen Initiator und Target beim SCSI–Bus.

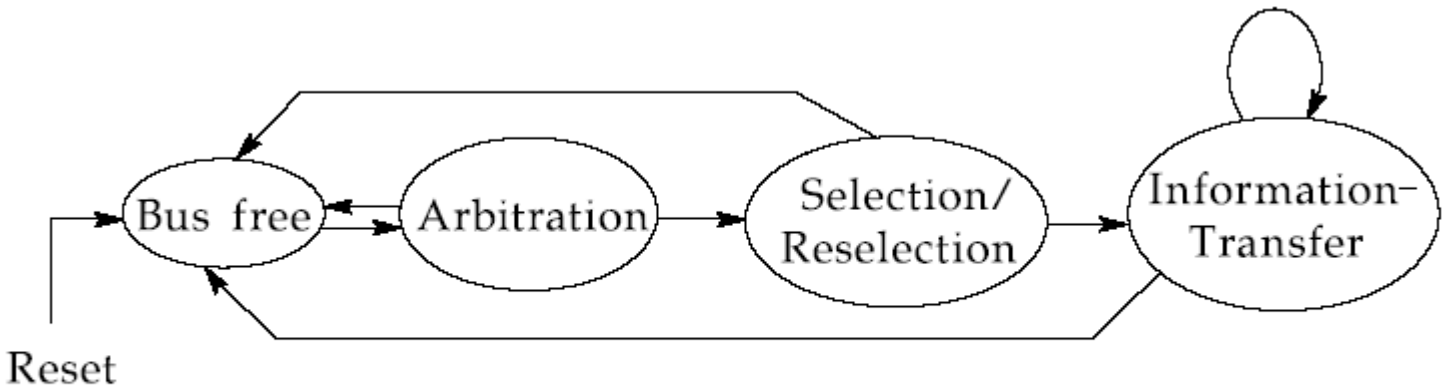


Abb. 8.26. Mögliche Zustände des SCSI–Busses.

Die folgende Tabelle stellt beispielhaft die Aufteilung einer Festplatte in zwei primäre Partitionen dar:

LBA	Inhalt
0	MBR und Partitionstabelle der Festplatte
1	Bootsektor der ersten primären Partition
2	Erster Datensektor der ersten primären Partition
⋮	⋮
L_1	Letzter Datensektor der ersten primären Partition
L_1+1	Bootsektor der zweiten primären Partition
L_1+2	Erster Datensektor der zweiten primären Partition
⋮	⋮
L_2	Letzter Datensektor der zweiten primären Partition

Tabelle 8.2. Bekannte Dateisysteme für verschiedene Betriebssysteme.

DOS	Windows 95/98	OS/2	NT/Windows 2000/XP	LINUX
FAT	VFAT, FAT32	HPFS	NTFS, FAT32	Ext2fs, Reiserfs, Swapfs

Tabelle 8.3. Bedeutung der verschiedenen FAT–Einträge.

FAT–12	FAT–16	FAT–32	Bedeutung
000	0000	0000.0000	freier Cluster
XXX	XXXX	0XXX.XXXX	nächster Cluster
obige Zeile gilt nur, sofern nicht eine der nachfolgenden Belegungen			
FF0–FF6	FFF0–FFF6	FFFF.FFF0–FFFF.FFF6	reservierte Werte
FF7	FFF7	FFFF.FFF7	Cluster defekt
FF8–FFF	FFF8–FFFF	FFFF.FFF8–FFFF.FFFF	Letzter Cluster der Datei

Tabelle 8.4. Aufbau eines Verzeichniseintrags.

Offset	Bedeutung	Größe
00_H	Dateiname	8 Byte, ASCII
08_H	Erweiterung	3 Byte, ASCII
$0B_H$	Attribut	1 Byte
$0C_H$	reserviert	10 Byte
16_H	Uhrzeit der letzten Änderung	2 Byte
18_H	Datum der letzten Änderung	2 Byte
$1A_H$	Startcluster	2 Byte
$1C_H$	Dateilänge	4 Byte

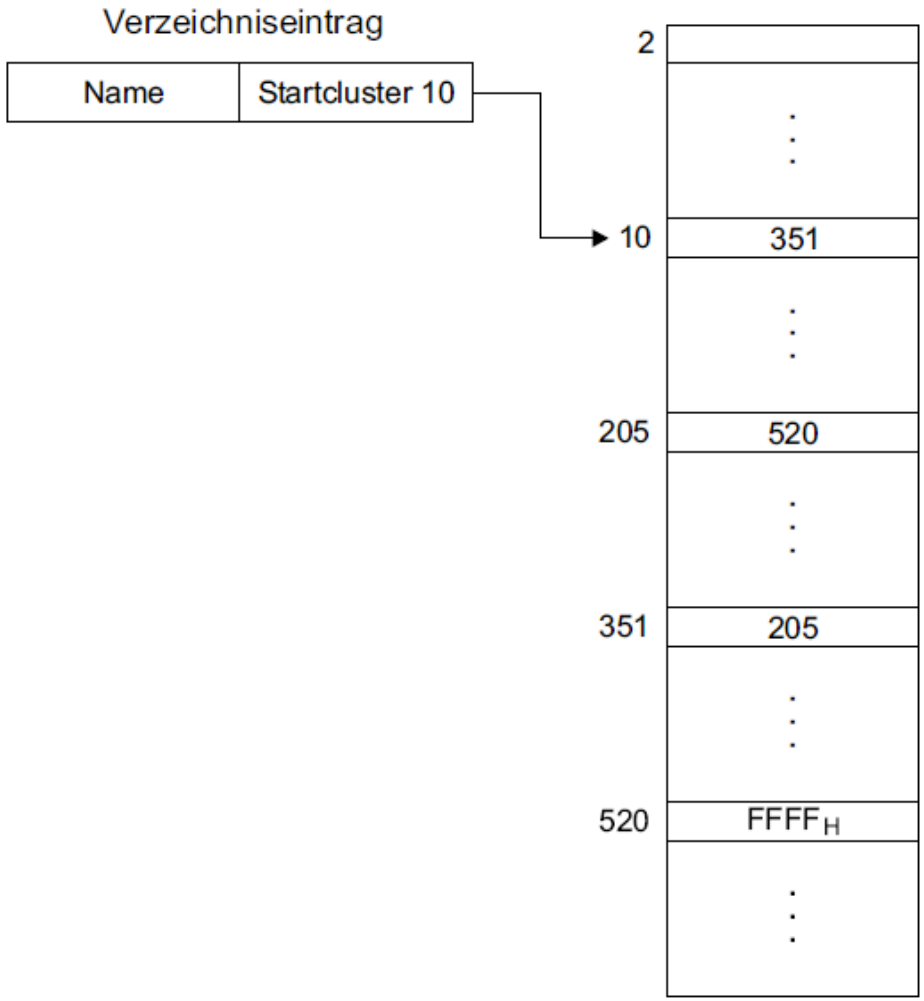


Abb. 8.27. Beispiel für das Lesen einer Datei mittels FAT.

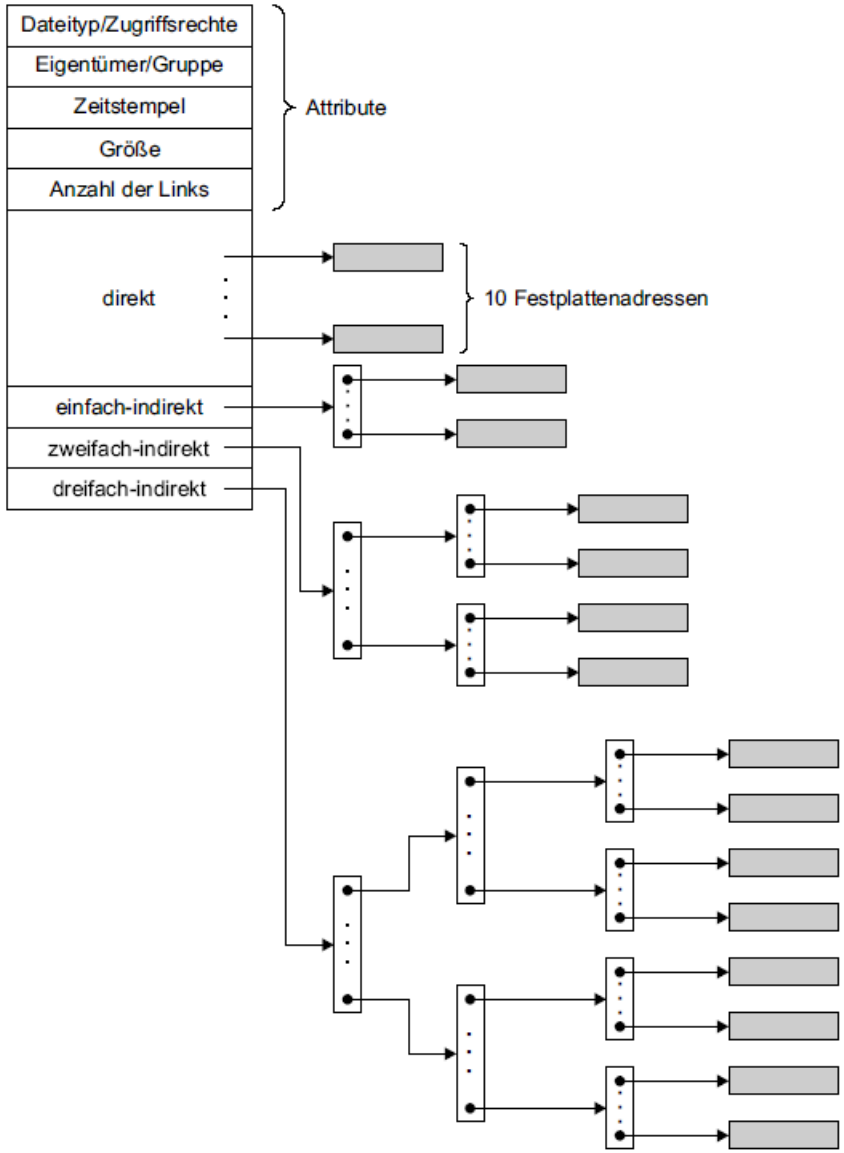


Abb. 8.28. Aufbau eines Inode.

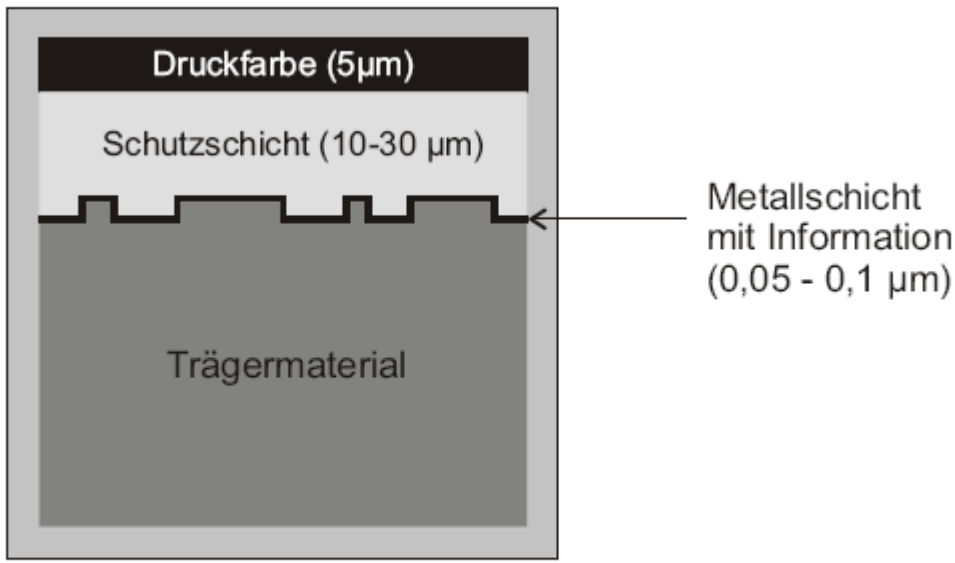


Abb. 8.29. Schichtenfolge bei einer CD–ROM.

