SKRIPTEN ZUR INFORMATIK

Rainer W. Schulze

Technische Grundlagen der Rechnerarchitektur

Im Script *Einführung in die Prozessorarchitektur* wird die grundlegende, jedoch nicht typenbezogene Arbeitsweise eines Mikroprozessors dargelegt. Das vorliegende Script *Technische Grundlagen der Rechnerarchitektur* vermittelt einen Einblick in den Entwurf von Mikrorechnersystemen und basiert auf den Architekturmerkmalen eines zwar elementaren, jedoch existierenden Prozessors. Auf diesem Niveau erschien es angebracht, dafür den Mikroprozessor Z80 zu favorisieren. Dessen Systemfamilie gestattet es in ihrer schlichten Architektur sehr anschaulich, das Signalspiel in einem Mikrorechnersystem zu erörtern. Sie bietet darüber hinaus aber auch einen Einblick in Möglichkeiten, Komponenten verschiedener Systemfamilien miteinander zu verkoppeln. Im vorliegenden Fall handelt es dabei um ausgewählte INTEL-Schaltkreise. Auf dieser Basis werden abschließend *Architekturen parallel strukturierter Rechnersysteme* in kurzer Form umrissen.

Das Script stellt eine Unterstützung technisch orientierter Lehrveranstaltungen an der TU Dresden dar. Herrn Dr. S. Schöne, Institut für Technische Informatik an der Fakultät Informatik, sei für die Durchsicht des Manuskripts an dieser Stelle herzlich gedankt.

Inhalt

1. TURING-MASCHINE

Elementares mathematisches Modell zur Algorithmenimplementierung ist die Turingmaschine (Abb. 1/1), bestehend aus einem *Lese/Schreibkopf* und einer *Steuereinheit*, auch bezeichnet als *Turingtafel*. Unter dem Lese/Schreibkopf läuft ein einseitig beschriebenes, unendlich langes Band entlang. Die Steuereinheit veranlaßt *Le*sen, *Schreiben* und *Bewegen* des Bandes.

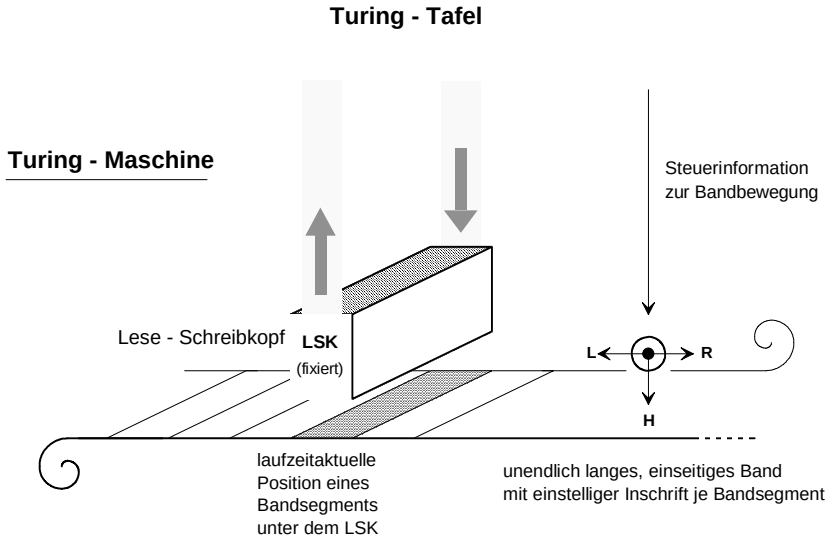


Abb. 1/1 Turingmaschine

Bandalphabet, Positions-, *Ereignis* und *Zustandsmenge* charakterisieren die Turingmaschine.

- **Bandalphabet** $\mathbf{B} = \{b_1, b_2, \ldots\ldots\ldots\ldots, b_m \;,\; b_0\}$
  Inschrift für ein Bandsegment — Leerzeichen

- **Positionsmenge** $\mathbf{P} = \{p_0, p_1, \ldots., p_{l-1}, p_l, p_{l+1}, \ldots, p, \ldots, p_{L-1}\}$
  ist eine linear gordnete Menge, deren Elemente p *Bandpositionen* sind.

Es sei $e_{[p]} \in \{ \in \mathbf{B} \mid \nu = 1, 2, \ldots, L \} = \mathbf{P} \times \mathbf{B}$
Eintrag auf der Bandposition p.
Jedem *Bandsegment* ist umkehrbar eindeutig ein Element $p \in \mathbf{P}$ als Bezeichner zugeordnet;
Die Aufeinanderfolge der Bandpositionen erfolgt gem.

Steuerinformation H/L/R

$H : p_l \rightarrow p_l \in \mathbf{P}$
$L : p_l \rightarrow p_{l-1} \in \mathbf{P}$
$R : p_l \rightarrow p_{l+1} \in \mathbf{P}$

- **Ereignismenge** $\mathbf{K} = \{k_0, k_1, ..., k, k'_l ..., K\}$
  ist eine geordnete Menge, deren Elemente k *Ereignislagen* der Turing-Maschine sind.

  Sei ${}^{k}e_{[p]}$ (bzw. ${}^{k'}e_{[p]}$)
  Eintrag auf *Bandposition* p in der *Ereignislage* k (bzw. k') .

- **Zustandsmenge** $\mathbf{Z} = \{z_1, z_2, ..., z_n , \underset{\text{Anfangszustand}}{z_0}\}$

  ist eine Menge, deren Elemente z *Zustände* heißen und genau einer Ereignislage k zugeordnet sind, d.h. ${}^{k}z$ (bzw. ${}^{k'}z$) ist Zustand der Turing-Maschine in der *Ereignislage* k (bzw.k').

Die Abbildung $\Delta_1 : ({}^{k}\mathbf{e}_{[p]}, {}^{k}\mathbf{z}) \rightarrow ({}^{k'}\mathbf{e}_{[p]}, {}^{k'}\mathbf{z}, \mathbf{L/R/H})$ bestimmt die Progression der Turingmaschine. Im Detail ist die Abbildung Δ gesplittet in

- $\Delta_1 : ({}^{k}\mathbf{e}_{[p]}, {}^{k}\mathbf{z}) \rightarrow {}^{k'}\mathbf{e}_{[p]}$: Rechenwerk
- $\Delta_2 : ({}^{k}\mathbf{e}_{[p]}, {}^{k}\mathbf{z}) \rightarrow {}^{k'}\mathbf{z}$: Programmsteuerwerk
- $\Delta_3 : ({}^{k}\mathbf{e}_{[p]}, {}^{k}\mathbf{z}) \rightarrow (\mathbf{L/R/H})$: Systemsteuerwerk

Rechenwerk, Programm- und Systemsteuerwerk sind die grundlegenden Komponenten eines *Prozessors*. Zusammen mit dem *Hauptspeicher* konfigurieren sie eine rechenfähige Anordnung (Abb. 2/1).

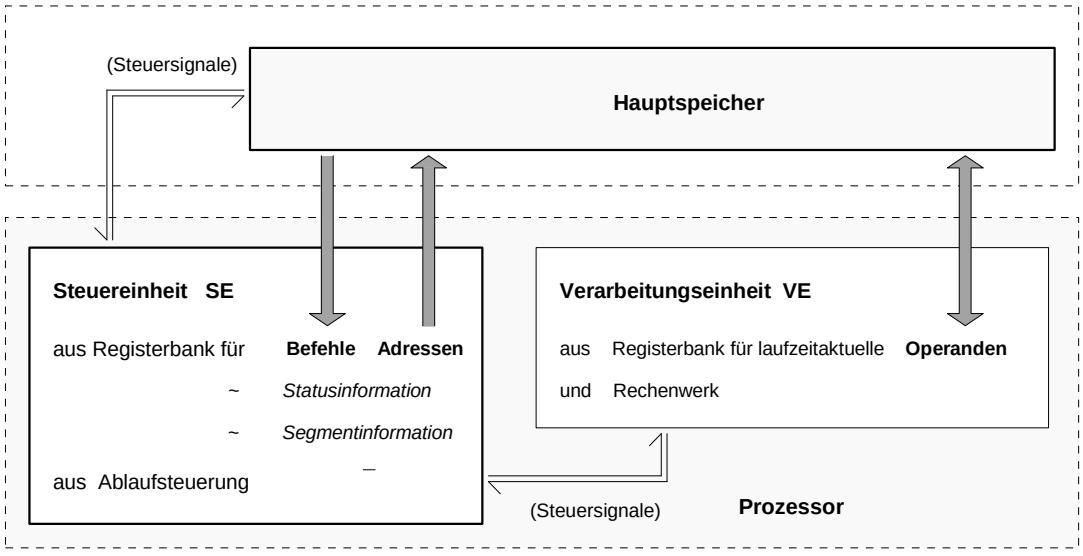


Abb. 2/1 Rechenfähige Anordnung aus *Prozessor* und *Hauptspeicher*

2. SERIELL STRUKTURIERTE RECHNERSYSTEME

Die Ausführungen im Kapitel beziehen sich auf ZILOG- und INTEL-Prozessoren in der jeweiligen Version, dokumentiert in entsprehenden Firmenschriften. Hier erfolgt eine Zusammenfassung vielfältiger Darlegungen. Ausführungen zum Abschnitt *Befehlsaufbau und Adressierungsarten* sind der Literatur (s. Verzeichnis) entnommen und dort zu vertiefen.

2.1 Befehlsaufbau und Adressierungsarten

Befehl wird dargestellt als **Befehlswort**

Operationsarten:

- arithmetisch/logische Operationen
- Verschiebung/Rotation von Registerinhalten
- Teste, Setzen und/oder Löschen einzelner Bitstellen in Operanden- und Steuerwörtern
- Transportoperationen zwischen Speicherplätzen (Hauptspeicherzellen, Register)
- E/A-Operationen
- Organisations- und Programmablaufoperationen (Sprünge, Unterprogrammaufrufe)

Arbeitet ein Prozessor mit Befehlen abgestufter Länge, so ist diese Länge auch noch durch den Befehlstyp und die Art der Operandenadressierung bestimmt.

Adreßteil:

- zur Angabe der Operandenadresse(n) im Hauptspeicher oder in der Registerbank

Adressierungsarten

direkte Verfahren	Adresse ist im Adreßteil des Befehlswortes (direkt) enthalten	
indirekte Verfahren	(nur) Verweisadresse	im Adreßteil des Befehlswortes
	auf Speicheradresse des Operanden	

Notationen

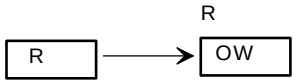
- A = <B> A ist der Inhalt des Speicherplatzes, dessen Adresse B lautet (kurz: A ist der Inhalt von B)
- B = >A< B ist Adresse von A
- A : = B + C A ergibt sich aus Summe von B und C
- A : = < B + C > A ergibt sich aus dem Inhalt des Speicherplatzes, dessen Adresse gleich B + C ist

Es ist	A	Adresse eines Hauptspeicherplatzes	BSA	Basisadresse
	AT	Adreßteil	D	Displacement
	R	Nummer eines Registers	I	Indexzahl
	B	Nummer eines Basisregisters	OW	Operandenwort
	X	Nummer eines Indexregisters		

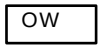
Adressierungsart und formale Notierung	Es steht im **Adreßteil** des Befehlswortes	Es steht im **Register** mit Adresse R, X oder B	Es steht im **Hauptspeicher**

direkte Hauptspeicher-adressierung
OW : = <A>

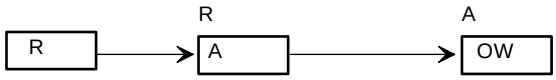
direkte Register-adressierung
OW : = <R>



Direktwert im Adreßteil
OW : = AT

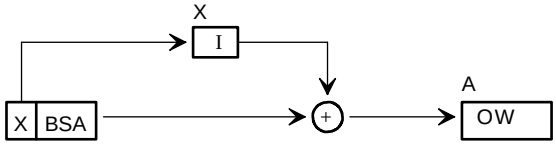


Hauptspeicheradressierung über Register
OW : = <A>
OW : = <<R>>

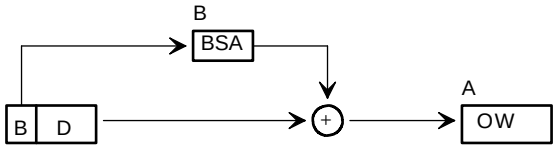


Hauptspeicheradressierung über Hauptspeicherplatz
OW : = <A_2>
OW : = <<A_1>>

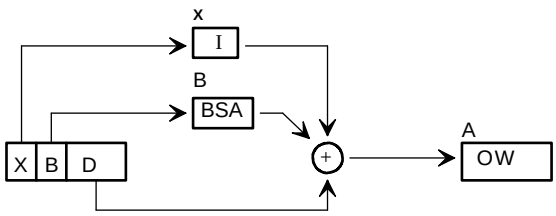
indizierte Hauptspeicheradressierung
OW : = < BSA + I >
OW : = < BSA + <X> >



relative Hauptspeicheradressierung
OW : = < BSA + D >
OW : = < <B> + D >



indizierte und relative Hauptspeicheradressierung
OW : = < I + BSA + D >
OW : = < <X> + <B> + D >



2.2 Komponenten eines Rechnerkerns

Abb. 1/2 enthält die folgenden Komponenten eines Rechnerkerns:

- **Rechenwerk**

  techn. Realisierung
  ---
  siehe *Skripten zur Informatik:* ***Digitale Strukturen zur Implementierung von Binärarithmetik***

  Arbeitsweise
  ---
  jeder Maschinenbefehl
  impliziert eine spezifische Folge elementarer Schritte

  Ein *Flag-Register* protokolliert den Status der Verarbeitung

- **Steuerwerk**

aus Einrichtungen zur
- # Generierung der Mikrooperationsfolge (Mikroprogrammsteuerwerk, Abb. 2/2)
- # Prozessorsteuerung (extern → intern),
- # Systemsteuerung (intern → extern),
- # Bussteueung (bidirektional)

und Detailfunktionen
- # Festlegung der Reihenfolge des Befehlslesen,
- # Dekodierung der gelesenen Befehle,
- # Adreßrechnung über Indexregister,
- # Verbindungsüberwachung zur Peripherie,
- (# Generierung von Steuersignalen für Co-Prozessoren).

Die Ablaufsteuerung des Rechnerkerns ist ein *monotoner Prozeß*, gekennzeichnet durch repitierende *Zyklen*, zusammengefaßt in einer *Zentralen Steuerschleife* für Verarbeitungs- und Sprungbefehle.