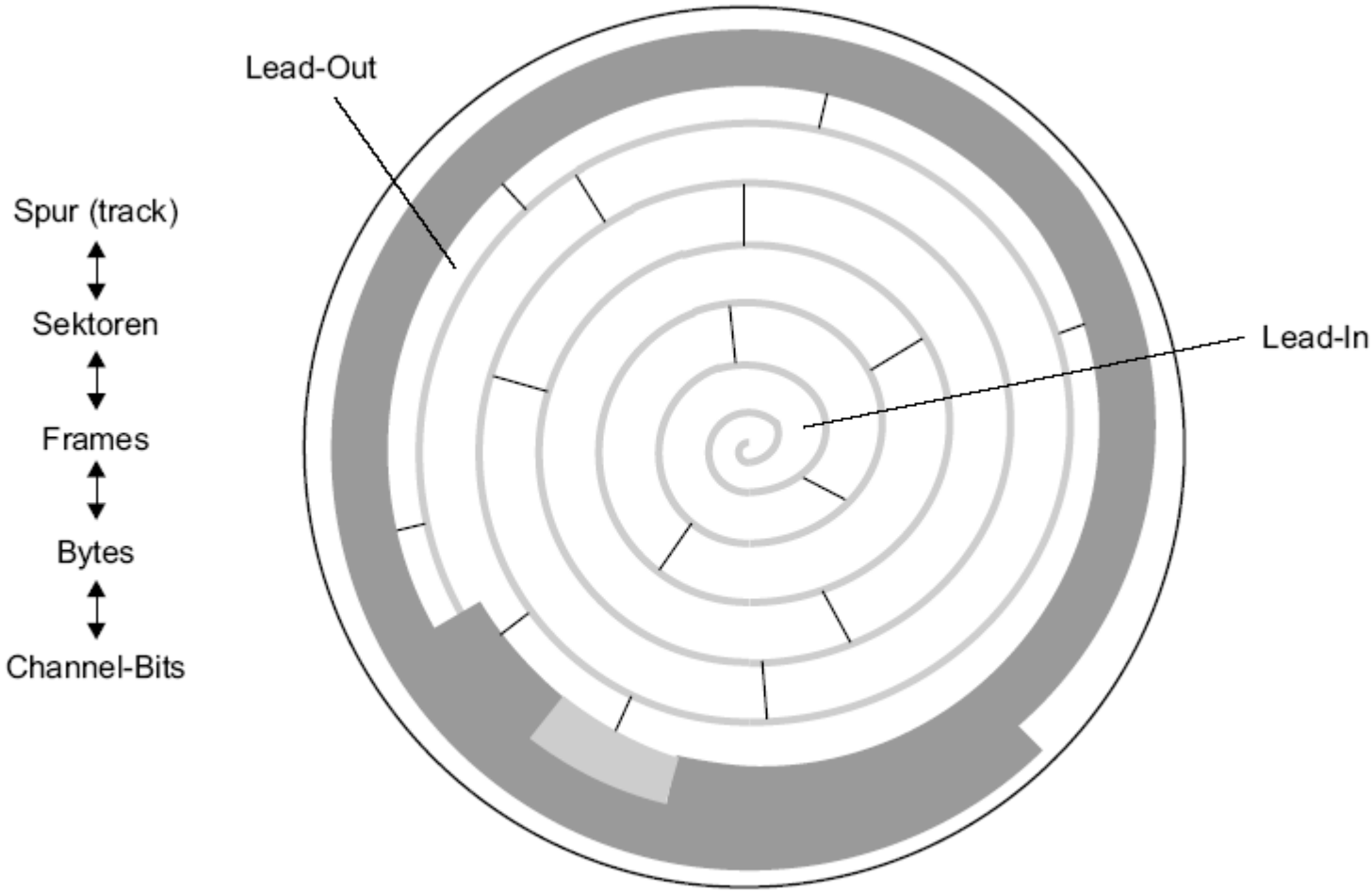


Abb. 8.30. Datenorganisation bei einer CD–ROM.

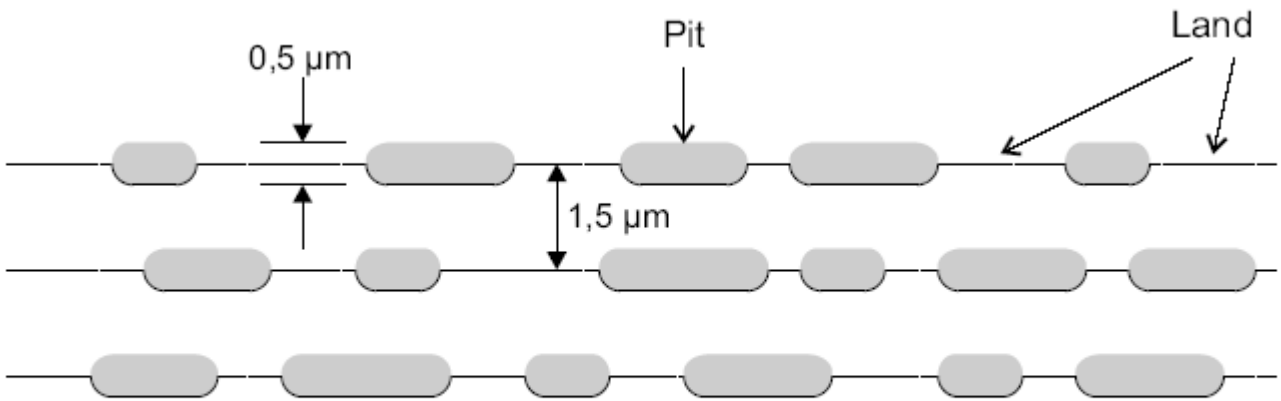


Abb. 8.31. Abmessung der Speicherelemente auf einer CD–ROM.

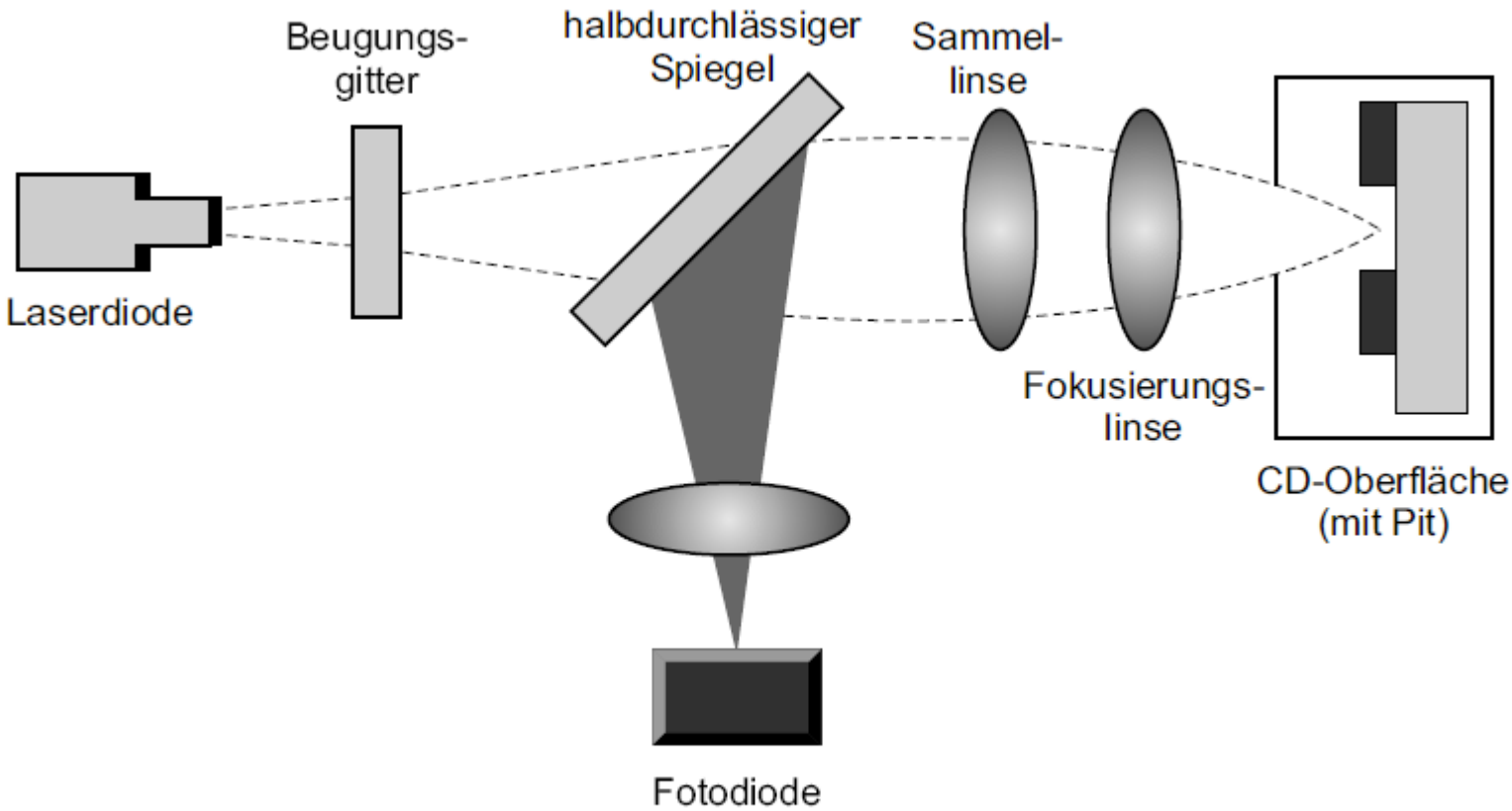


Abb. 8.32. Schematischer Aufbau der Leseoptik.

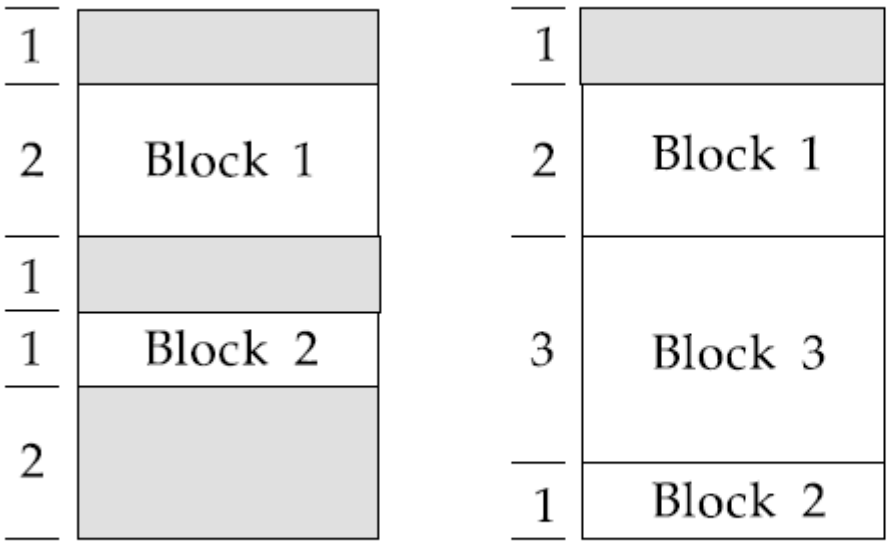


Abb. 8.33. Einfügen eines Blocks der Länge 3 ist erst nach Verlagerung des Blocks Nr. 2 möglich.

R	M	Bedeutung
0	0	nicht referenziert und nicht verändert
0	1	nicht referenziert aber verändert
1	0	referenziert aber nicht verändert
1	1	referenziert und verändert

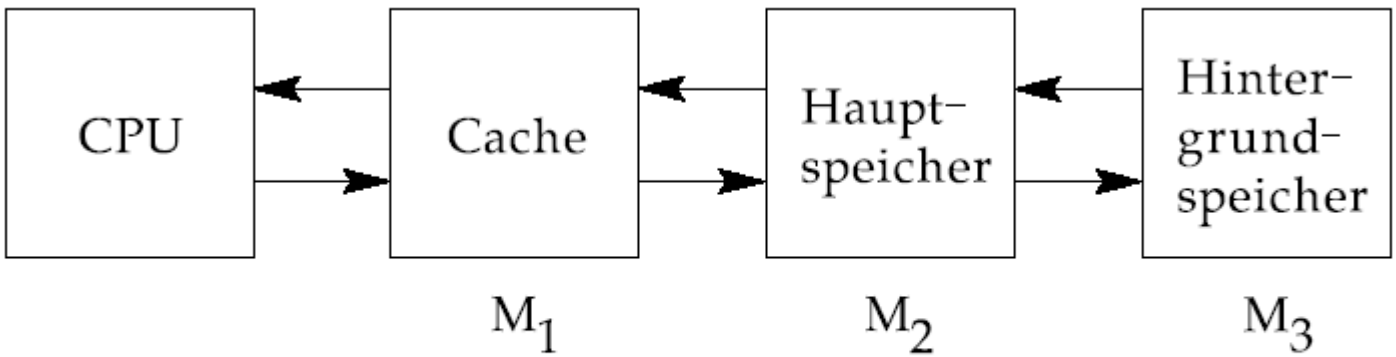


Abb. 8.34. Dreistufige Speicherhierarchie mit Cache

9. Ein-/Ausgabe und Peripheriegeräte

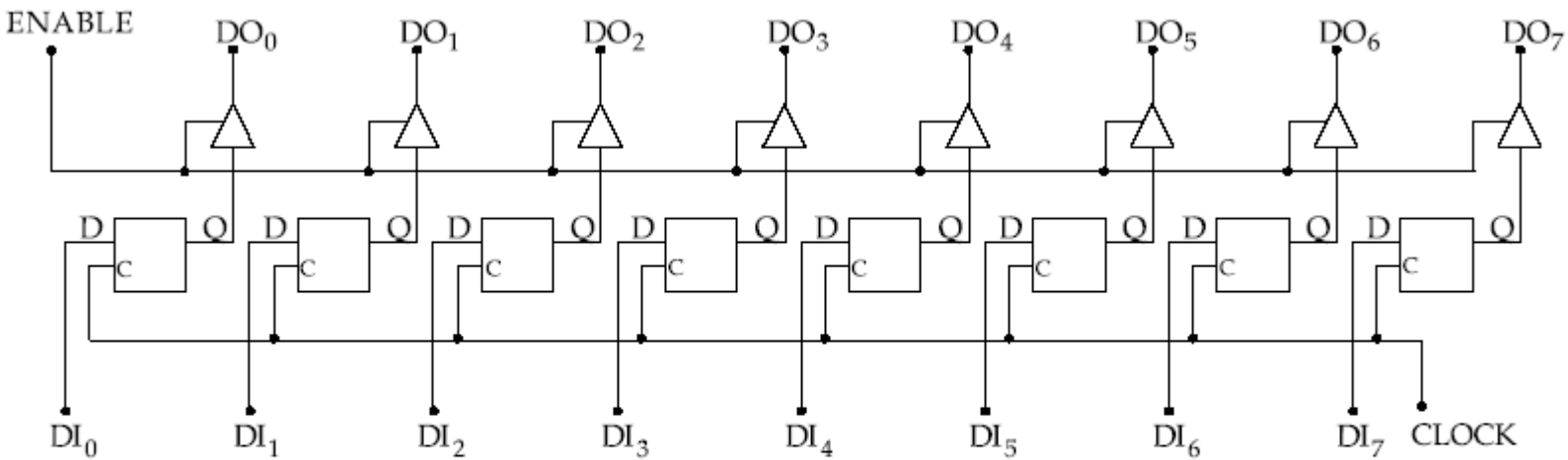


Abb. 9.1. Aufbau eines einfachen digitalen Bausteins zur byteweisen Eingabe oder Ausgabe.