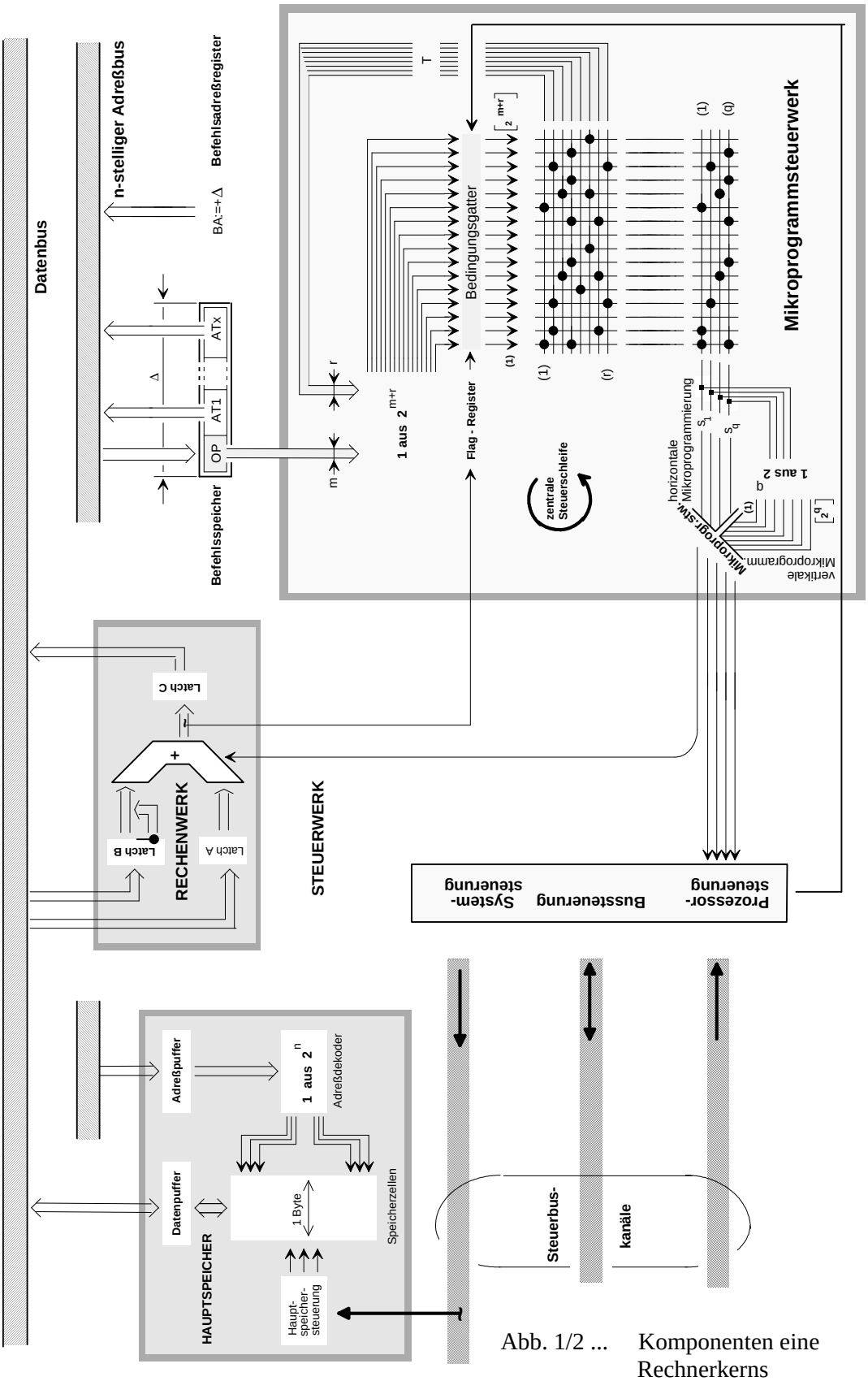


Abb. 1/2 ... Komponenten eine Rechnerkerns

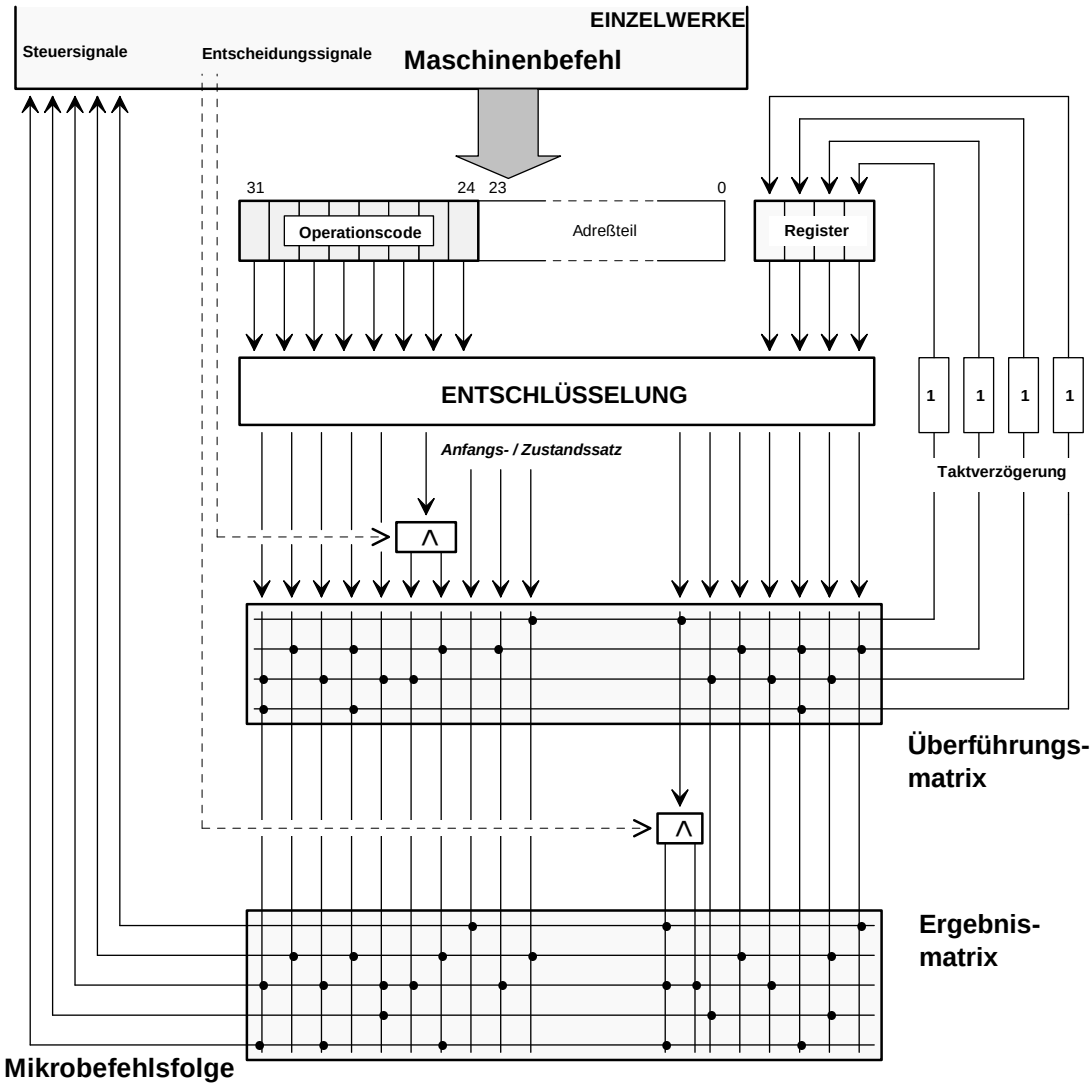


Abb. 2/2 Prinzipielle Struktur eines Mikroprogrammsteuerwerkes

Ausführungsformen eines Mikroprogrammsteuerwerkes

speicherprogrammiert

Alle *Mikrobefehle* sind in einem *Microcode-ROM* gespeichert.

Dekodierung eines *Maschinenbefehls* liefert die Anfangsadresse in einem Microcode-ROM.

Jedes Speicherwort im Microcode-ROM enthält:

- Mikrobefehl,
- Informationen zur Bildung der nachfolgende Zugriffsadresse auf den Microcode-ROM

Vorteil: kein Hardwareeingriff notwendig bei Änderung des Mikrobefehlssatzes

Nachteil: relativ geringe Generierungsgeschwindigkeit der Mikrobefehle bzgl.

festverdrahtet

Alle *Mikrobefehle* sind in einer *festverdrahteten Logik* gespeichert.

Vorteil: hohe Generierungsgeschwindigkeit der Mikrobefehle

Nachteil: Änderung des Mikrobefehlssatzes erfordert einen Neuentwurf der festverdrahteten Logik.

2.3 Grundstruktur und -funktion eine Mikrorechnersystems

Während der vorangegangene Abschnitt die Komponenten des Rechnerkerns behandelte, erörtert der vorliegende Abschnitt ausgewählte Funktionen eines zugrunde gelegten Mikrorechnersystems. Abb. 3/2 zeigt die Grundstruktur eines ZILOG-Systems.

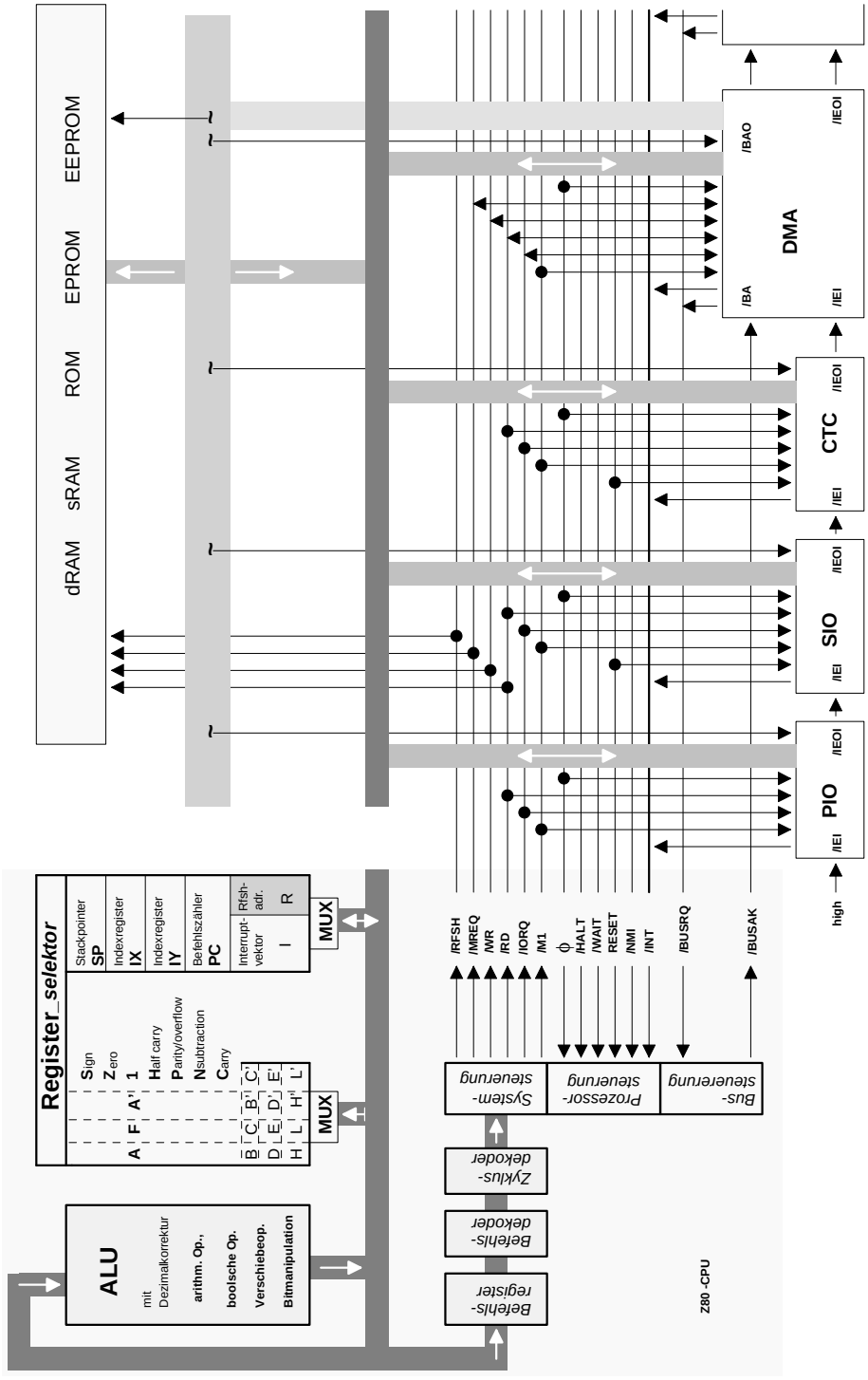


Abb. 3/2 Grundstruktur eines ZILOG-Systems

Kennzeichnende Eigenschaften eines Mikroprozessors sind:

- **Befehlsvorrat** ,
- **Unterbrechungsbehandlung** (*Interruptmodi, Anzahl Interrupteingänge, ...*),
- **Registersatz**

Speicherung von Befehlen	
Speicherung von Operanden zur Befehlsabarbeitung	i.d.R. alle Reg.
Speicherung von Resultaten nach der Befehlsabarbeitung	i.d.R. alle Reg.
Speicherung von Rückkehradressen bei Unterbrechungen	**Stack**
Bereitstellung von Adressen für Unterbrechnungsroutinen	Interrupt-Register
Protokollierung aktueller Zustände	**Flag-Register**

- *Flag-Register* -

Signum-Flag

0<A : **S**ign → low 0>A : **S**ign → high

Bsp.: $+12_{10} = \mathbf{0}\ 000\ 1100_2$ $-12_{DEZ} = \mathbf{1}\ 111\ 0011_2 + 1 = \mathbf{1}111\ 0100_2$

Zero-Flag

0<>A : **Z**ero → low 0=A : **Z**ero → high

Carry-Flag für arithmetische Operationen im Zahlenbereich > 1Byte

Bsp.:

0 000 0010	1 001 0001 $_2$	1. Operand =	657_{10}
0 000 0100	0 111 0000 $_2$	1. Operand =	1136_{10} +
0 000 0111	**1**0000 0001$_2$	Resultat =	1793_{10}

Parity/Overflow-Flag

logische/Verschiebeoperation bedingt für

ungerade Bitanzahl : $\mathbf{P}_{.../...}$ → low
gerade Bitanzahl : $\mathbf{P}_{.../...}$ → high