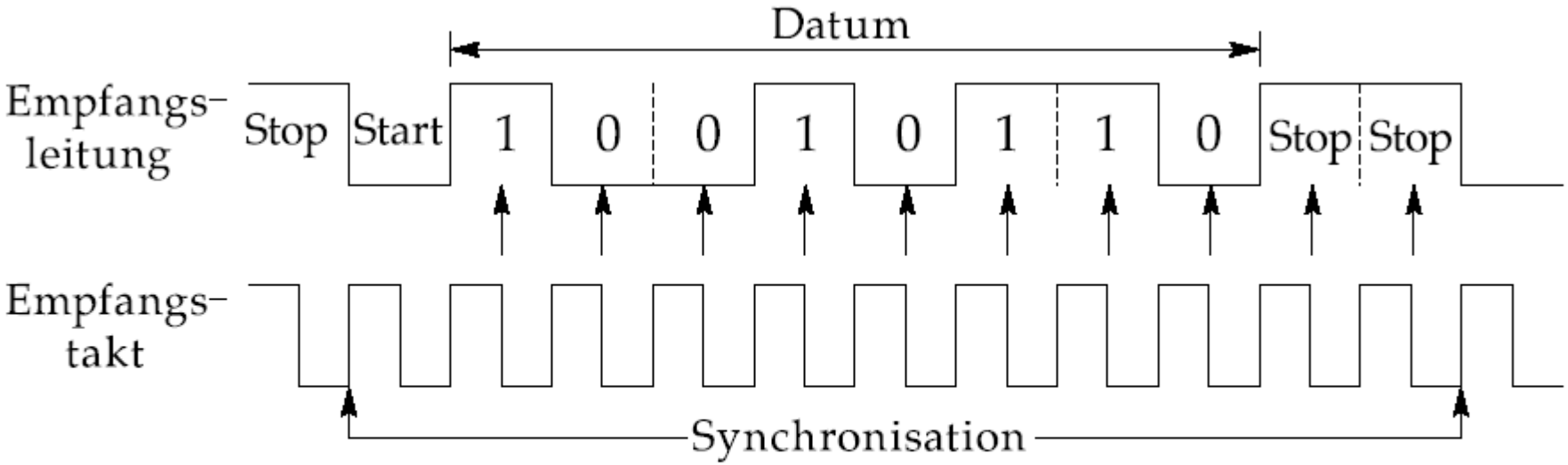


Abb. 9.2. Asynchrones Übertragungsformat mit 1 Start–Bit, 8 Daten–Bits und 2 Stopp–Bits

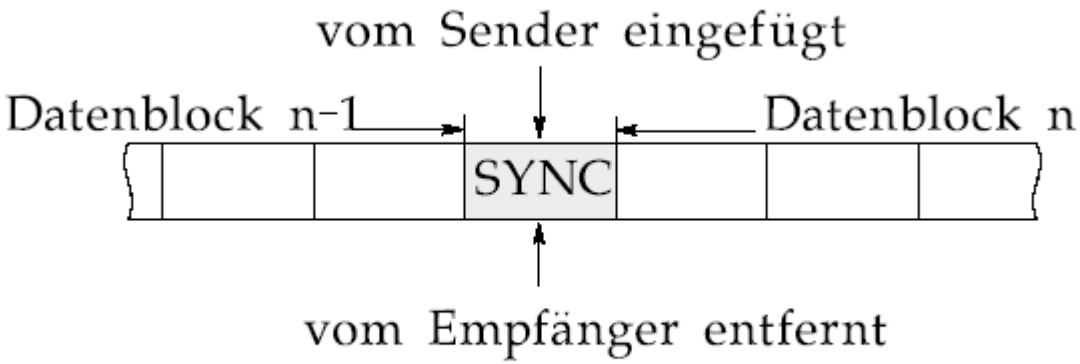


Abb. 9.3. Synchronbetrieb eines seriellen E/A–Bausteins

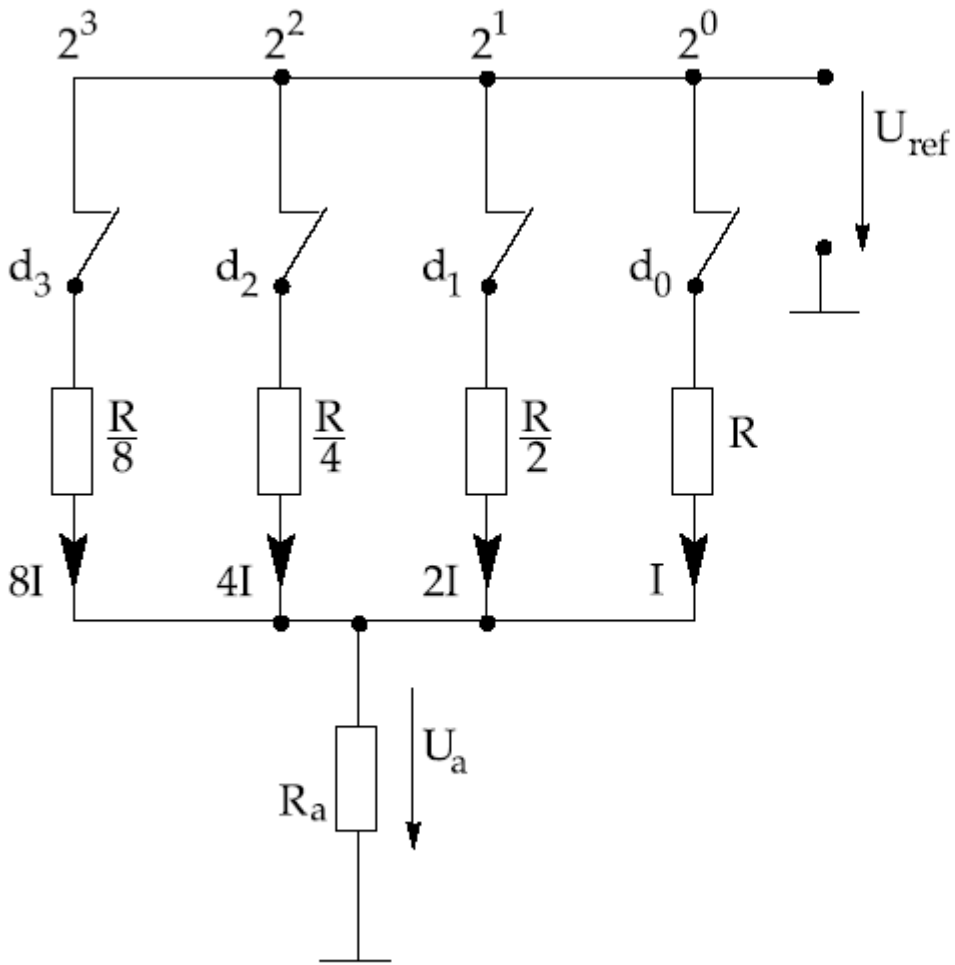
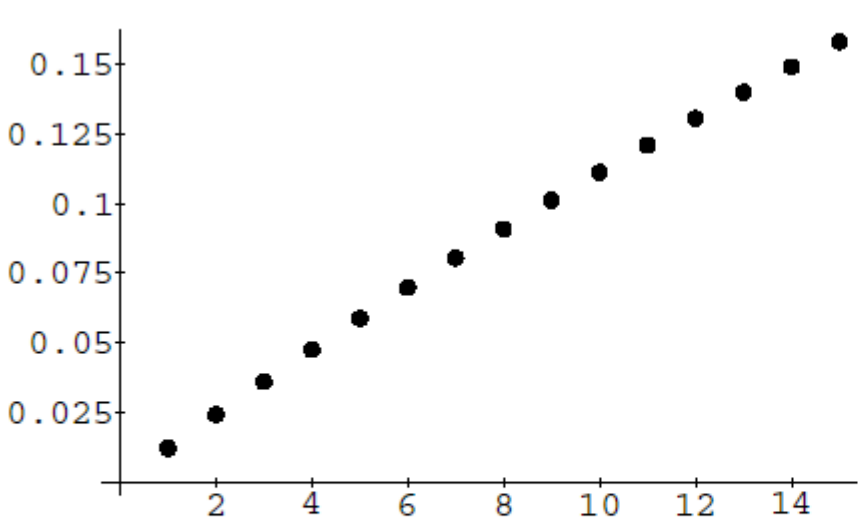


Abb. 9.4. 4–Bit D/A–Umsetzer mit abgestuften Widerstandswerten.



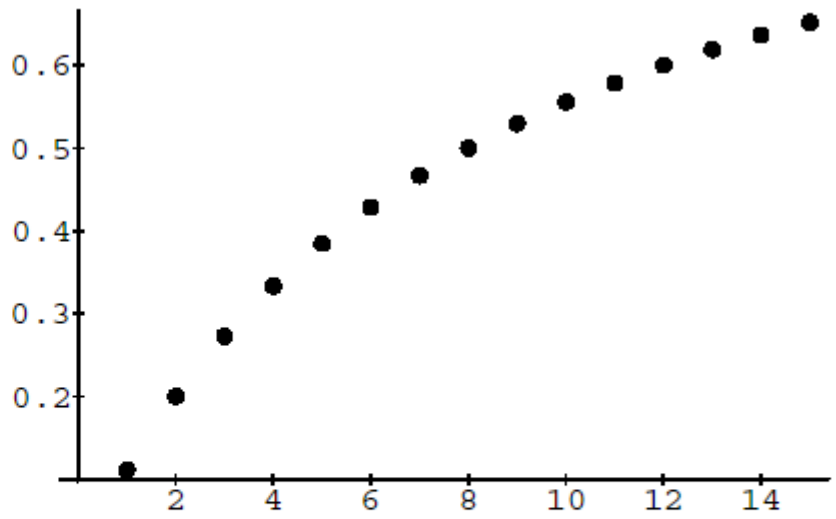


Abb. 9.5. Vergleich der Kennlinien zweier 4–Bit D/A–Umsetzer mit abgestuften Widerständen (R_a=1 KΩ, $U_{ref} = 10V$; links R=80 KΩ; rechts: R=8 KΩ). In beiden Kennlinien ist die Spannung U_a in V über dem dezimalen Eingabewert abgetragen.