Resultat der arithmetischen Operation zweier vorzeichenbehafteter Operanden bedingt für:

$-128 \leq$ Resultat ≤ 127 : $\mathbf{P}_{.../...}$ → low
sonst : $\mathbf{P}_{.../...}$ → high

Half carry-Flag für BCD-Arithmetik

Übertrag $D_3 \rightarrow D_4$ im A : $\mathbf{H}_{alf\ carry}$ → high
kein Übertrag ... $\mathbf{H}_{alf\ carry}$ → low

N Addition/Subtraktion-Flag für BCD-Arithmetik

zuletzt ausgeführter Befehl war Addition od. Subtraktion : $\mathbf{N}_{Add/Sub}$ → high
sonst : $\mathbf{N}_{Add/Sub}$ → low

- *Stackpointer - SP* -

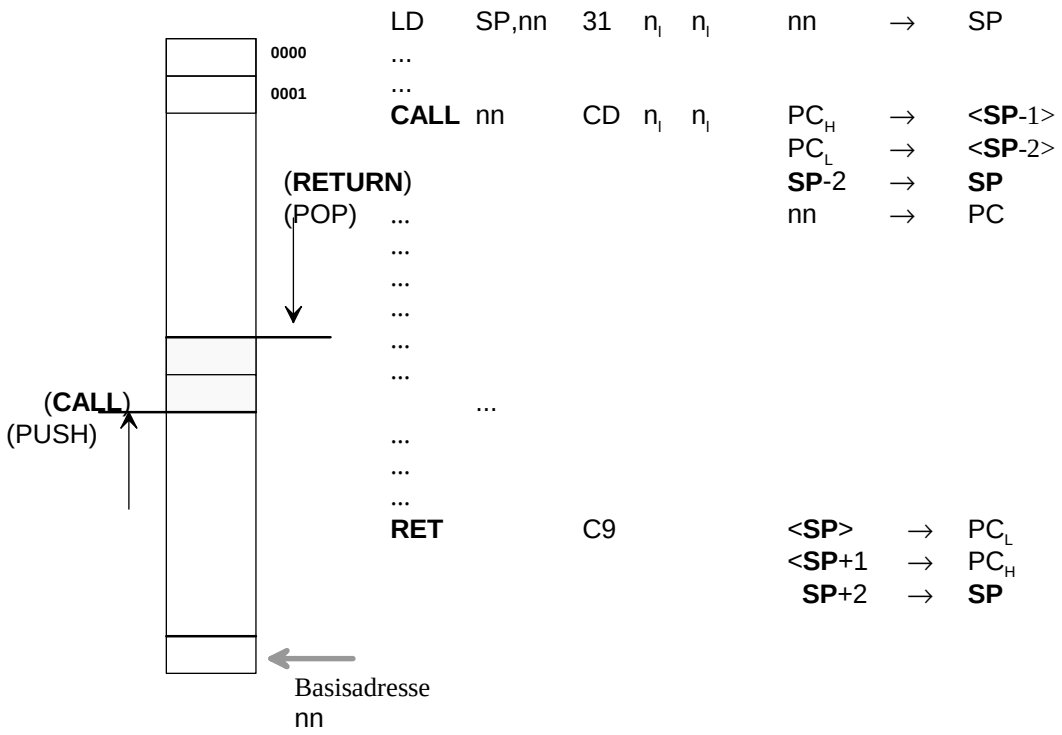
Der Stackpointer ist nach dem LIFO-Prinzip organisiert und enthält die Adresse eines

RAM-Kellers

- *zur Aufnahme von Rückkehradressen nach der Abarabeitung von Unterprogrammen, Interruptroutinen etc. ,*
- *zur Parameterübergabe ,*
- *zur Rettung von Registerinhalten .*

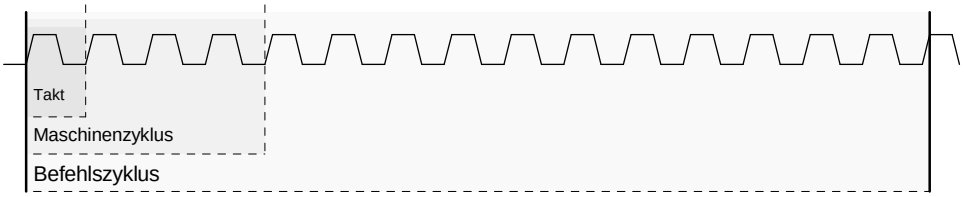
In der *Initialisierungsroutine* der CPU, ausgelöst z.B. durch das *RESET-Signal : <PC, I, R> → <0, 0, 0>*

erfolgt Laden der **Basisadresse** für den RAM-Keller in das **SP**-Register :



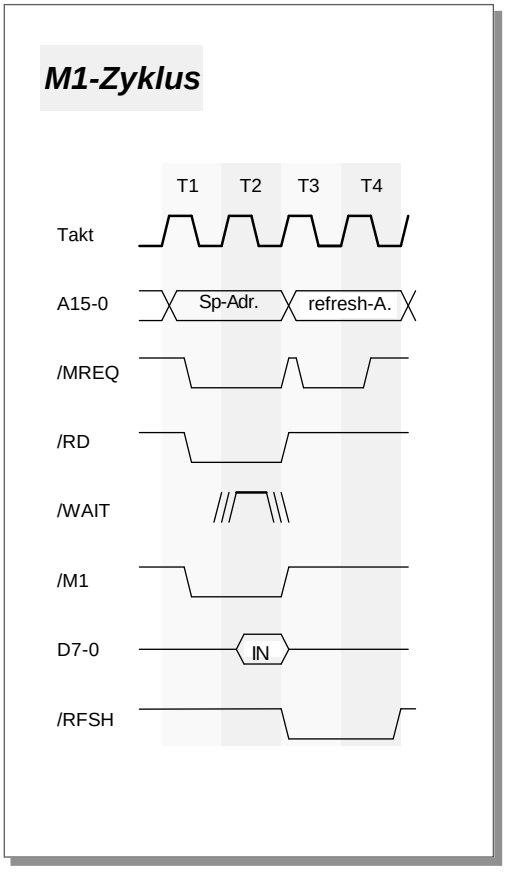
2.3.1 Maschinenzyklen des Mikroprozessors

Der Arbeitsablauf in einem Mikroprozessor ist in *Maschinenzyklen* organisiert.
Ein *Maschinenzyklus* besteht aus mehreren Takten,
Mehrere *Maschinenzyklen* bilden einen *Befehlszyklus.*

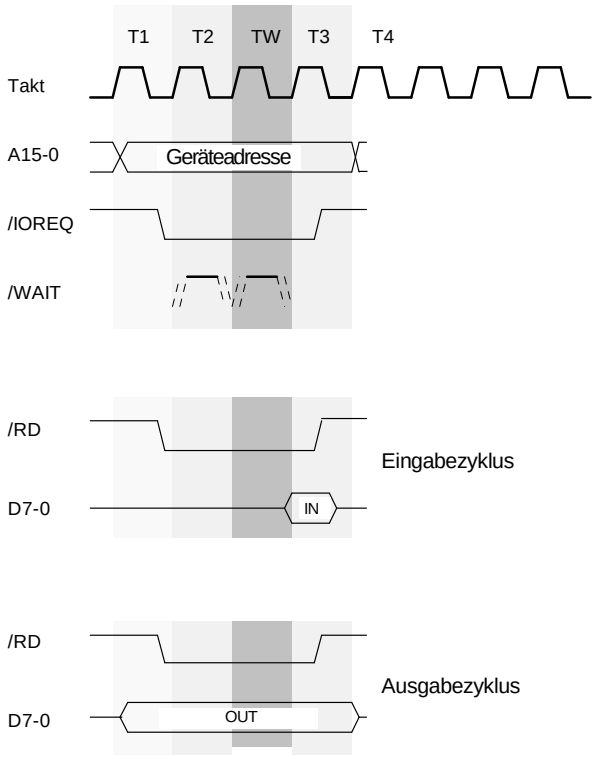


Maschinenzyklen können sein

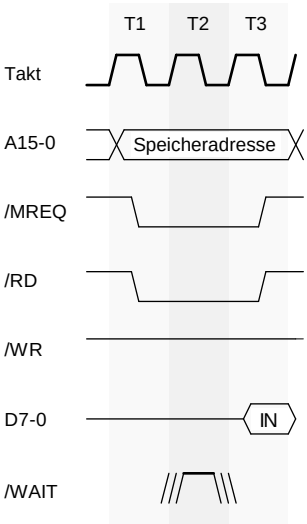
- **Befehlshole_Zyklus** (ist immer der 1. Zyklus: *M1-Zyklus*)
- **Speicherlese/ -schreib_Zyklus**
- **E/A_Zyklen**
- **Interrupt_Zyklen**



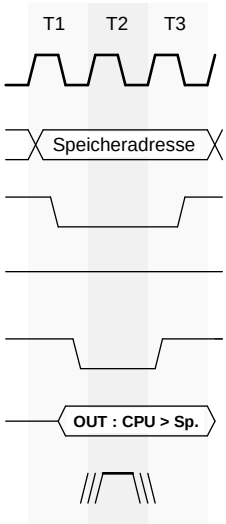
E/A-Zyklus

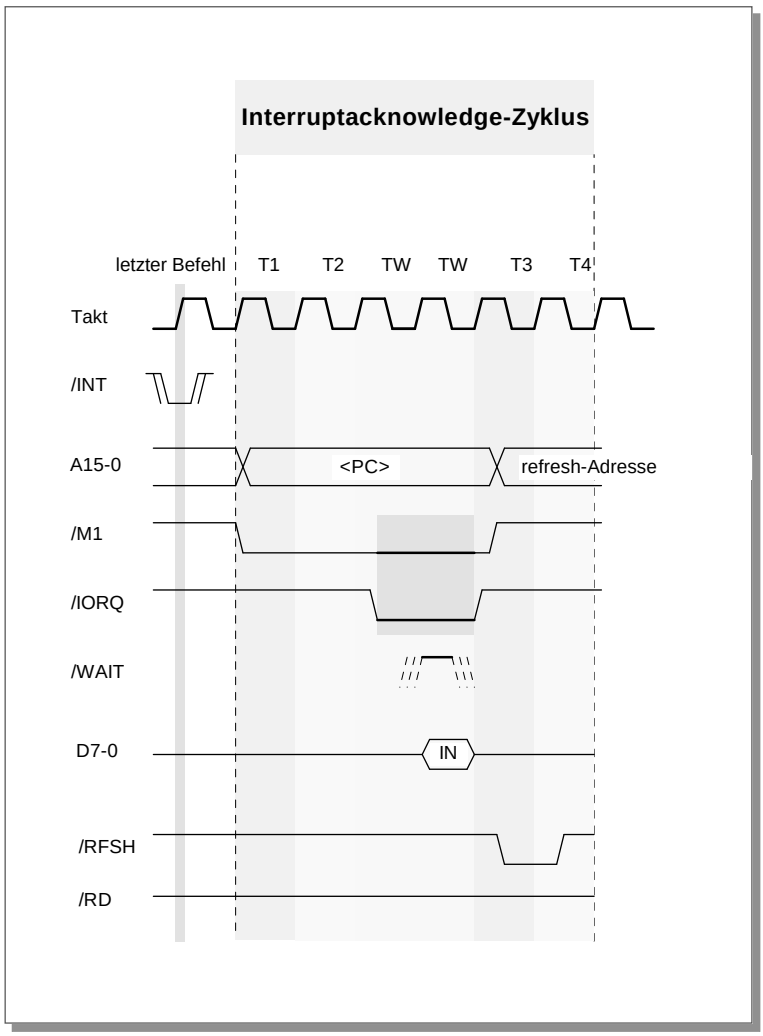


Speicher-Lese-Zyklus



Speicher-Schreib-Zyklus




Interruptacknowledge-Zyklus
letzter Befehl
T1
T2
TW
TW
T3
T4
Takt
/INT
A15-0
<PC>
refresh-Adresse
/M1
/IORQ
/WAIT
D7-0
IN
/RFSH
/RD

2.3.2 Interruptsteuerung eines Mikroprozessors

Prinzip des Interruptbetriebes

Das *Interruptanforderungssignal* wird zum Zeitpunkt des letztens Taktes im laufenden Befehlszyklus prozessorintern ausgewertet.

Der aktuelle Befehlsfluß im Mikroprozessor wird angehalten zwecks (vorrangiger) Abarbeitung einer extern anliegenden Unterbrechungsanforderung. Nachfolgend wird im Mikroprozessor eine *Unterbrechungsroutine* zur Behandlung der Unterbrechungsanforderung abgearbeitet.

Zu unterscheiden ist zwischen

***Anforderungssignal /NMI und /INT*:**

/NMI *nichtmaskierbarer Interrupt :*
Die Interruptannahme kann durch den Prozessor **nicht** verweigert werden und erfolgt prinzipiell, wenn
- der aktuelle Befehlszyklus beendet worden ist und
- keine Anforderung auf Busmasterschaft vorliegt (/BUSRQ=high).

/INT *maskierbarer Interrupt :*
Über die Annahme des Interrupt-Signals entscheidet ein speziell gesetztes *Statusbit* im Prozessor.
Das Statusbit *maskiert* den Interrupteingang.

Die Maskierung des Interrupteingangs erfolgt mittels der

Befehle	
DI	INT-FF rücksetzen
EI	INT-FF setzen

Die Annahme des Interrupt-Signals erfolgt,
- wenn das INT-FF gesetzt ist,
- wenn der aktuelle Befehlszyklus beendet worden ist,
- wenn keine Anforderung auf Busmasterschaft vorliegt (/BUSRQ=high) und
- wenn keine nichtmaskierte Interruptanforderung vorliegt (/NMI=high).

Es ist zu beachten:

- Das INT-FF ist nach einem RESET immer (!) rückgesetzt.
- Das INT-FF wird bei jeder (!) Interruptannahme rückgesetzt, um weitere Unterbrechnungen zu verhindern.
  Jedoch ist es möglich, durch den EI-Befehl auch in die Abarbeitung einer laufenden Unterbrechung eine nächste Unterbrechung "einzuschachteln".

Die **/NMI-Behandlung** erfolgt prinzipiell durch den *Unterprogrammsprung* auf
---
Adresse 0066_h.

Die **/INT-Behandlung** erfolgt in den drei *Modi*: IM0, IM1 und IM2

Befehl:	Codierung:				
...	...				
...	...				
DI	F3				
...	...				
EI	FB				
...	...				
...	...				
IM k	ED op mit	k:	0	1	2
		op	46	56	5E

---

adr_H := <I>, adr_L := <D0-7>

Nach einem RESET arbeitet die CPU im Modus 0.

2.3.2.1 Interruptmodi

- MODE 0 -

Das den Interrupt anfordernde Gerät transferiert über den Datenkanal D0-7 eine Kennung, die von der CPU als Befehl interpretiert wird.

Kodierung als **RESTART**-Befehl: I I # # # I I I mit ### als Anfangsadresse des abzuarbeitenden Unterprogramms

				Befehlscode
RST0 :	II	**000**	III	C7
RST8 :	II	**00I**	III	CF
RST10 :	II	**0I0**	II I	D7
RST18 :	II	**0II**	III	DF
RST20 :	II	**I00**	III	E7
RST28 :	II	**I0I**	III	EF
RST30 :	II	**II0**	III	F7
RST38 :	II	**III**	III	FF

INTA-Zyklus	Sp-Zyklus	Sp-Zyklus	M1-Zyklus
6 Takte	3 Takte	3 Takte	4 Takte
/M1 low /IORQ low	PC_H retten	PC_L retten	Lesen des 1. Op-Codes auf der Adrsse 0_H
	Retten der Rückkehradresse		oder 8_H oder ... oder 038_H
Befehl **RST** über D0-7 in die CPU laden			! 8 Byte Adreßabstand !

Kodierung als **CALL**-Befehl: I I 0 0 I I 0 I und der nachfolgenden Entgegennahme der **Startadresse für das Unterbrechungsprogramm**

INTA-Zyklus	E-Zyklus	E-Zyklus	Sp-Zyklus	Sp-Zyklus	M1-Zyklus
6 Takte	4 Takte	4 Takte	3 Takte	3 Takte	4 Takte
/M1 low /IORQ low	niederwertiges Adreßbyte	höherwertiges Adreßbyte	PC_H retten	PC_L retten	Lesen des 1. Op-Codes auf der Adresse a_L a_H
Befehl CALL über D0-7 in die CPU laden	**Startadresse des Unterbrech. programms auf D0-7**		Retten der Rückkehradresse		

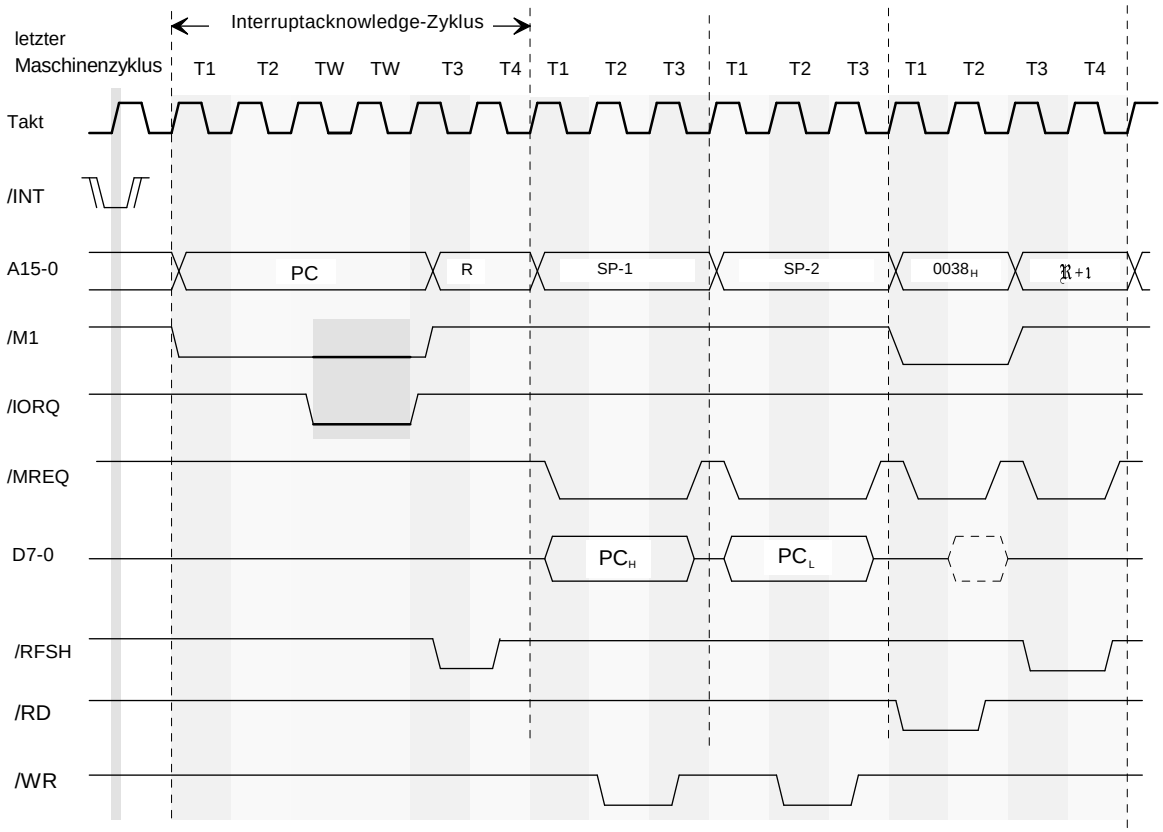
- *MODE 1* -

Dieser Modus wird durch den Interruptsteuerbefehl IM1 programmiert.

Nach Interrupt-Annahme (/INT low) reagiert die CPU intern mit der Implementierung des Befehls **RST38**, d.h. mit einem Sprung zur Adresse 38_h - der Startadresse des Unterbrechungsprogramms.

INTA-Zyklus	Sp-Zyklus	Sp-Zyklus	M1-Zyklus
6 Takte	3 Takte	3 Takte	4 Takte
/M1 low /IORQ low **RST38** wird intern eingefügt	PC_H retten	PC_L retten Retten der Rückkehradresse	Lesen des 1. Op-Codes auf der Adrsse 38_H

(demzufolge kein IV nötig)



- *MODE 2* -

Dieser Modus wird durch den Interruptsteuerbefehl IM2 programmiert.

Nach Interrupt-Annahme (/INT low) durch die CPU fordert die CPU vom Interruptrequester den Interruptvektor **IV** an (Abb. 4/2).

Für den *Mode 2* muß die CPU initialisiert werden mit der

- Bereitstellung einer Tabelle mit Startadressen der Unterbrechungsbehandlungsroutinen und dem
- Laden des **I**nterrupt-Registers (I) in der CPU mit dem high-wertigen Adreßteil der *Startadreßtabelle.*

(Dazu mußte zuvor der Interruptrequester mit dem Interruptvektor **IV** geladen worden sein.)

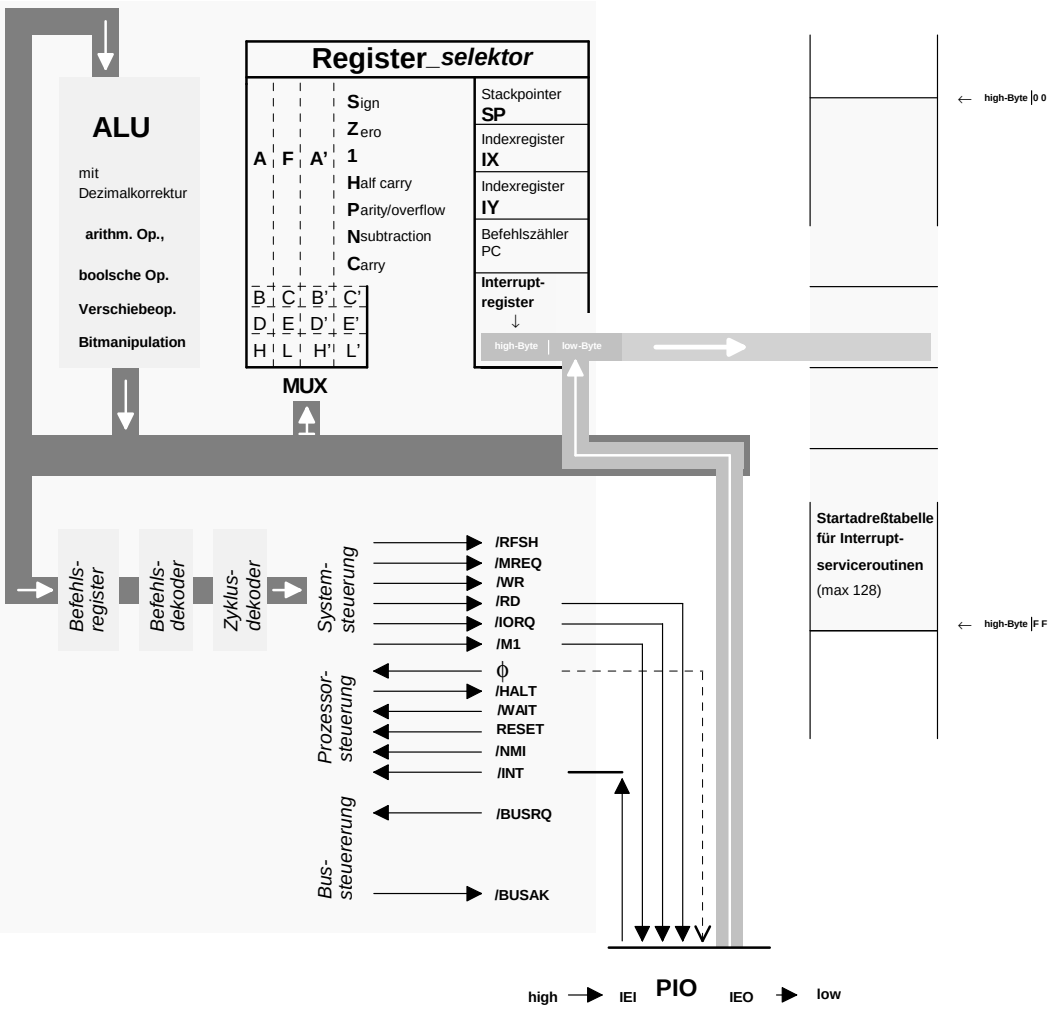
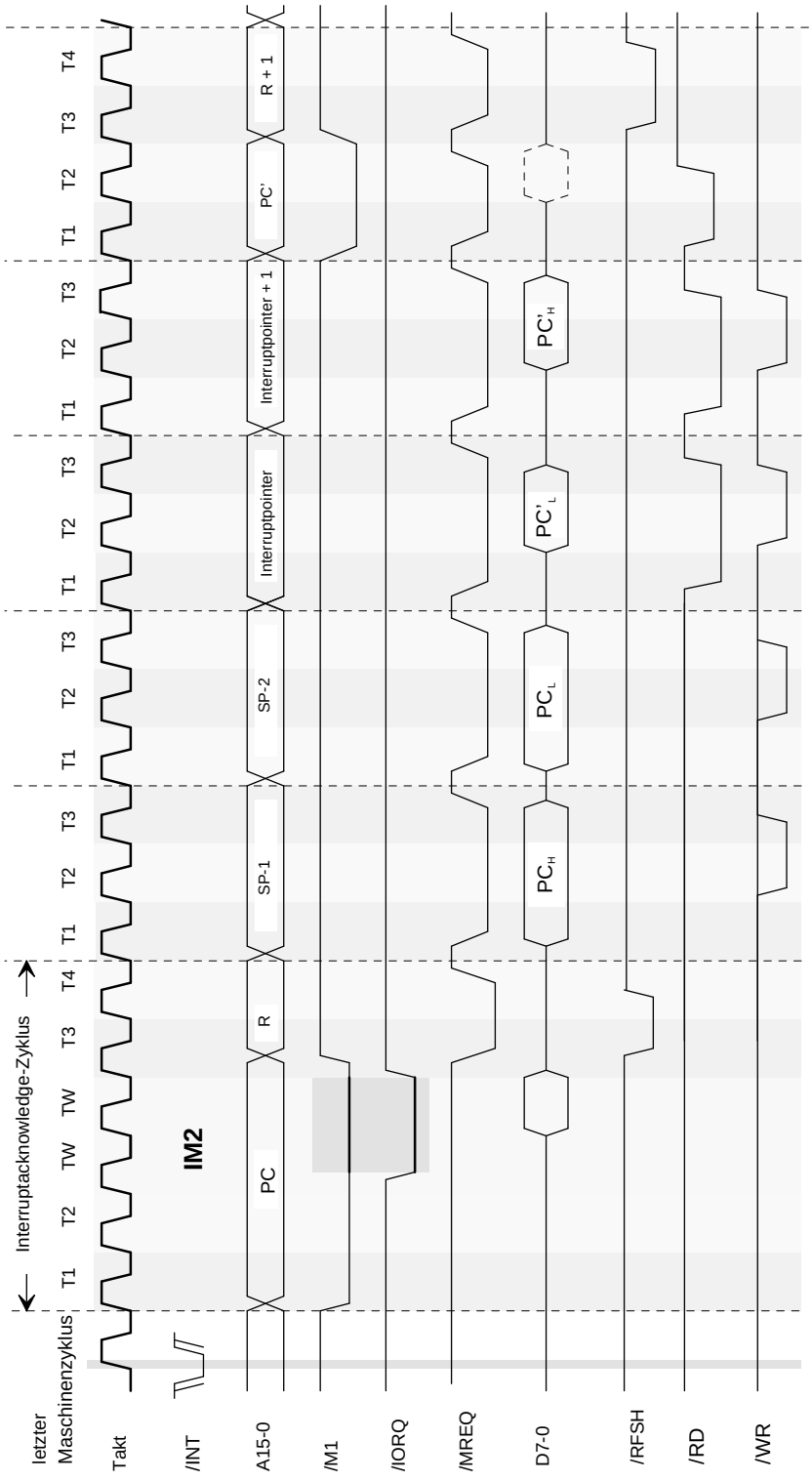


Abb. 4/2 Bildung der Speicherzugriffsadresse im MODE 2


letzter Maschinenzyklus
Interruptacknowledge-Zyklus
T1 T2 TW TW T3 T4 T1 T2 T3 T1 T2 T3 T1 T2 T3 T1 T2 T3 T4
Takt
/INT
IM2
A15-0
PC
R
SP-1
SP-2
Interruptpointer
Interruptpointer + 1
PC'
R + 1
/M1
/IORQ
/MREQ
D7-0
PCH
PCL
PC'L
PC'H
/RFSH
/RD
/WR

INTA-Zyklus	SP_**w**-Zykl.	Sp_**w**-Zykl.	Sp_**r**-Zykl.	Sp_**r**-Zykl.	M1-Zyklus
6 Takte	3 Takte	3 Takte	3 Takte	3 Takte	4 Takte
/M1 low /IORQ low	PC_H	PC_L	1. Tab.wert	2. Tab.wert	Lesen des 1. Op-Codes auf der im PC eingestellten Speicheradresse
*Interrupt-Requ. ladet **IV** über D0-7 in die CPUladen*	*Retten der Rückkehradresse*		*Startadresse der Unterbrechungsroutine aus dem Speicher in den PC laden*		

2.3.2.2 Funktion und Betriebsarten eines INTERRUPT-CONTROLLERS

Ein Interruptsignal veranlaßt das System zur Reaktion auf Ausnahmesituationen.