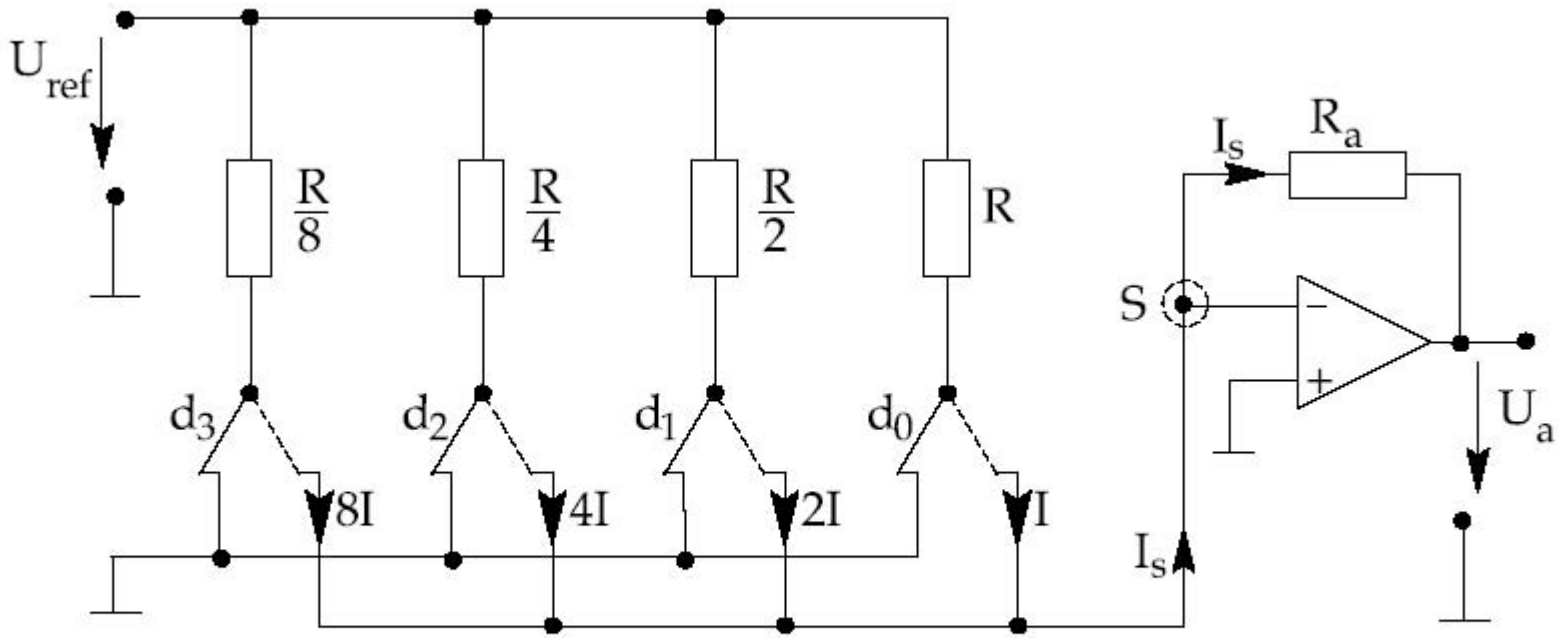
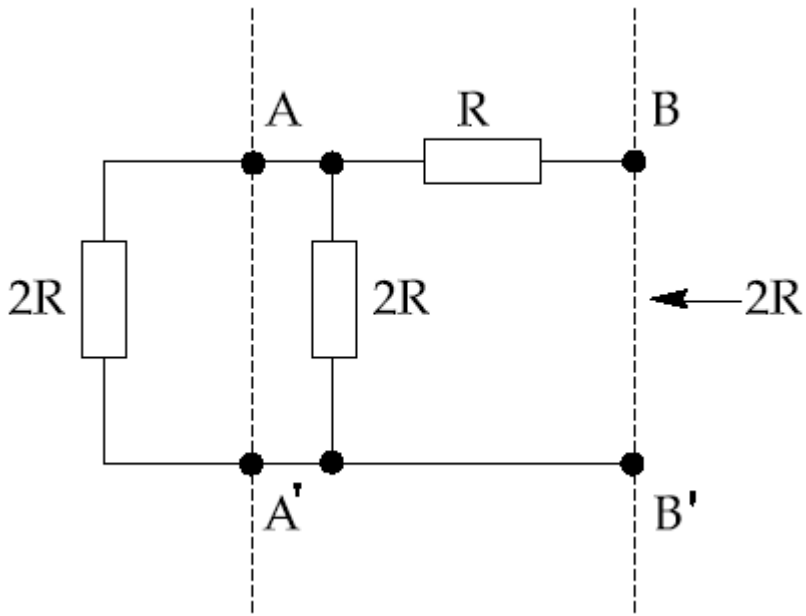


Abb. 9.6. Aufbau eines 4–Bit D/A–Umsetzers mit abgestuften Widerstandswerten und Operationsverstärker



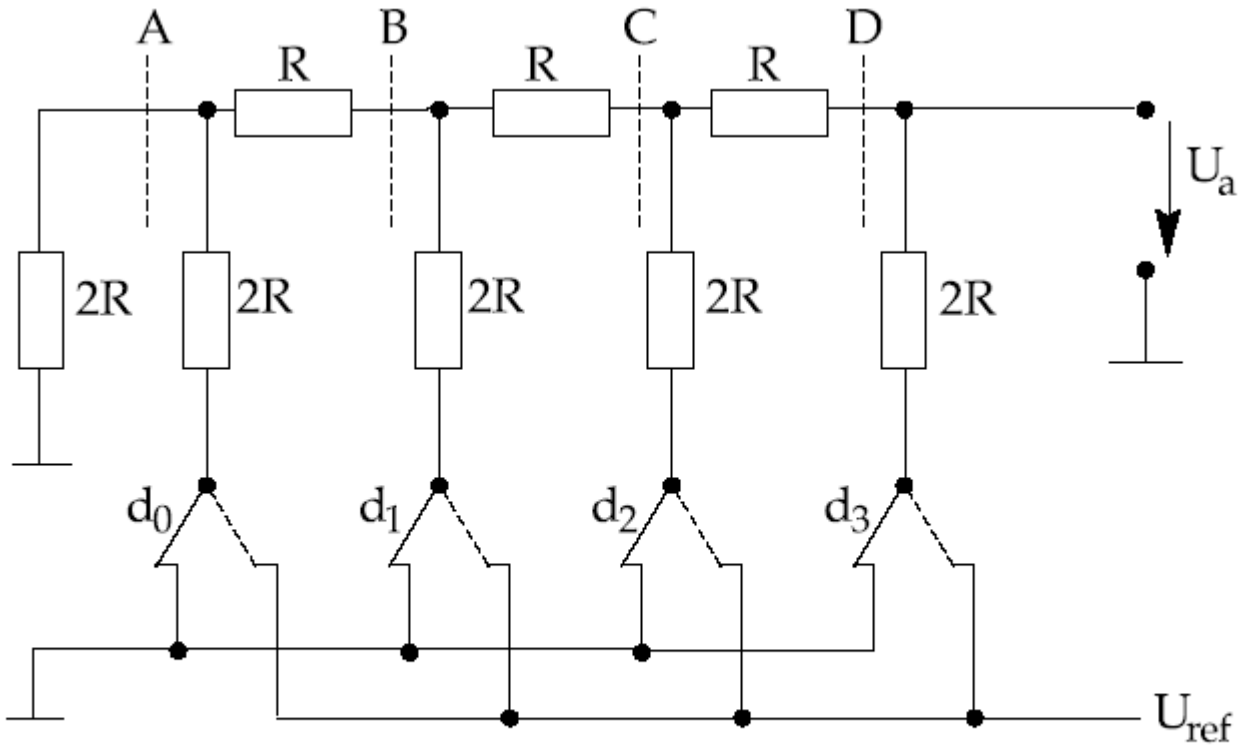
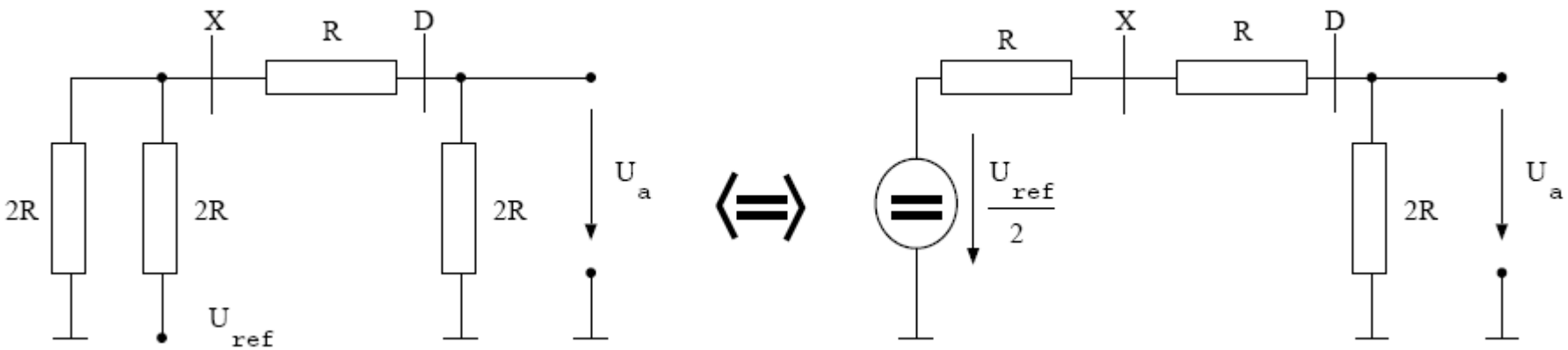


Abb. 9.7. Aufbau eines R–2R D/A–Umsetzers für eine Wortbreite von 4–Bit.



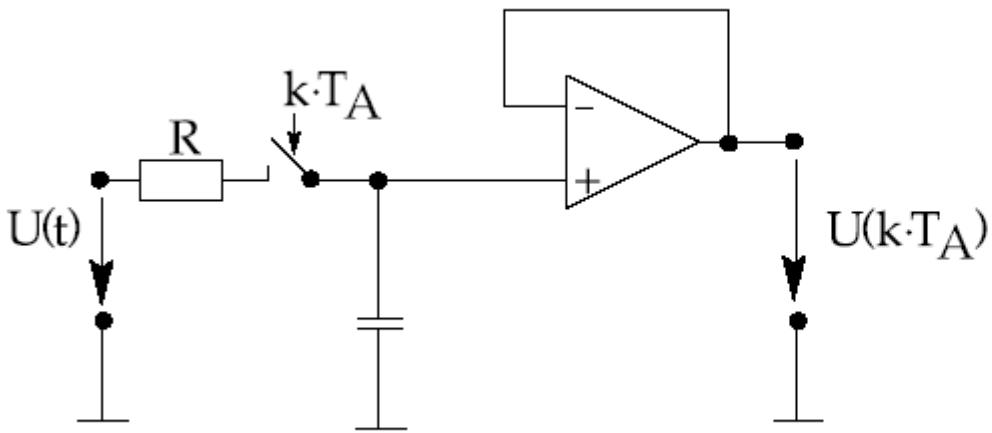


Abb. 9.8. Schaltung eines S&H–Gliedes. T_A bezeichnet die Periodendauer des Abtastsignals

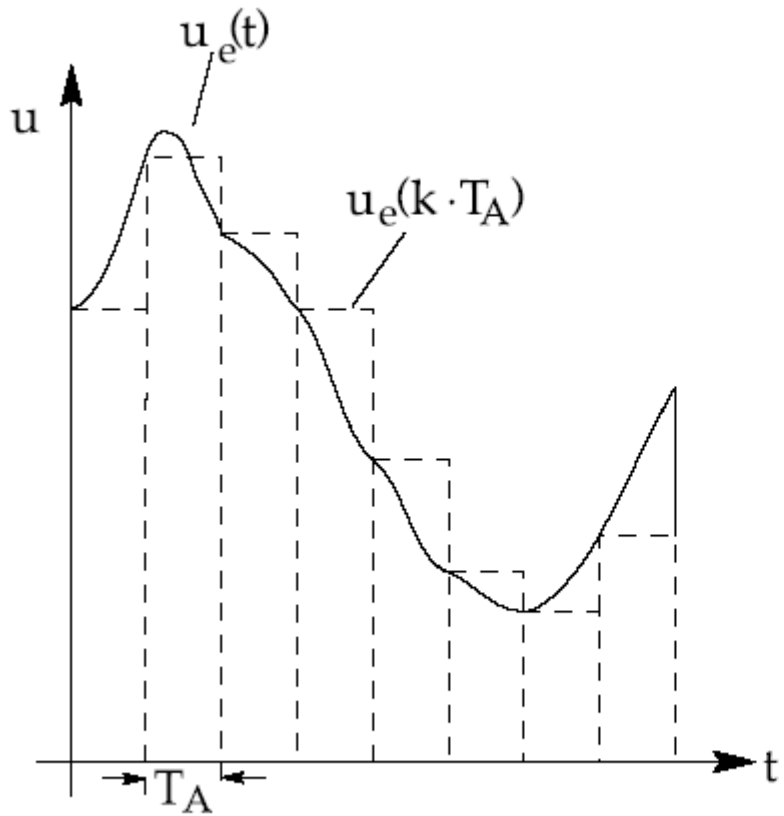
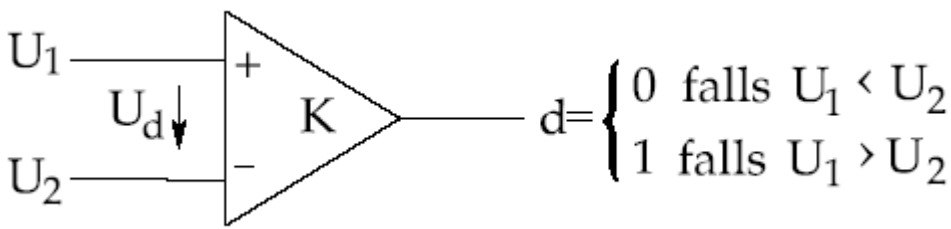
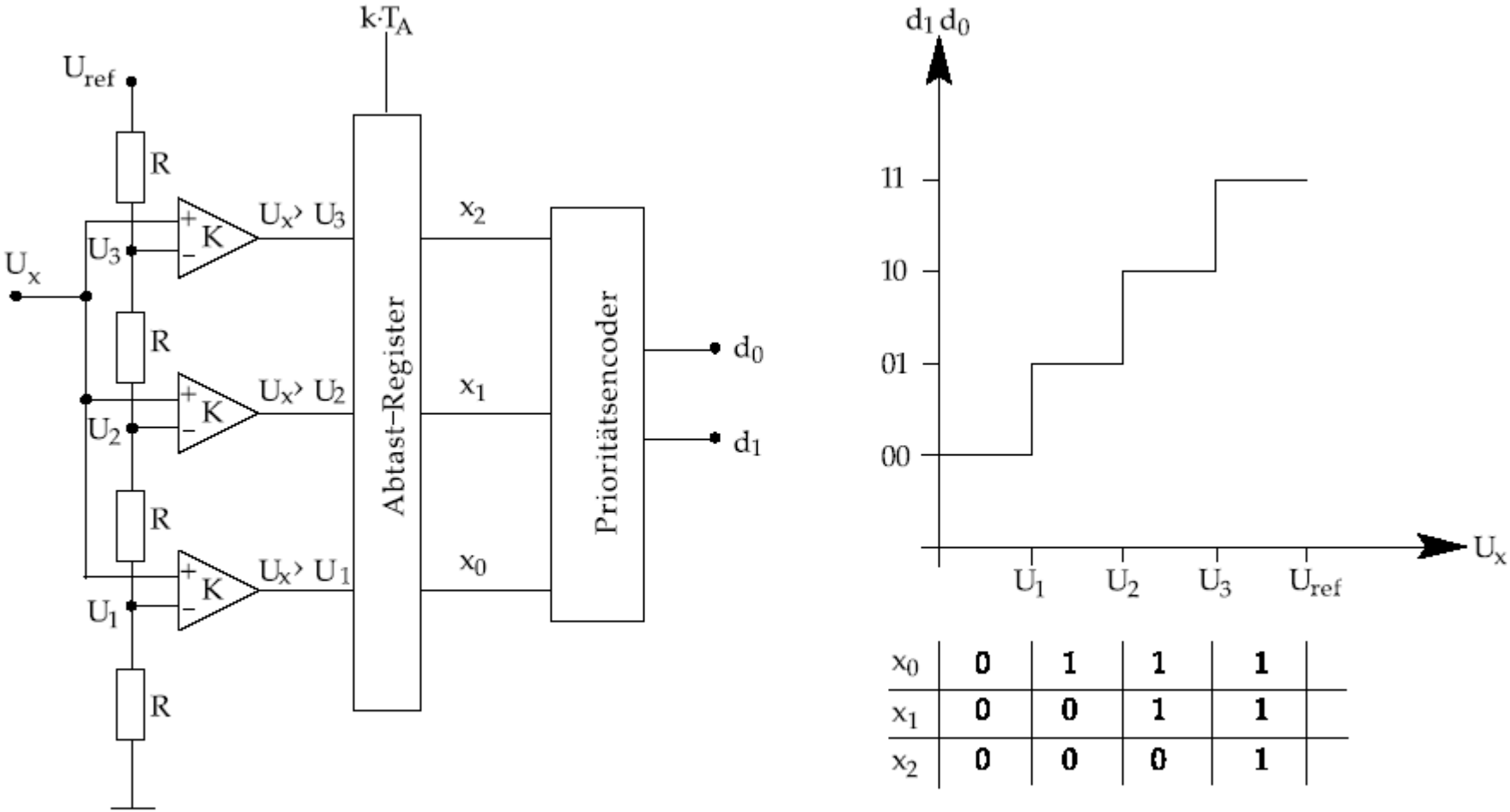