Einen Interrupt auslösende Ursachen können sein:

- *time out* während einer laufenden Datenübertragung,
  gerichtet auf den Bumaster und/oder _-slave;
- Spannungsausfall in der Stromversorgungseinheit,
  gerichtet auf alle Prozessormoduln;
- peripheres Gerät bei E/A-Forderung,
  gerichtet auf den Busmaster zwecks Freigabe des Busses durch die Rücknahme des Busanforderungssignals;
- Datenempfänger bei Feststellung von Übertragungsfehlern,
  gerichtet auf den Datensender;
- Sektoransteuerung einer Diskette oder einer HD.

Ein ***INTERRUPT-CONTROLLER*** dient zur

- Priorisierung von Unterbrechungsanforderungen an die CPU,
- Bereitstellung der entsprechenden Interruptvektoren.

Die Reaktion des Mikroprozessors auf die Annahme eines Interruptsignals besteht in der Abarbeitung einer Interruptroutine, spezifiziert durch den *Interruptvektor*.

Der INTERRUPT-CONTROLLER i...59 ist ein spezieller Schaltkreis zur
---

- Priorisierung zeitgleich erhobener Interruptanforderungen;
- Bereitsstellung einer *Interruptvektors* (Abb. 5/2).

Über die Interruptleitungen /INT7-0 beim MULTIBUS wird das Interruptsignal vom Slave zum Master übertragen. Der Master enthält zur Spezifizierung der ankommenden Interruptsignale aus dem Slave in der Regel einen solchen INTERRUPT-CONTROLLER.

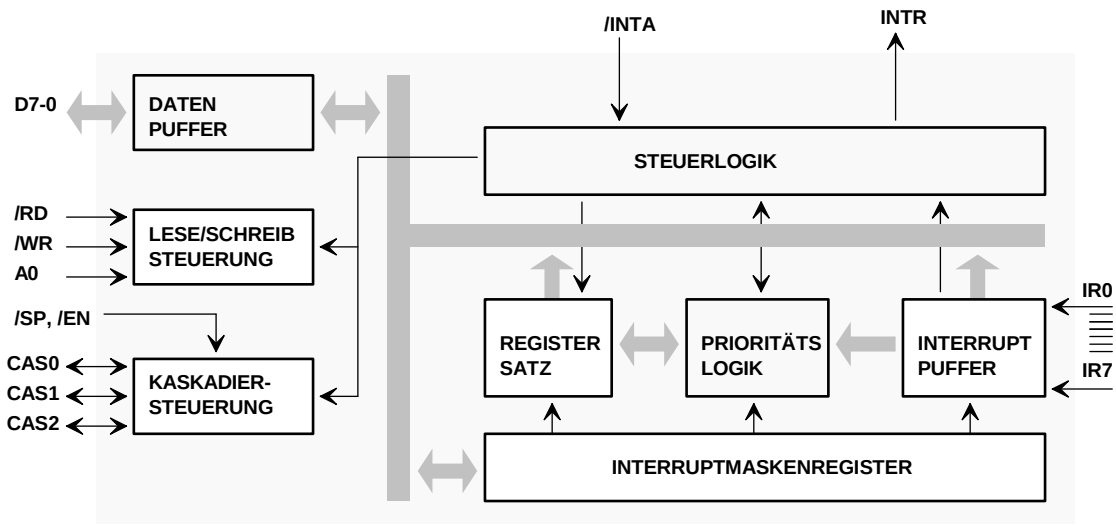


Abb. 5/2 INTERRUPT-CONTROLLER i...59

- *Leitungsbasierte Interruptsignalisierung* -

(auch bez. als **non bus vectored interrupt**)

Die 8 Interruptleitungen des MULTIBUS (das zugrunde gelegte Bussystem) werden auf die Interrupteingänge IR7-0 des Interrupt-Controllers geschaltet. Jeder der 8 Interruptleitungen ist eine spezielle Interruptroutine zugeordnet (Abb. 6/2).

De Interrupt-Controllers hat die Aufgabe, die auf den Interruptleitungen anliegenden Forderungen gegeneinander zu priorisieren und den für die priorisierte Anforderung entsprechenden Interruptvektor an den Mikroprozessor zu übertragen. Da in diesem Fall kein Vektor über das Bussystem transportiert wird, spricht man vom *non bus vectored interrupt*.

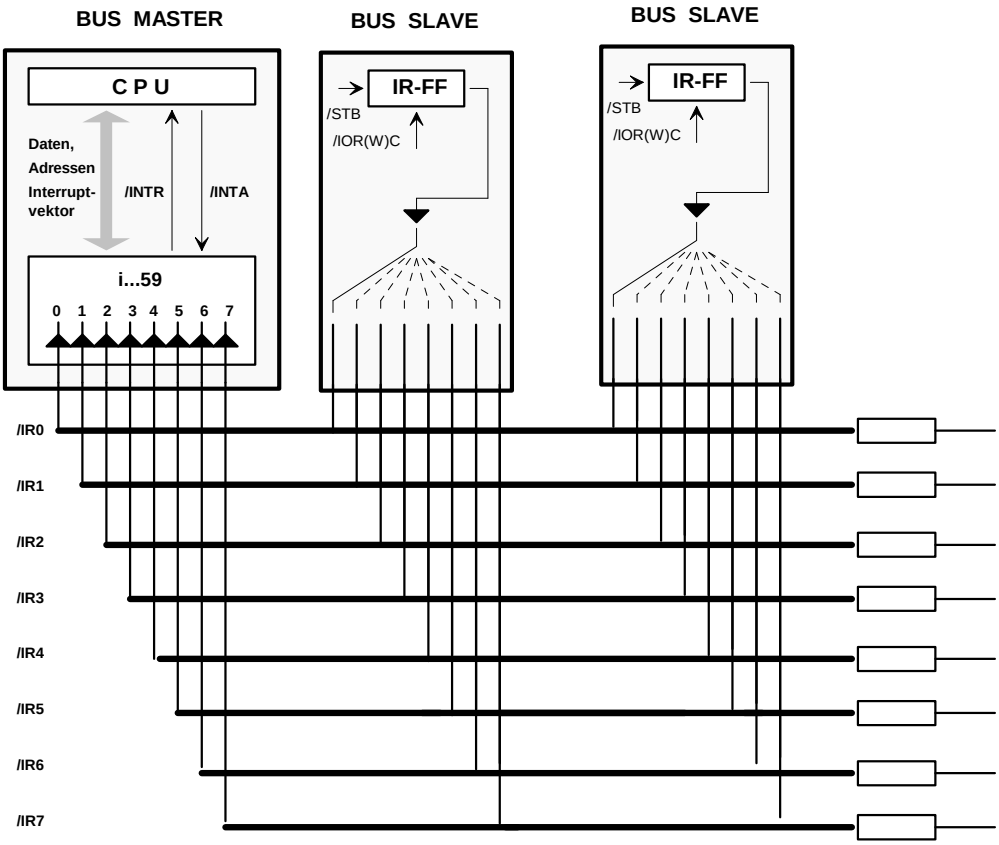


Abb. 6/2 Leitungsbasierte Interruptsignalisierung

- *Vektorbasierte Interruptsignalisierung* -

(auch bez. als **bus vectored interrupt**)

In diesem Fall enthält bereits jeder (!) Slave einen Interrupt-Controller zur Priorisierung zwischen mehreren lokalen (!) Interruptanforderungen, die zeitgleich erhobenen werden. Nach erfolgter Priorisierung aktiviert der Interrupt-Controller auf der Slave-Seite ein globales (!) Interruptanfordrungssignal /INTR, das über einen Interruptkanal des MULTIBUS auf der Masterseite einem zugeordneten Interrupteingang /IR* des Interrupt-Controllers aufgeschaltet wird (Abb. 7/2).

- Der Interrupt-Controller des Masters priorisiert die globale Interruptanforderung , d.h., es wird der Slave mit dem höchstpriorisierten globalen Interruptanforderungssignal selektiert.

- Mittels Interruptvektor teilt der priorisierte Slave über den Datenkanal des MULTIBUS dem Master mit, welche lokale Interruptanforderung slaveseitig priorisiert worden ist.

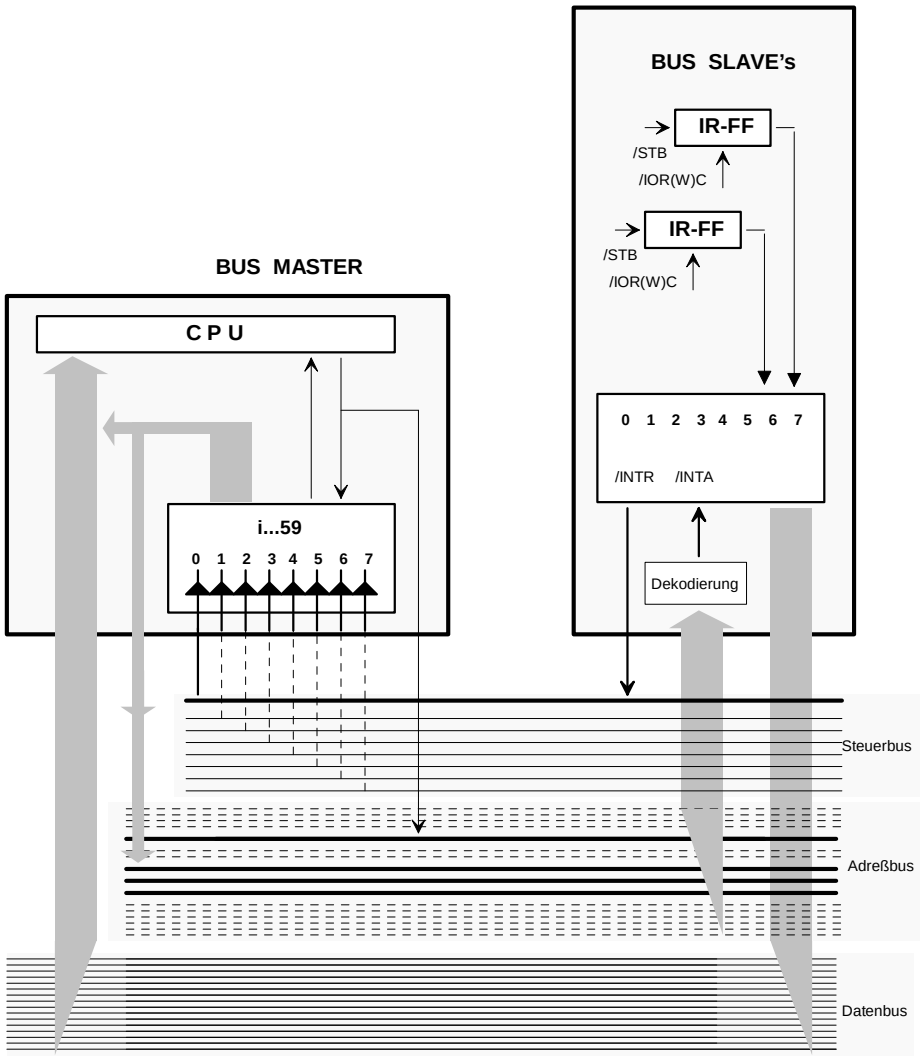


Abb. 7/2 Vektorbasierte Interruptsignalisierung

2.4 Ein-/Ausgabesteuerung eines Mikrorechnersystems

Behandelt wir in diesem Abschnitt der bitparallele Transfer eines Datenstroms in Eingabe- und in Ausgaberichtung über ein E/A-Interface. Schwerpunktmäßig wird dabei auf die Unterstützung durch die CPU eingegangen.

2.4.1 Pallel Input Output (PIO) - Interface

Das PIO-Interface hat die Aufgabe, den parallelen Datentransfer zwischen dem Mikrorechner und der Peripherie zu organisieren.

Peripherer Geräte mit paralleler Datenein- und -ausgabe:

- Drucker,
- Tastaturen,
- Bildschirme,
- Terminals,
- digitale und analoge Prozeß-E/A-Baugruppen

Zu unterscheiden ist zwischen

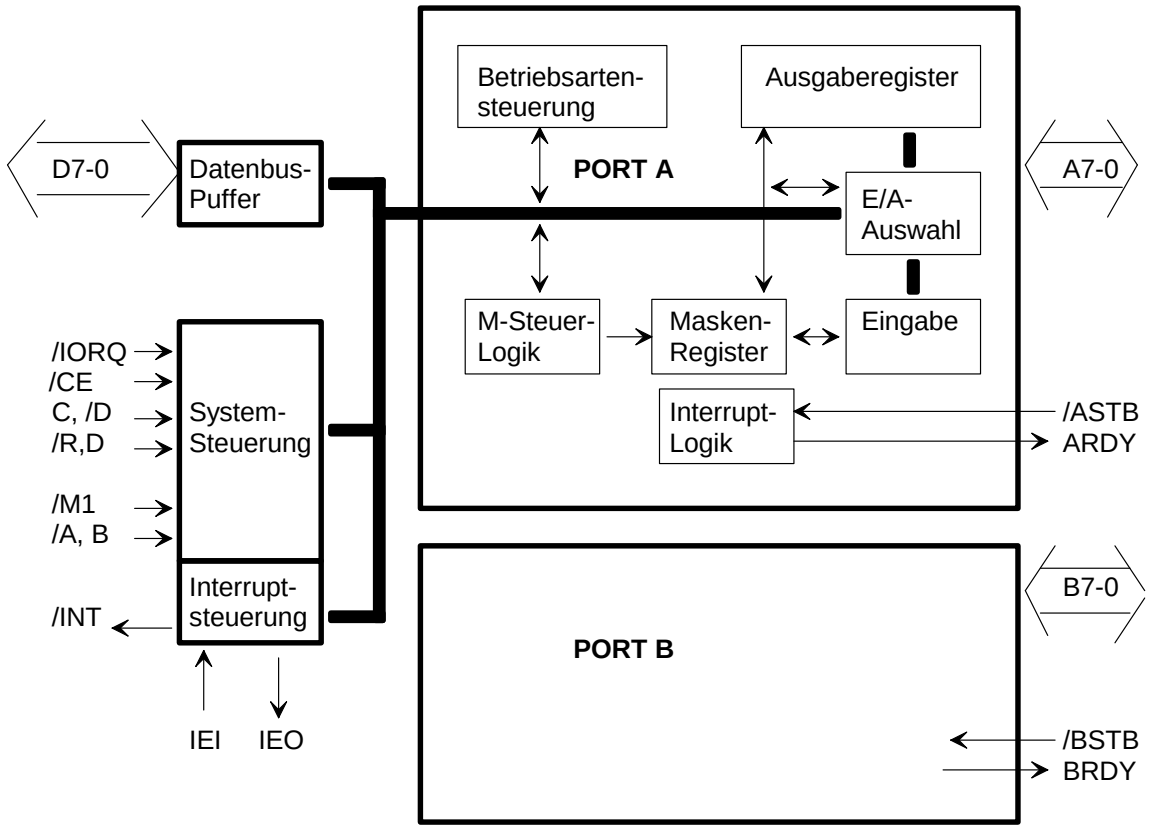
- Signalen zur Steuerung der Peripherie (/STB und RDY) und den notwendigen
- Signalen zur Steuerung des Systems (Systemsteuersignale).