Abb. 9.9. Abtastung eines Signals in äquidistanten Abständen.





x_0	0	1	1	1	
x_1	0	0	1	1	
x_2	0	0	0	1	

Abb. 9.10. Aufbau eines parallelen A/D–Umsetzers mit 2–Bit Auflösung. Rechts: Kennlinie.

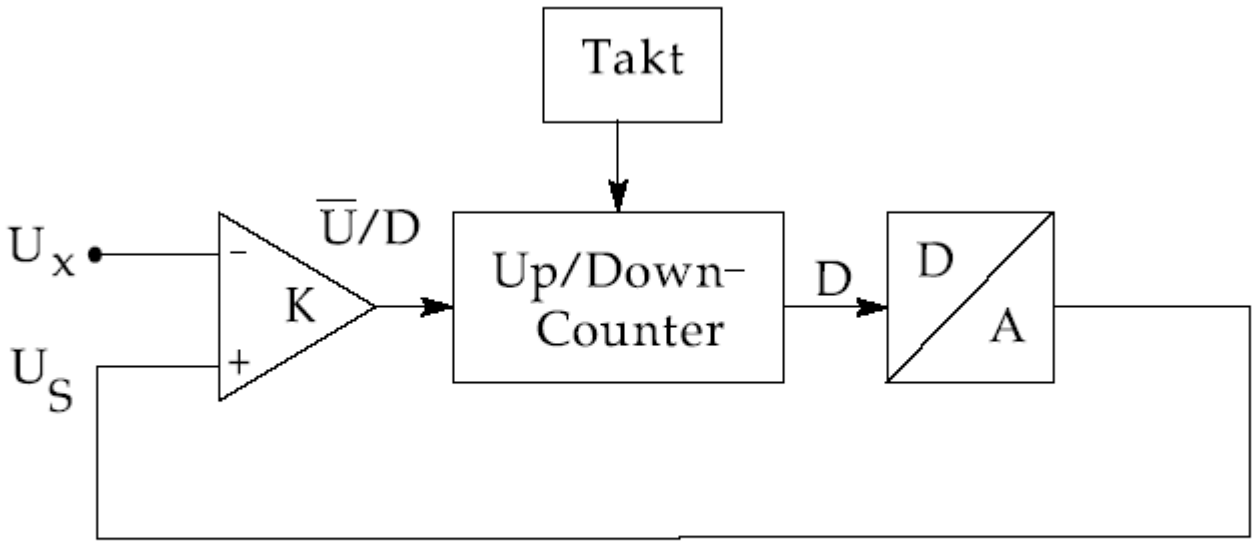


Abb. 9.11. Prinzip eines A/D–Umsetzers mit Vor–/Rückwärts Zähler.

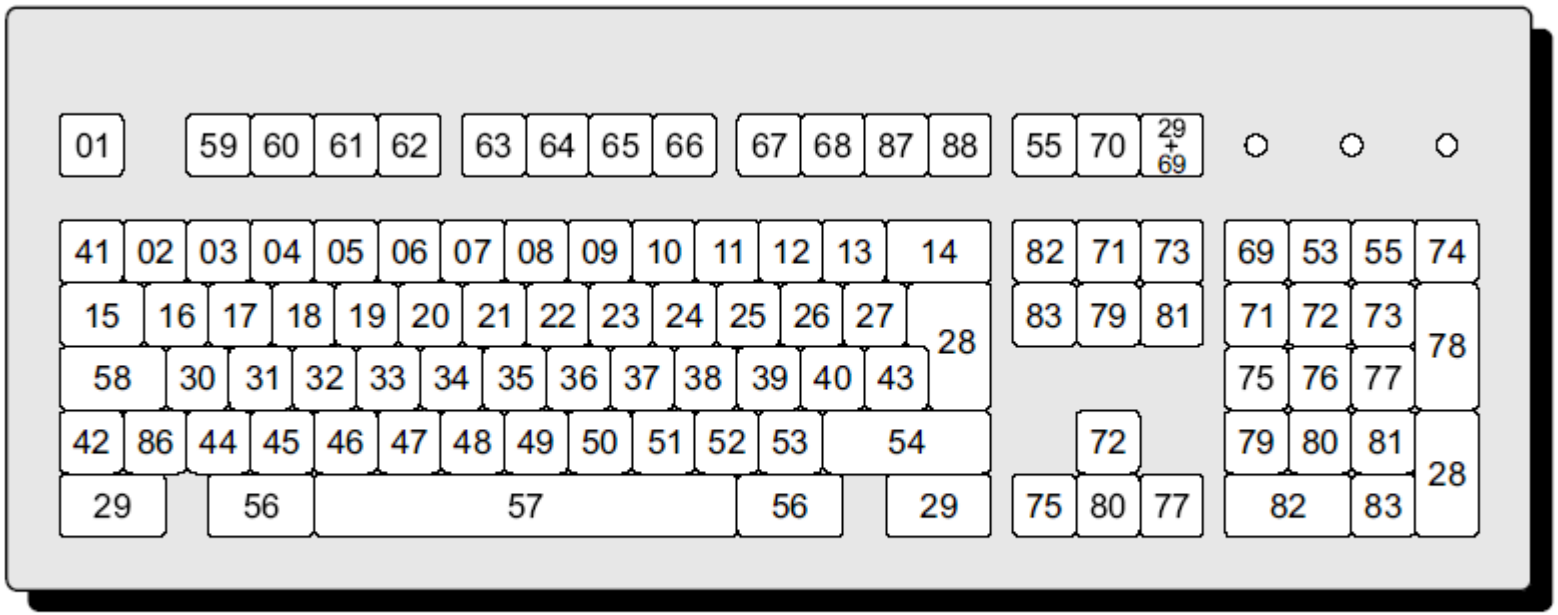


Abb. 9.12. Tastenlayout und Scancodes einer MF II–Tastatur

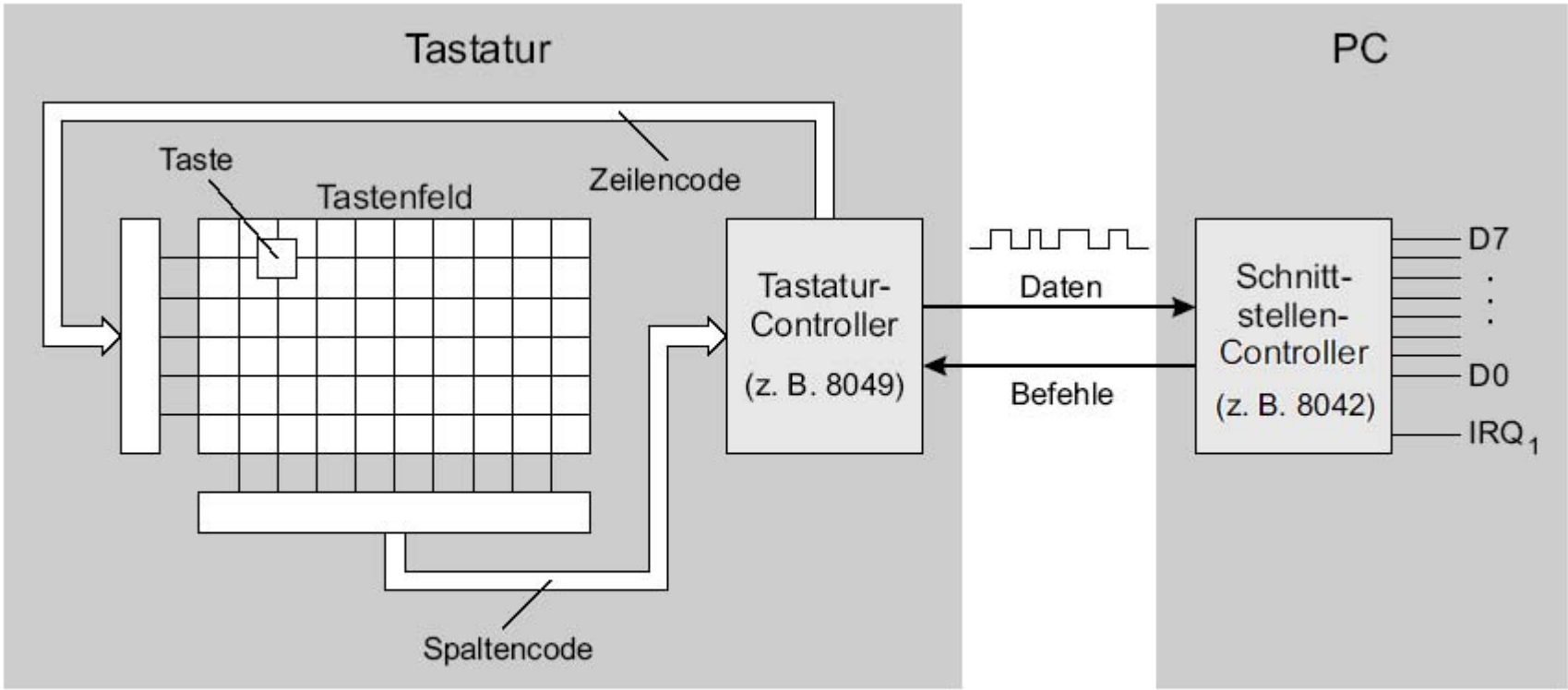


Abb. 9.13. Datenaustausch zwischen Tastatur– und Schnittstellen–Controller.