Der PIO-Schaltkreis generiert lediglich Peripheriesteuersignale und prinzipiell keine Systemsteuersignale. Demzufolge können dessen Aktivitäten nur durch die CPU gesteuert werden. Dazu ist es notwendig, daß beide Schaltkreise miteinander Steuersignalen austauschen. Zwischen

- CPU und PIO erfolgt eine interruptgesteuerte Zustandssignalisierung.

In der Regel bedient ein Prozessor mehrere periphere Geräte. Demzufolge empfängt ein Prozessor auch mehrere Interruptsignale, kann allerdings pro Zeiteinheit nur auf ein einziges Interruptsignal reagieren. Deshalb ist es notwendig, alle Interruptsignale gegeneinander zu priorisieren. Unabhängig davon, welches Interruptsignal von jeweils höchster Priorität ist, muß der aktuell priorisierte Interruptrequester (im vorliegenden Fall der PIO-Schaltkreis) sowohl den Beginn als auch die Beendigung seiner CPU-Inanspruchnahme der Systemumgebung signalisieren.

Der PIO-Schaltkreis funktioniert in der Weise, daß er über das Pin IEI (=high) exklusiv berechtigt wird, den Interruptvektor /INT an die CPU auszusenden und über das Pin IEO (=low) die Inanspruchnahme dieser Möglichkeit der Systemumgebung signalisiert.



Der PIO-Schaltkreis[1] arbeitet in den folgenden Betriebsarten:

mode0 : byteweise Ausgabe

mode1 : byteweise Eingabe

mode2 : bidirektionale Ein-/Ausgabe ausschließlich über **Kanal A**

Kanal B muß in mode3 gestzt werden

A(B)RDY und /A(B)STB Ausgabe- bzw. Eingabe-Steuersignale

mode3 : Einzelbitsteuerung

- zur bit-parallelen Ausgabe von Steuersignalen
- zur /INT-Generierung bei Übereinstimmung zwischen Eingabewort und Maskenregister

(1)	HIGH/AND	letztes auf	high	gehende Eingabebit löst /INT aus
(2)	LOW/AND	.....	low	.....
(3)	HIGH/OR	jedes auf	high	.....
(4)	LOW/OR	.....	low	.....

[1] Die dargestellte Grundstruktur ist prinzipiell, dementsprechend sowohl für *Z80-PIO* als auch für den programmierbaren Interface-Schaltkrei s *i8255* charakteristisch.

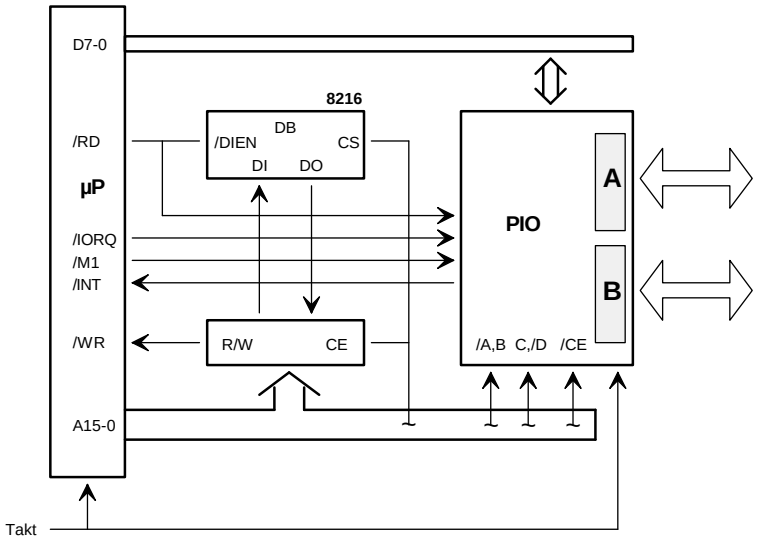


Abb. 8/2 Mikrorechnerminimalkonfiguration,

bestehend aus Prozessor zur Befehlsausführung,
D/A-Bus zum D/A-Transfer,
Steuersignalleitungen zum Lesen, Schreiben, Interruptsignalisierung,
Speichereinheit für Daten und Befehle,
PIO-Schaltkreis für den E/A-Transfer

Adreßbus :	A0	→	PIO :	/A,B	zur Portauswahl
	A1	→	PIO :	C,/D	zur Steuer-/Datenwortübernahme
	A2-7	→	PIO :	/CE	zur Schaltkreisselektion

/M1 → PIO : zur PIO-Synchronisation
bei exklusiver Nutzung zu interpretieren als
RESET, d.h. : Maskenregister gelöscht
IntENABLE FF gelöscht
Port A & B im *tri-state*
Handshakesignale sind inaktiv

/M1 & /IORQ	→	CPU :	Interruptvektoranforderung
/RD & /IORQ	→	CPU :	Datenwortübertragung
/IORQ	→	PIO :	Steuer-/Datenwortübertragung

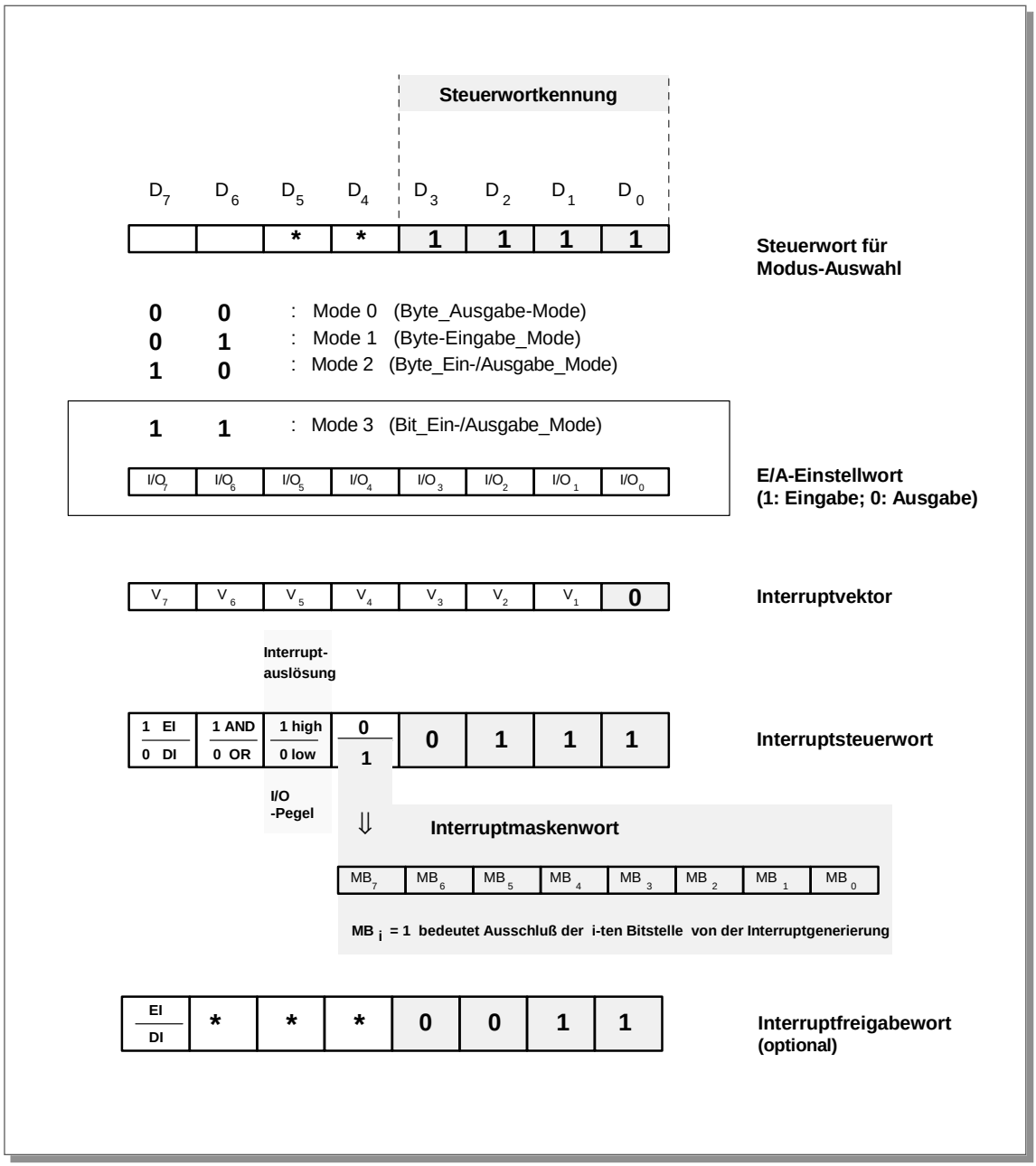
- Steuerwortfolge zur Initialisierung des PIO-Schaltkreises -

---

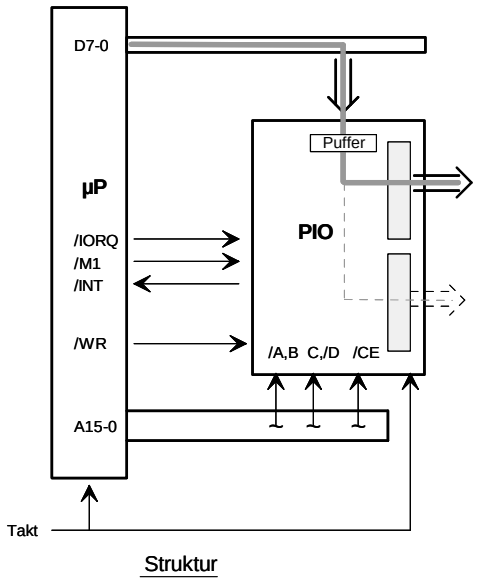
Die Initialisierung erfolgt durch einen Datenstrom von der CPU zum PIO-Schaltkreis

- einzeln für jedes Port, ausgewählt über die Datenleitung A_0,
- vom PIO-Schaltkreis interpretiert als *Steuerwortsequenz* infolge A1=high
- in einer einzuhaltenden

Reihenfolge der Initialisierung

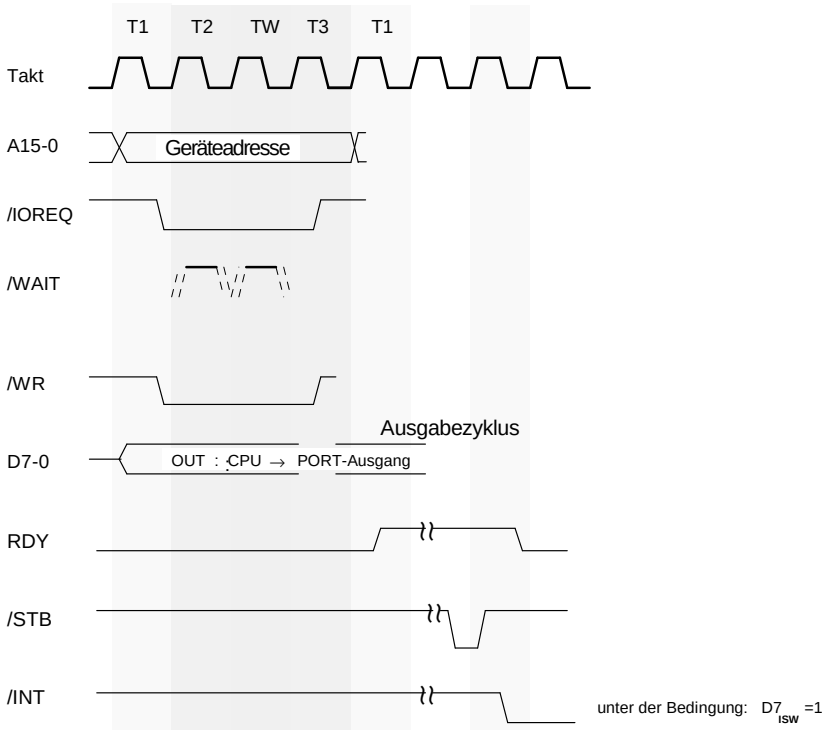


mode 0 : ***byteweise Ausgabe***
(aBf *Byteausgabemode)*



Struktur

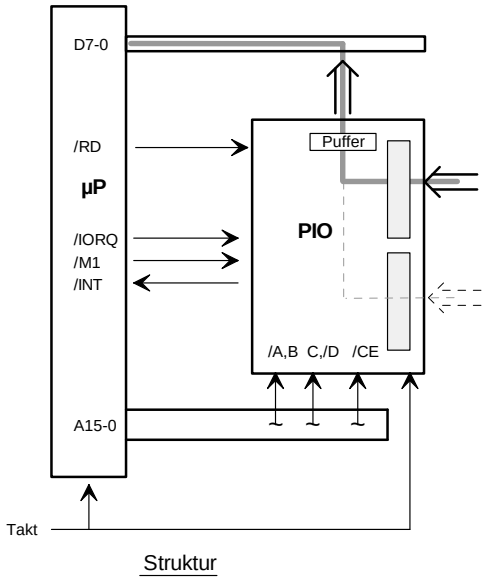
Zeitverlauf



Sowohl über Port A als auch über Port B kann ein Datenbyte ausgegeben werden. Aus der CPU wird Datenbyte für Datenbyte in den Pufferspeicher des PIO-Schaltkreises eingeschrieben. Die Auswahl der Ausgabeports für das eingeschriebene Datenbyte legt ein entsprechendes Ressourcensteuerwort fest.

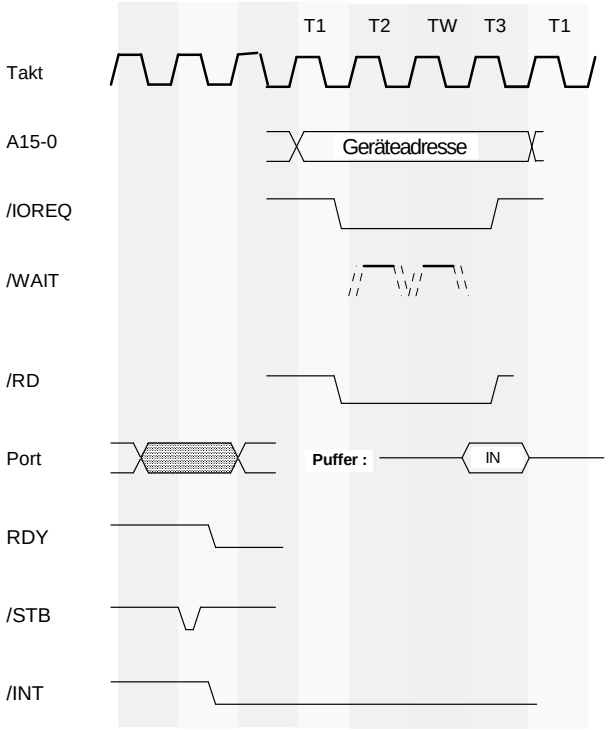
Über ein Interrupt-Signal kann der CPU die erfolgte Ausgabe des Datenbytes aus dem Ausgabeport in die Peripherie signalisiert werden, z.B. zwecks Übertragung eines nächsten Datenbytes in den Datenpuffer des PIO-Schaltkreises.

mode1 : ***byteweise Eingabe***
(aBf *Byteeingabemode)*



Struktur

Zeitverlauf



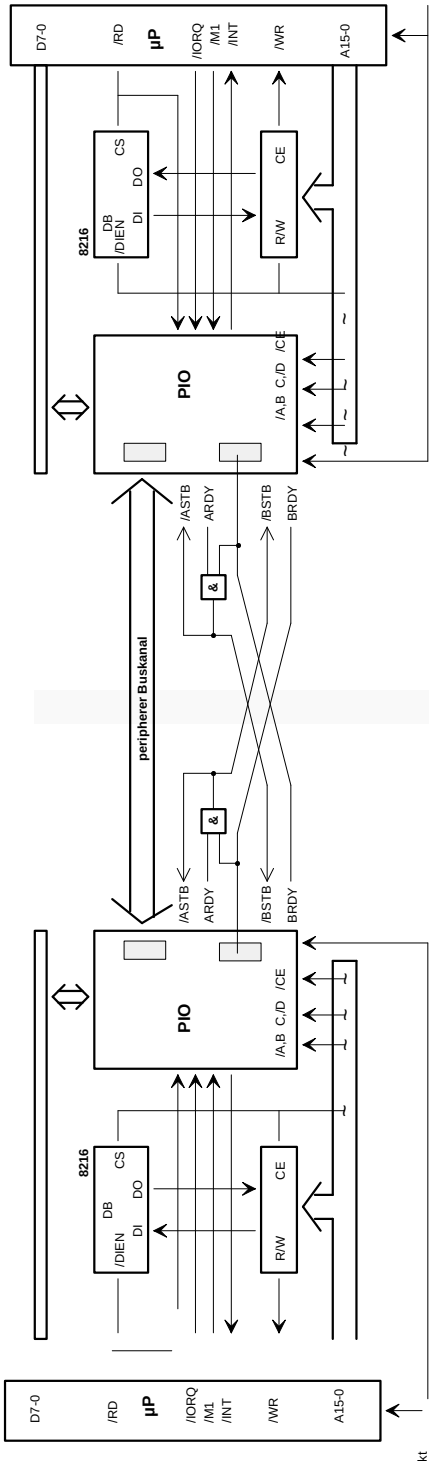
Im Byteeingabmode nimmt der eingabebereite (RDY=high) PIO-Schaltkreis aus der Peripherie ein Datenbyte entgegen (/STB : low → high) und signalisiert der CPU (/INT : high → low) das Vorhandensein eines Datenbytes zum Einlesen in die CPU.

mode2 : ***bidirektionale Ein-/Ausgabe***
über Kanal **A** (Kanal **B** ist ohne Bedeutung bzw. ohne definierte Bitkombination)

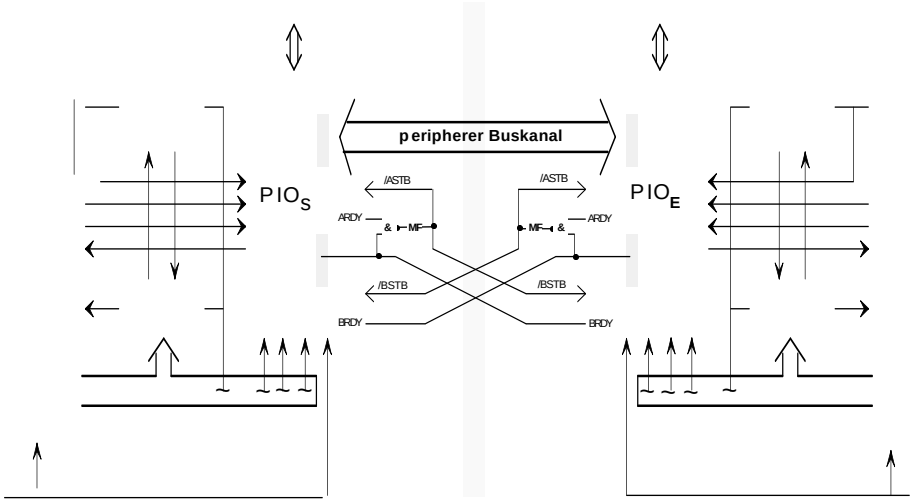
Eingabe-Steuersignale: ARDY, /ASTB
Ausgabe-Steuersignale: BRDY, /BSTB

Beispielanwendung

Kopplung zweier Mikrorechner
über zwei PIO-Schaltkreise
im bidirektionalen Datentransfer



***Mikrorechner S**ender* ***Mikrorechner E**mpfänger*



(1) Mikrorechner **S** transferiert im **<u>E/A-Zyklus</u>** ein Datenbyte in das Port $\mathbf{A}_S$

/IORQ ↑ impl. $ARDY_S$↑ ,

demzufolge $ARDY_S$ =high .

Mikrorechner **E** ist infolge $BRDY_E$ =high bereit zur Datenbyteübernahme.

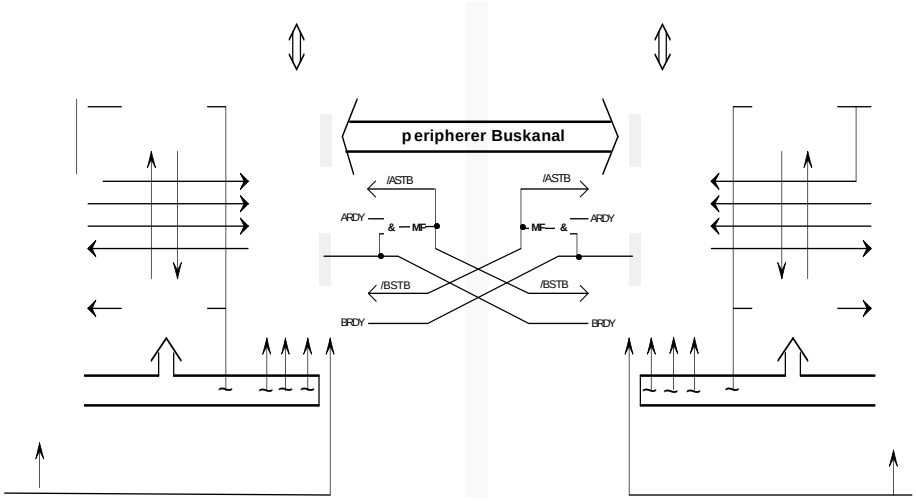
(2) [$BRDY_E$ =high] zeitgleich mit [$ARDY_S$↑] impl. *sowohl* $/ASTB_S$↓ *als auch* $/BSTB_E$ ↓

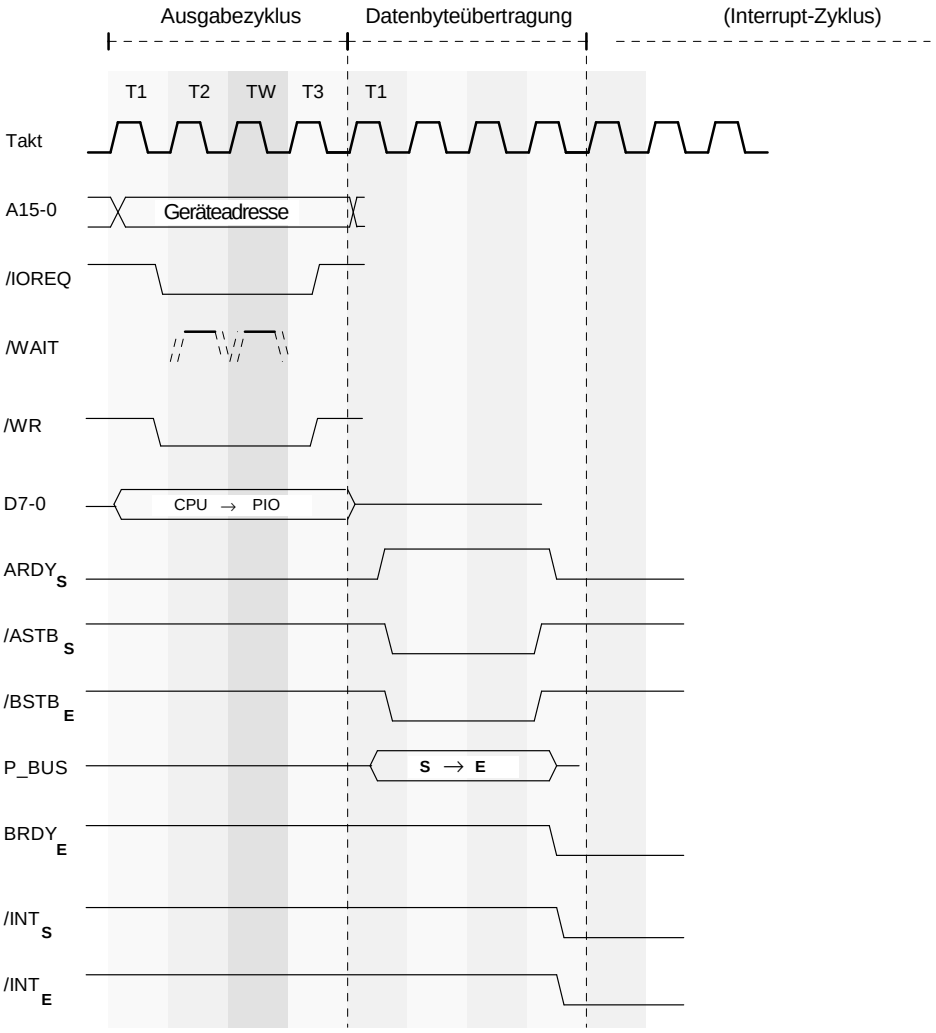
Damit erscheint sendeseitig auf dem peripheren Bus das Datenbyte. Ein Mono-Flop kann für eine einstellbare Zeit die Steuersignale $/ASTB_S$ =low und $/BSTB_E$ =low und damit auch das Datenbyte auf dem peripheren Bus belassen.

(3) Der Abfall des Mono-Flop's hat taktsychron $/ASTB_S$↑ und $/BSTB_E$↑ zur Folge:

- $/ASTB_S$↑ impl. $ARDY_S$ ↓ : Port A_S enthält kein weiteres Ausgabebyte
- $/ASTB_S$↑ impl. $/INT_S$ ↓ : PIO_S unterbricht CPU zur Abarbeitung einer Interruptroutine, z.B. erneutes Laden von Port A_S
- $/BSTB_E$↑ impl. $BRDY_E$ ↓ : Port A_E ist nicht mehr zur Übernahme eines weiteren Datenbytes bereit
- $/BSTB_E$↑ impl. $/INT_E$ ↓ : PIO_S unterbricht CPU zur Abarbeitung einer Interruptroutine zwecks Übernahme des Inhalts von Port A_E in die CPU.