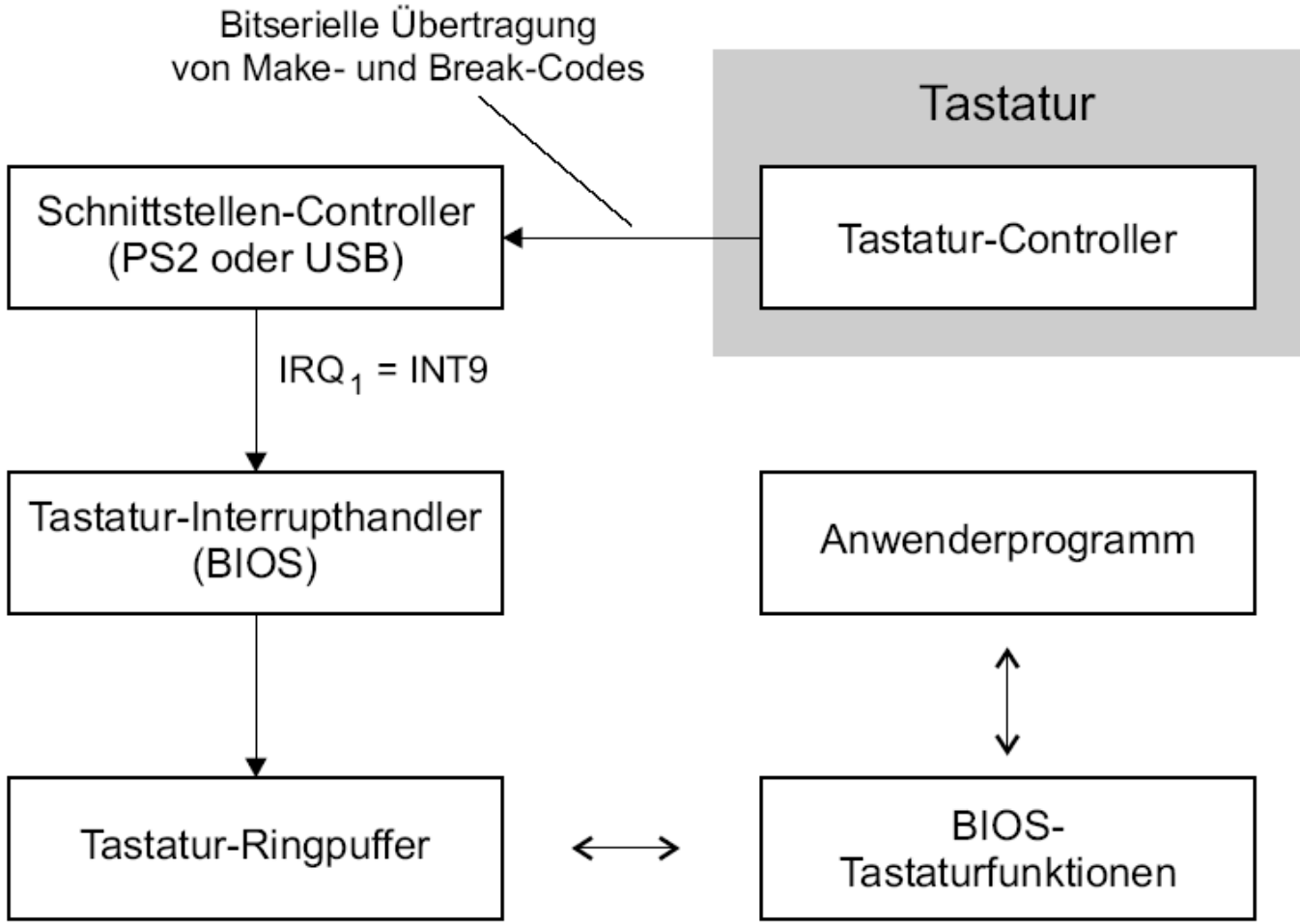


Abb. 9.14. Speicherung von Make– und Break–Codes im Tastatur–Ringpuffer

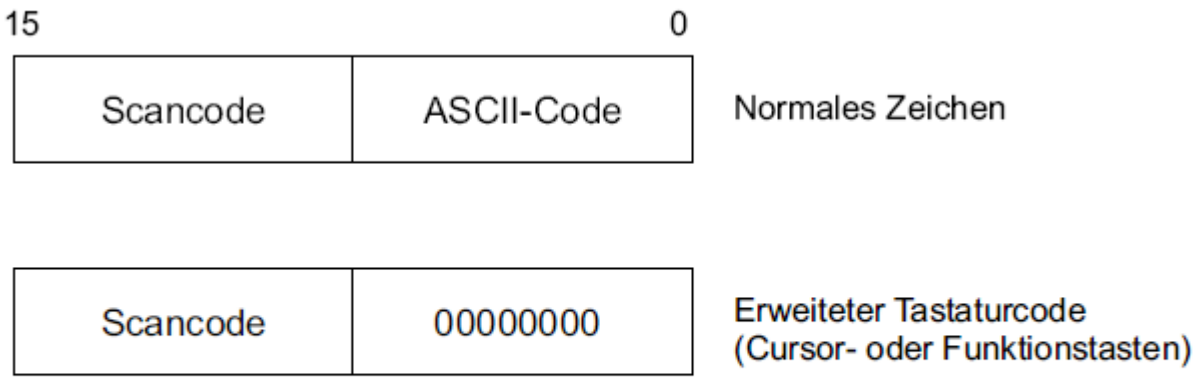


Abb. 9.15. Aufbau eines Eintrags im Tastatur–Ringpuffer

Tabelle 9.1. Funktionen des BIOS–Interrupts 16_H

Code	Aufgabe
00_H	Nächstes Zeichen lesen
01_H	Pufferstatus ermitteln
02_H	Zustand der Umschalttasten ermitteln
03_H	Verzögerungszeit und Wiederholrate programmieren
05_H	Scan– und Zeichencode in den Tastaturpuffer schreiben
10_H	Lesen eines Zeichen von MF II–Tastatur
12_H	Zustand der Umschalttasten von MF II–Tastatur ermitteln

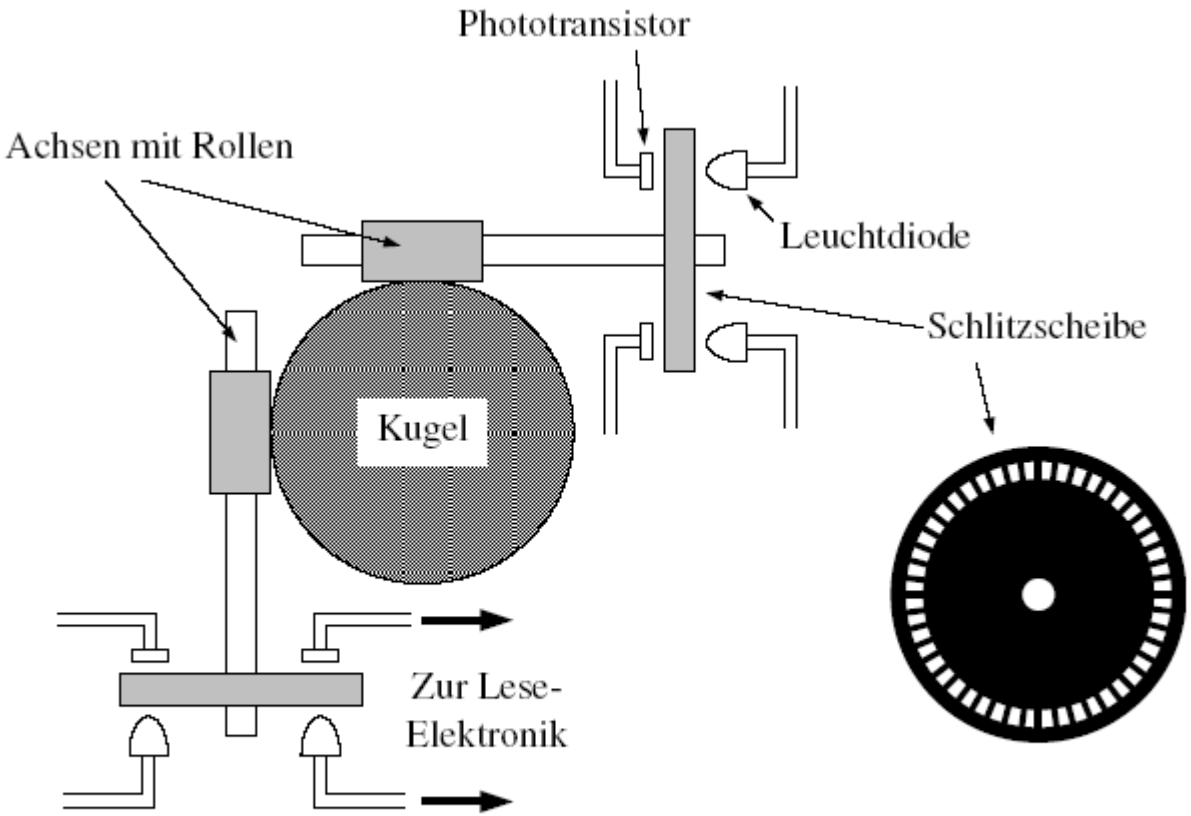


Abb. 9.16. Schematischer Aufbau einer Rollmaus

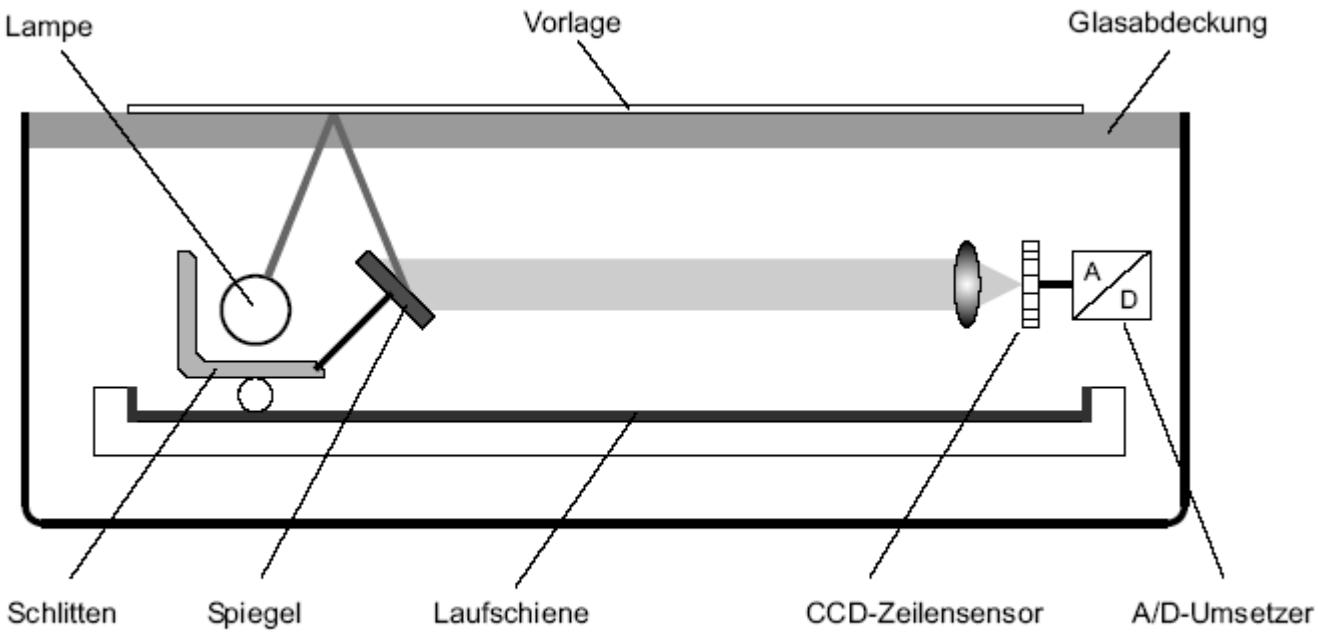


Abb. 9.17. Schematischer Aufbau eines Flachbettscanners

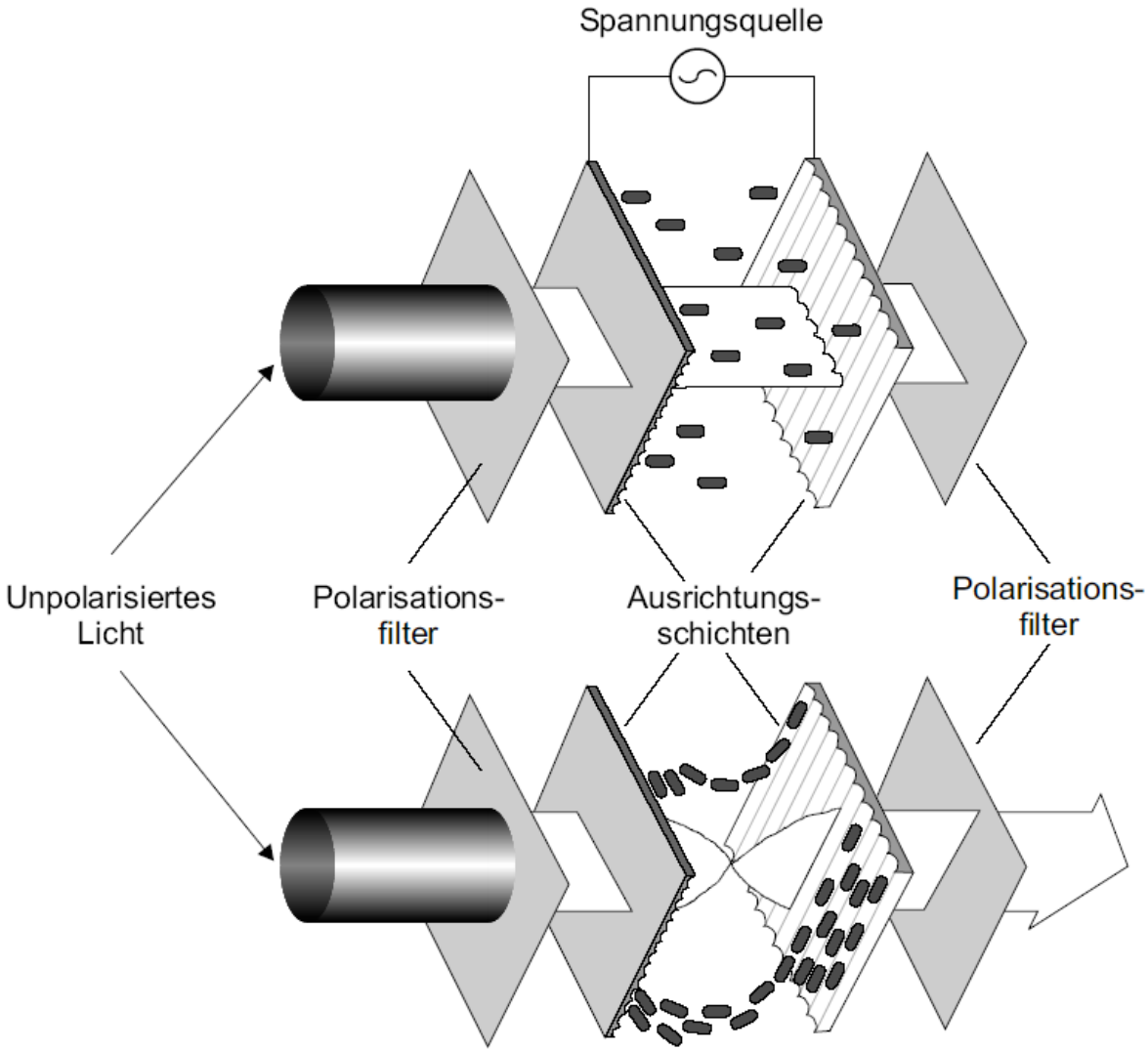


Abb. 9.18. Funktionsprinzip eines LCD–Bildschirms

Tabelle 9.2. Übersicht über Pixelfehlerklassen nach ISO 13406–2. Typ 1: ständig leuchtend; Typ 2: ständig dunkel; Typ 3: gemischt.

Klasse	Typ 1	Typ 2	Typ 3
I	0	0	0
II	2	2	5
III	5	15	50
IV	50	150	500

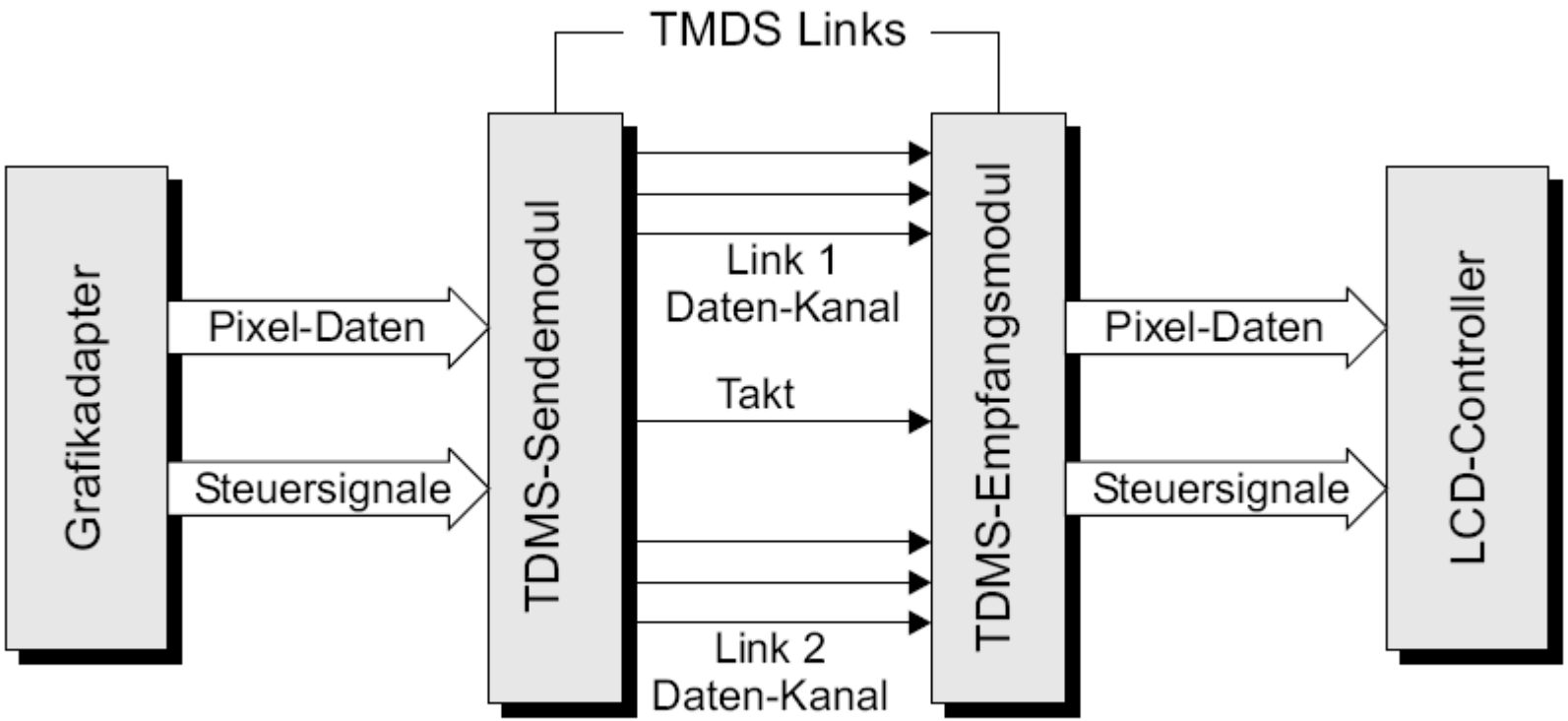
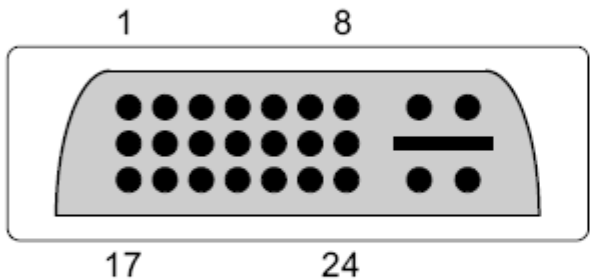


Abb. 9.19. Digitale Signale der DVI–Schnittstelle



Pin	Signal	Pin	Signal
1	Data2-	13	Data3+
2	Data2+	14	+5V
3	Data2/4 Shld	15	GND
4	Data4-	16	HPD
5	Data4+	17	Data0-
6	DDC-C	18	Data0+
7	DDC-D	19	Data0/5 Shld
8	(Analog Vs)	20	Data5-
9	Data1-	21	Data5+
10	Data1+	22	Cik Shld
11	Data1/3 Shld	23	Clock+
12	Data3-	24	Clock-

Abb. 9.20. DVI–Stecker mit Anschlussbelegung

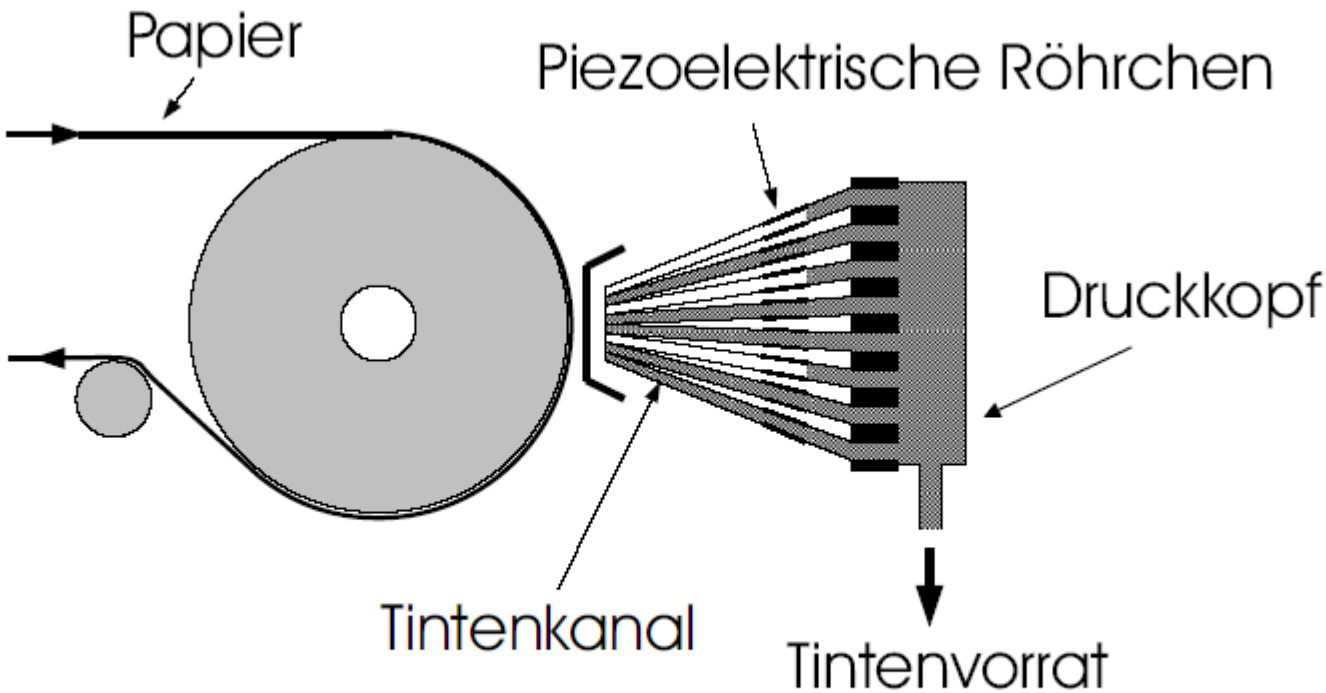


Abb. 9.21. Druckkopf eines Tintenstrahldruckers nach dem Piezo–Verfahren.

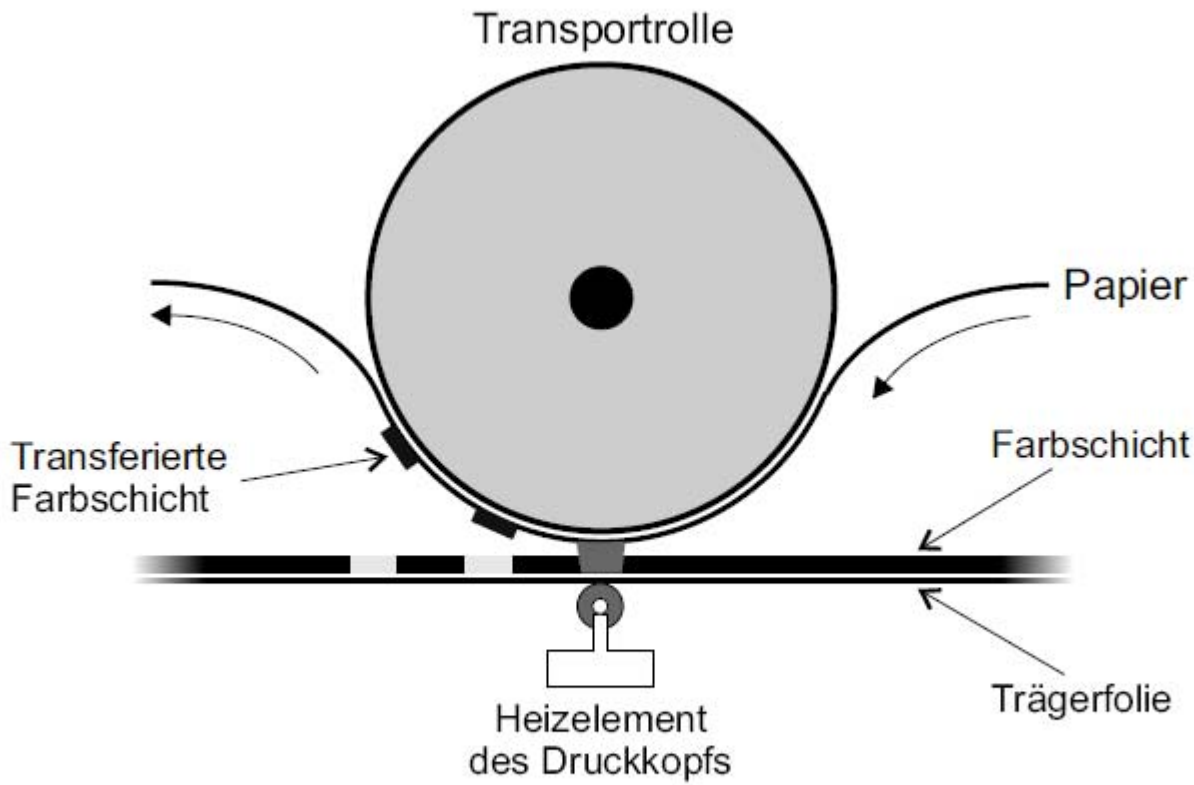


Abb. 9.22. Funktionsprinzip eines Thermosublimationsdruckers

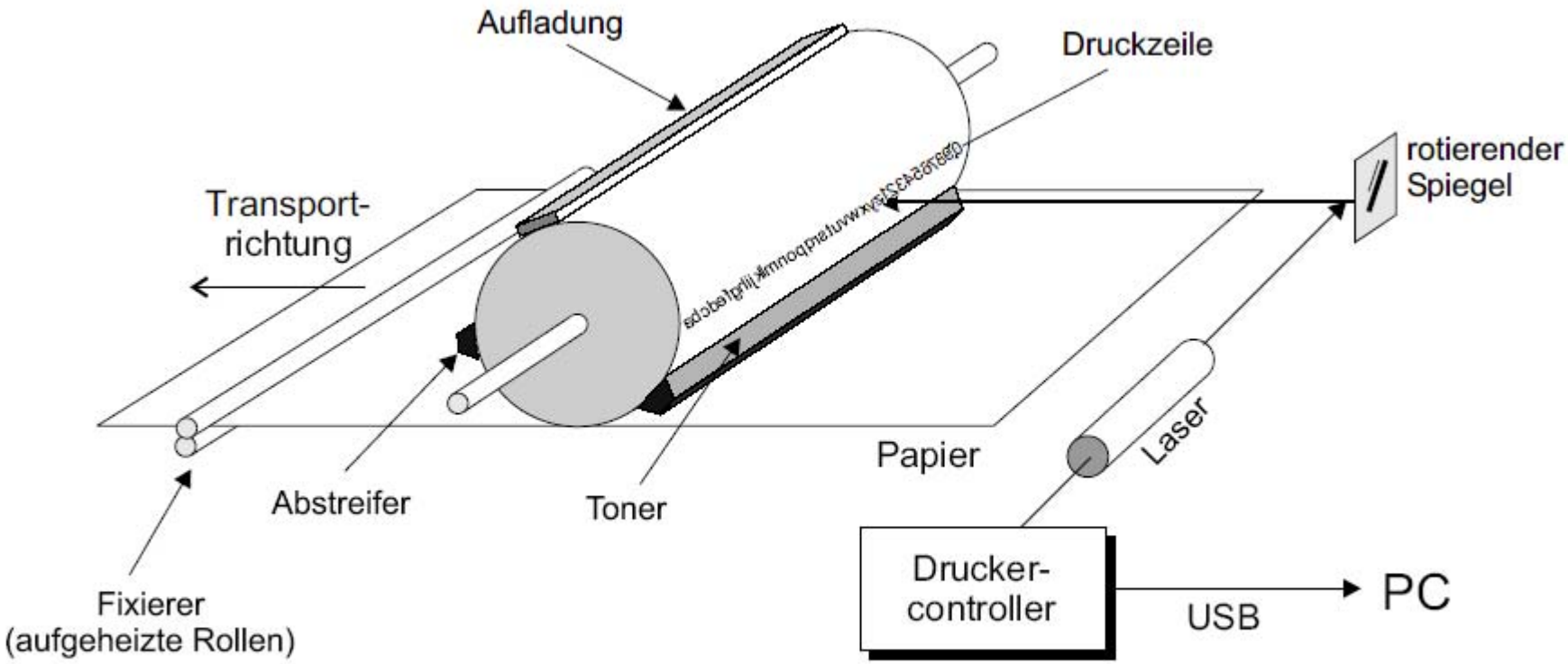


Abb. 9.23. Schematischer Aufbau eines Laserdruckers

10. Aktuelle Computersysteme

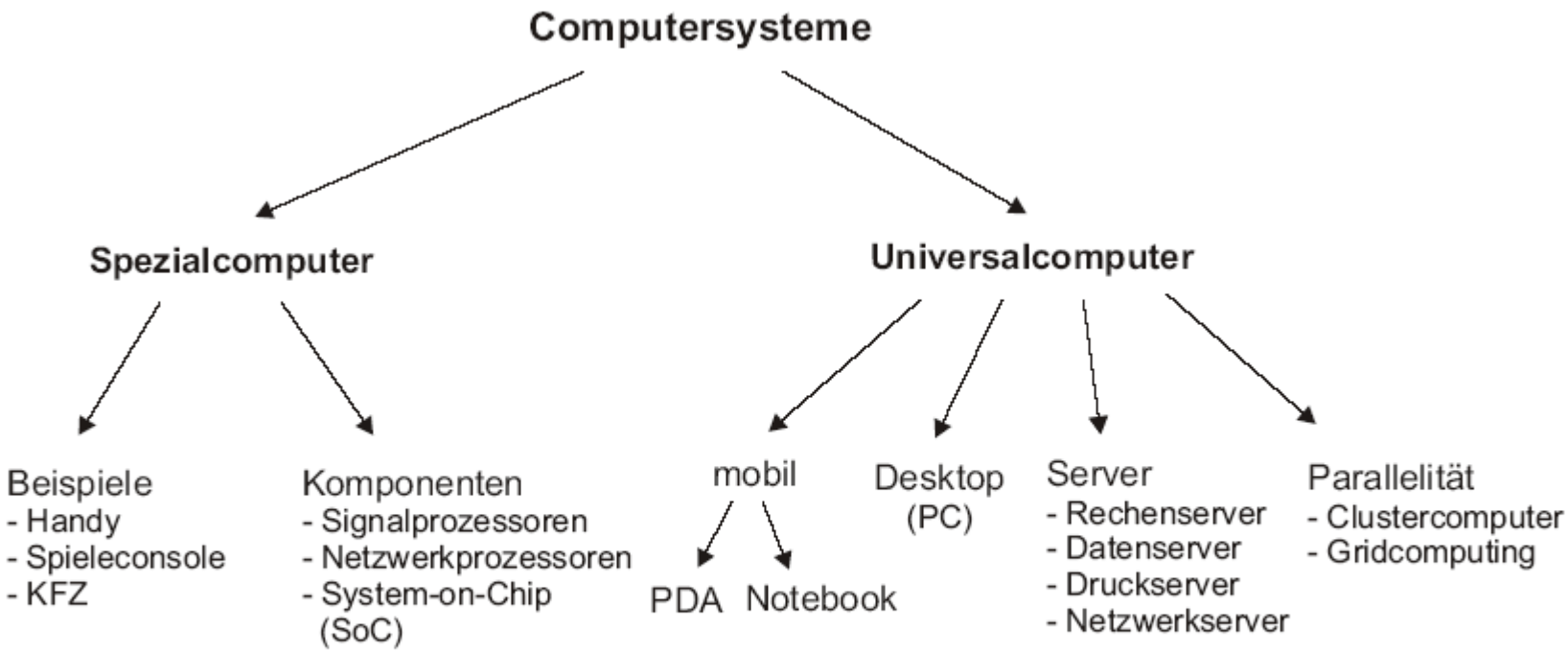


Abb. 10.1. Übersicht über die verschiedenen Arten von Computersystemen.

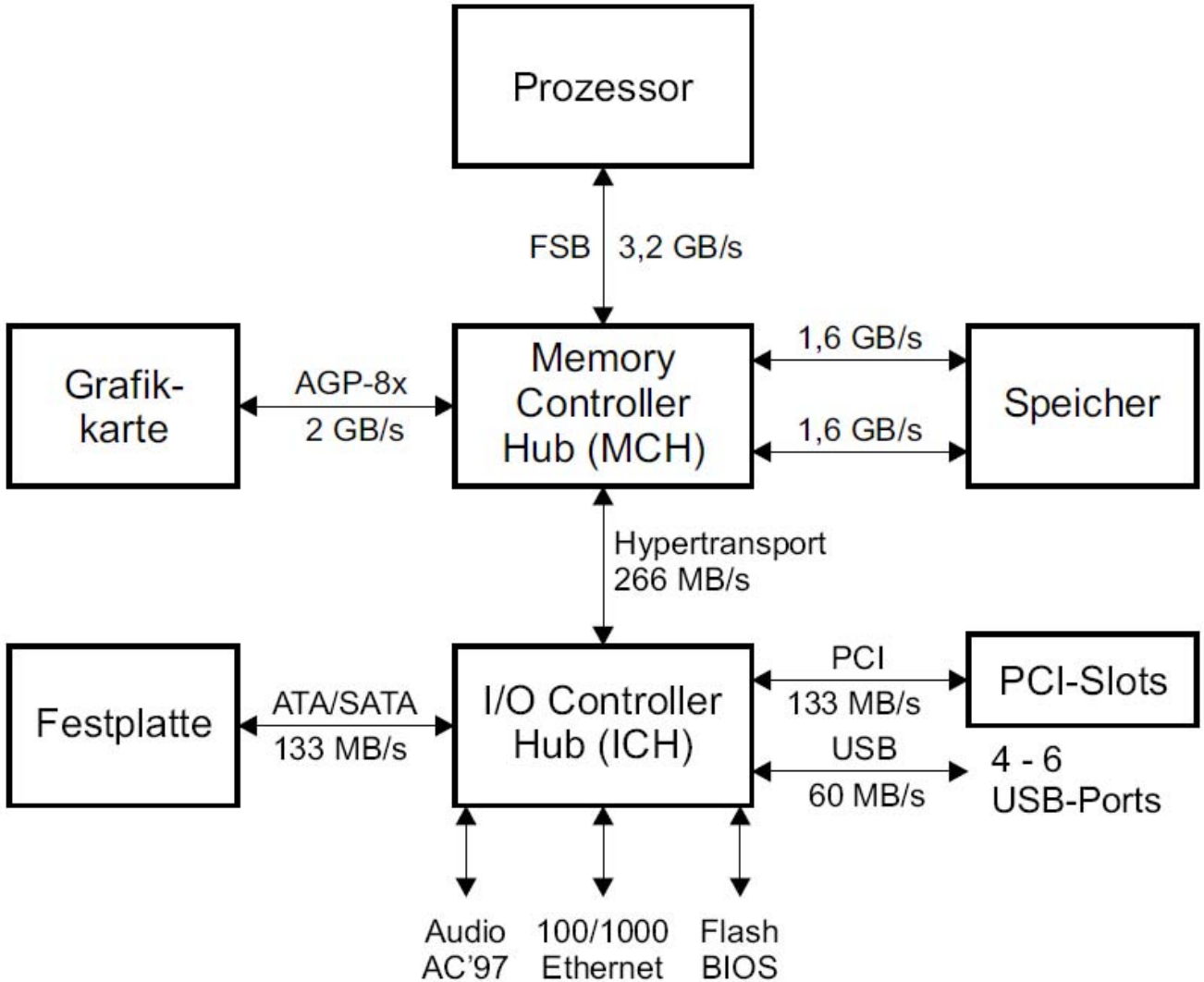


Abb. 10.2. Chipsätze synchronisieren den Prozessor mit Speicher– und Ein–/Ausgabebussen. Dargestellt ist der Aufbau eines Desktop–Computers nach der Intel Hub Architecture.

Tabelle 10.1. Leistungsvergleich der derzeit schnellsten Desktop–Prozessoren von AMD und Intel (nach [*Windeck, 2004*])

Prozessor	SYSmark 2004	CINT2000_base	CFP2000_base
Athlon 64 FX–53	198	1539	1426
Pentium 4 EE	201	1619	1526

Tabelle 10.2. Übersicht über die gebräuchlichsten DIMM–Speicher (Stand: Dezember 2004)

	DDR 333 PC 2700	DDR 400 PC 3200	DDR 533 PC 4200
Wortbreite	64 Bit	64 Bit	64 Bit
phys. Takt	166 MHz	200 MHz	266 MHz
effekt. Takt	333 MHz	400 MHz	533 MHz
einkanalig	2,7 GByte/s	3,2 GByte/s	4,2 GByte/s
zweikanalig	5,4 GByte/s	6,4 GByte/s	8,4 GByte/s

A. Kurzreferenz Programm opw

Befehl	Abk.	Funktion	Beispiel
`S = Steuerwort`	`s =`	Festlegen des Steuerwortes.	`s = xxxx01011x0`
`clock`	`c`	Takten des Operationswerkes.	`clock`
`dump`	`d`	Ausgabe der Registerinhalte des Operationswerkes	`dump`
`X = Konstante`	`x =`	Setzt den Eingang X auf bestimmte Werte.	`x = $-5`
`Y = Konstante`	`y =`	Setzt den Eingang Y auf bestimmte Werte.	`y = $1e`
`quit`	`q`	Beendet die Simulation.	`quit`
`EQ? Marke`	`eq?`	Springt zur Marke, wenn die Summanden des Addierers gleich sind.	`EQ? equal`
`NEQ? Marke`	`neq?`	Springt zur Marke, wenn die Summanden des Addierers ungleich sind.	`NEQ? loop`
`PLUS? Marke`	`plus?`	Springt zur Marke, wenn das höchstwertigste Bit des Ergebnisses des Addierers gesetzt ist.	`PLUS? plus`
`MINUS? Marke`	`minus?`	Springt zur Marke, wenn das höchstwertigste Bit des Ergebnisses des Addierers nicht gesetzt ist.	`MINUS? minus`

Konstruktor	Funktion	Beispiel
:	Trennen von Befehlen, die in einer Zeile stehen.	`clock : dump`
;	Text, der dem Semikolon folgt, wird vom Programm ignoriert (Kommentare).	`dump ; Ergebnis ausgeben`
>	Definieren einer Marke.	`>loop`

B. Kurzreferenz Programm ralu

Befehl	Abk.	Funktion	Beispiel
`control` *Steuerwort*	`co`	Festlegen des Steuerwortes.	`co $14321`
`clock`	`c`	Takten der RALU.	`clock`
`dump`	`d`	Ausgabe der Registerinhalte etc.	`dump`
`set` *Reg-nr. Konstante*	`s`	Setzt Register auf bestimmte Werte.	`set 2 #-5`
`quit`	`q`	Beendet das RALU–Programm.	`quit`
`carry` 0/1	`cy`	Das Carry–Flag wird gesetzt oder gelöscht.	`carry 1`
`jmpcond` *Prüfmaske Marke*	`jc`	Springt zur Marke, wenn die UND–Verknüpfung zwischen Status und Maske ungleich null ist.	`jc $40 loop`
`jpncond` *Prüfmaske Marke*	`jnc`	Springt zur Marke, wenn die UND–Verknüpfung zwischen Status und Maske gleich null ist.	`jnc $40 lb`

Konstruktor	Funktion	Beispiel
:	Trennen von Befehlen, die in einer Zeile stehen.	`clock : dump`
;	Text, der dem Semikolon folgt, wird vom Programm ignoriert (Kommentare).	`cy 0 ; Carry löschen`
>	Definieren einer Marke.	`>loop`