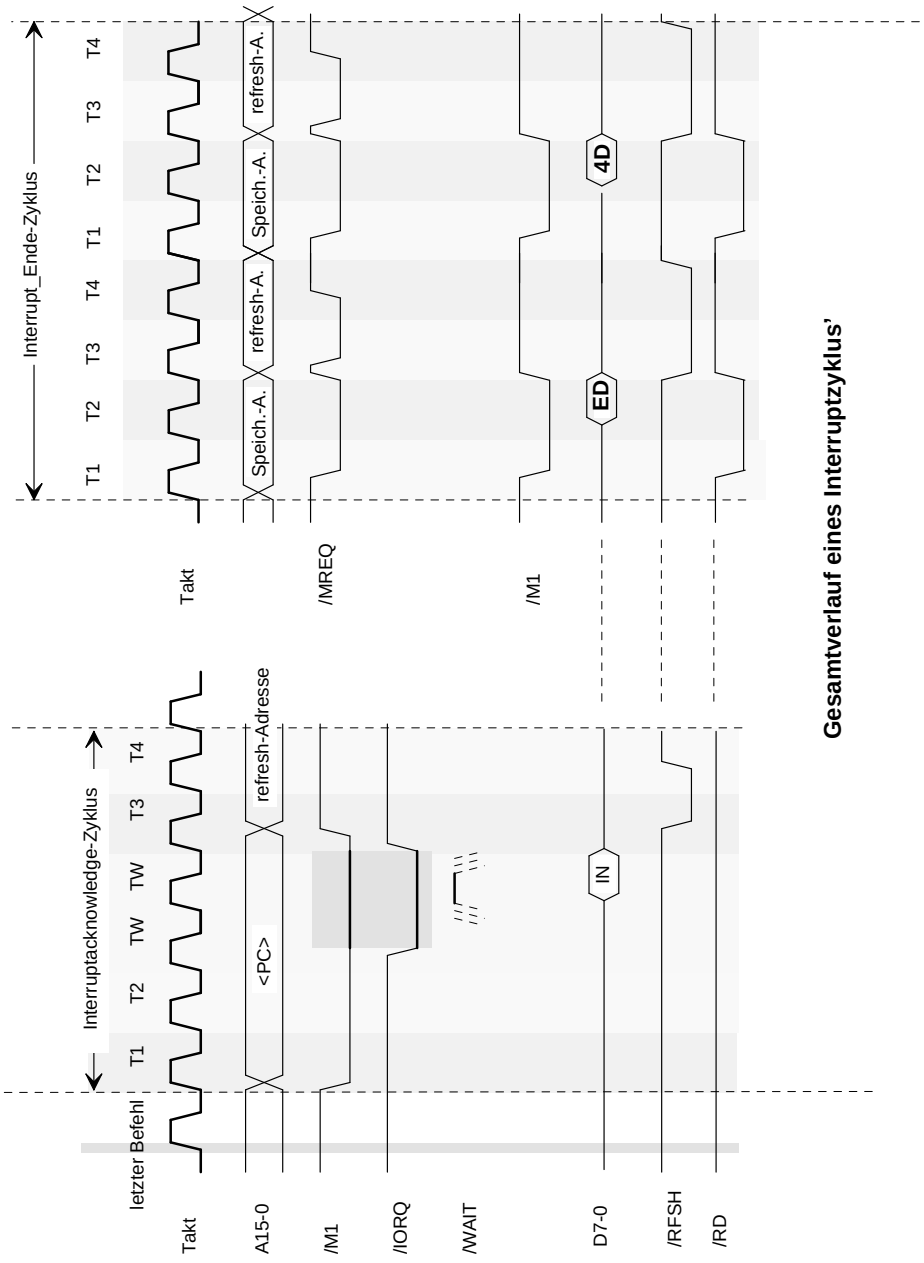
Die Datenbyte-Übertragung vom Mikrorechner **S** in den Mikrorechner **E** ist beendet.


peripherer Buskanal
/ASTB
/ASTB
ARDY
& MF
MF &
ARDY
/BSTB
/BSTB
BRDY
BRDY


Ausgabezyklus
Datenbyteübertragung
(Interrupt-Zyklus)
T1
T2
TW
T3
T1
Takt
A15-0
Geräteadresse
/IOREQ
/WAIT
/WR
D7-0
CPU → PIO
ARDY S
/ASTB S
/BSTB E
P_BUS
S → E
BRDY E
/INT S
/INT E

Die Beendigung eines Interrupt-Zyklus' wird durch den in der Folge letzten Befehl ***RE****turn from* ***I****nterrupt* **(RETI)**-Befehl in der Interruptroutine angewiesen. Der RETI-Befehl bewirkt in der CPU das Rückschreiben der (vor Beginn der Interruptroutine) gekellerten Programmadresse. Hinweis: Vor der Ausführung des RETI-Befehls ist der Prozessor für die weitere Interruptannahme durch die Ausführung des Befehl **EI** zu befähigen. Dieser wird aber erst nach der Ausführung des nächsten Befehls wirksam.

Der Interruptrequester inspiziert als E/A-Ressource permanent den Datenbus. Im Falle des Lesens eines RETI-Befehls signalisiert der Interruptrequester das Ausgangssignal **IEO=high** und gibt damit auch nachfolgende Interruptrequester wieder frei.



mode3 : ***Einzelbitsteuerung***

zur bitparallelen Ein- und Ausgabe von Steuersignalen
und speziell zur /INT-Generierung

(Der PIO-mode 3 wird bei der Beschreibung der DMA-Bausteine exemplarisch erläutert.)

2.4.2 Direct_Memory_Access (DMA) - Interface

DMA-Schaltkreise dienen dem beschleunigten Datenverkehr sowohl zwischen E/A-Ressourcen und Speichern als auch zwischen denen selbst.

Charakteristika von DMA-Schaltkreisen:

- **DMA-Schaltkreis und CPU generieren Systemsteuersignale.**

  Während eines Datentransfers mittels DMA-Schaltkreis befindet sich der Prozessor im ***tri-state***[2] und der DMA-Schaltkreis besitzt die ***Systemmasterschaft***.

  Der Prozessor ist bei *Systemmasterschaft* des DMA passiv und kann demzufolge keine Systemsteuersignale generieren; diese werden vom DMA-Schaltkreis generiert.

  Generiert hingegen die CPU Systemsteuersignale, besitzt sie die *Systemmasterschaft* und der DMA-Schaltkreis ist *passiv*.**.**

  Fazit : Sowohl CPU als auch DMA-Schaltkreis generieren Systemsteuersignale, aber nur alternativ zueinander.

  Hinweis: Der PIO-Schaltkreis generiert keine Systemsteuersignale und arbeitet demzufolge nur unter CPU- oder DMA-Steuerung.

- **Der DMA-Schaltkreis muß der CPU die Forderung auf Systemmasterschaft signalisieren.**

  Die CPU gewährt dem DMA-Schaltkreis zwar bedingungslos, aber nicht zu allen Zeitpunkten die *Systemmasterschaft*.

  Die Überlassung der Systemmasterschaft durch die CPU an einen der DMA-Schaltkreise erfolgt ausschließlich auf Anforderung eines DMA-Schaltkreises im letzten Takt des laufenden Maschinenzyklus' der CPU mit /BUSRQ.

  Hinweis: Die Unterbrechnung der CPU durch ein Interrupt-Signal ist nur am Ende eines laufenden *Befehlszyklus'* möglich.

Abb. 8/2 zeigt die Zusammenschaltung von CPU und DMA-Schaltkreis.

---

[2] *three-state* : Die entsprechenden Prozessorpins besitzen einen unendlich großen Ausgangswiderstand, über sie erfolgt kein Stromfluß, sie gelten als galvanisch getrennt vom System

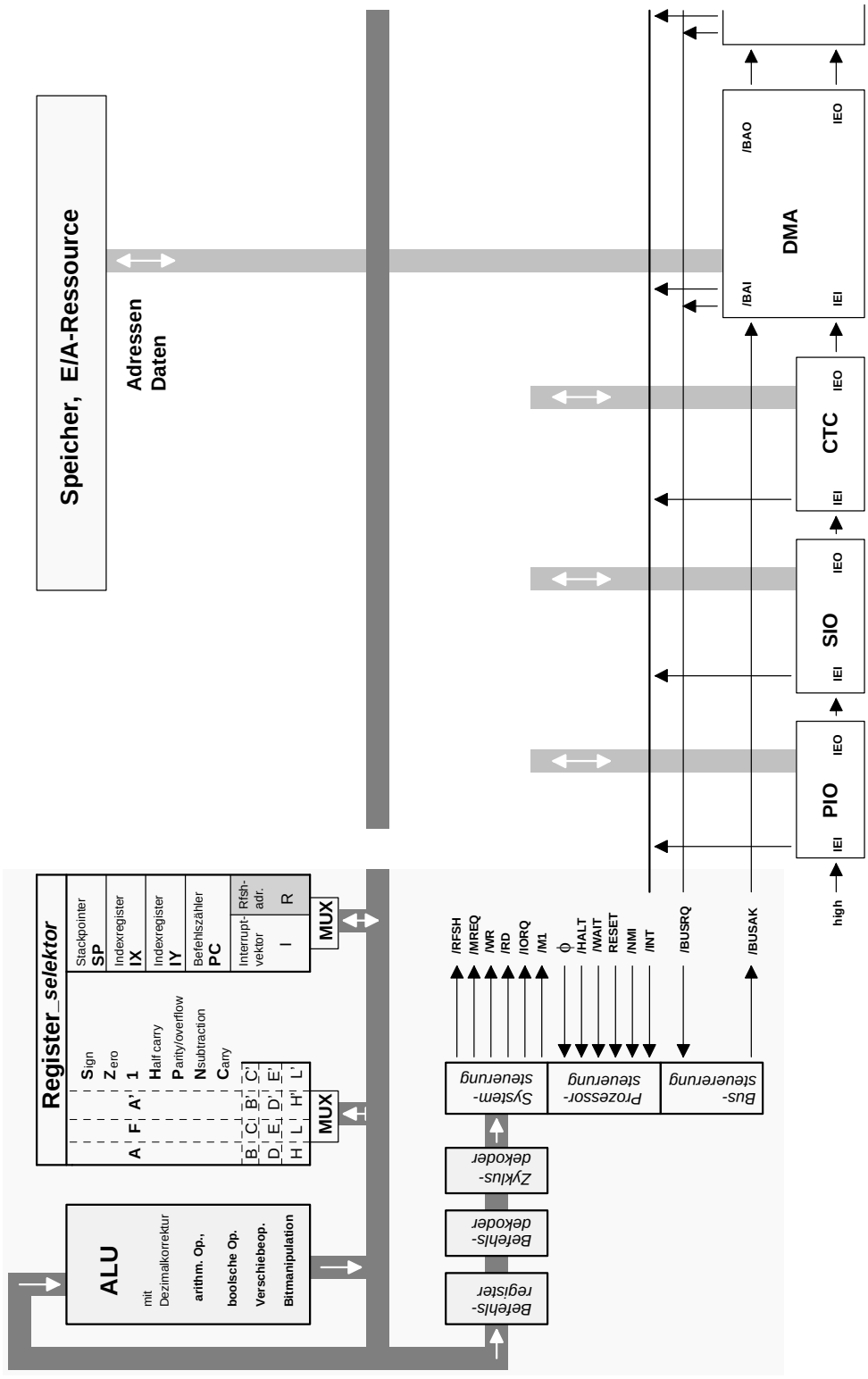


Abb. 8/2 Zusammenschaltung von CPU und DMA-Schaltkreis

Im Verkehr zwischen CPU und Peripherie-Schaltkreisen ist prinzipiell zu unterscheiden zwischen

***Unterbrechung* der CPU** durch einen Peripherieschaltkreis mittels *Interrupt-Signal* zwecks Veranlassung der CPU zur Durchführung einer *Unterbrechungsprogramms* für den peripheren Schaltkreis und der

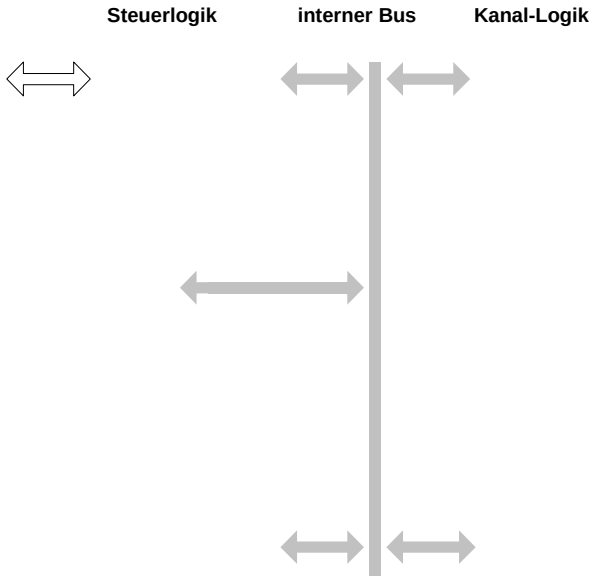
***Suspendierung* der CPU** durch einen peripheren Schaltkreis mittels *Busanforderungssignal* zwecks Erlangung der Systemmasterschaft für den peripheren Schaltkreis.

Funktionselemente eines DMA-Schaltkreises:

interne Steuerlogik
- zur Entgegennahme und Verarbeitung von Steuersignalen,
- zwecks Initialisierung des Schaltkreises,
- zur Bildung einer *daisy chain* zur Prioritätssteuerung,
- zur Generierung von Steuersignalen für den laufenden Datentransfer,
- zur Generierung des Datenenübertragungsendesignals

internes Bussystem
- zur Übertragung interner Steuersignale von der *internen Steuerlogik* zur *Kanalsteuerlogik*

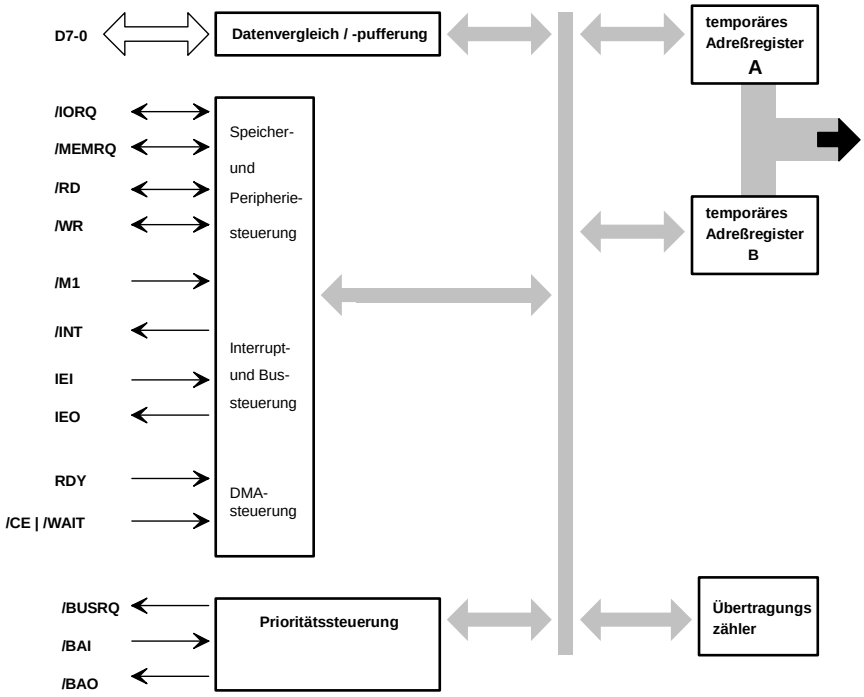
Kanalsteuerlogik
- zur Adressende(in)krementierung,
- zur Dekrementierung des Übetragungszählers,
- zum Byte-Vergleich



Funktionselemente eines DMA-Schaltkreises:

Die aufgeführten Funktionen verteilen sich über die verschiedenen Fabrikate der Schaltkreise.

2.4.2.1 ZILOG -DMA



Der Datentransfer erfolgt beim Z80-DMA über Pufferspeicher *durch den Schaltkreis hindurch* in 2 Zyklen (Lesen / Schreiben) der Dauer von jeweils 1...4 Takten

Die **Steuerlogik** ist untergliedert in

- ***Speicher- und E/A-Steuerung***
  zur Generierung bzw. auch zum Emfang von /MREQ, /IORQ, /RD, /WR ;

- ***Interruptsteuerung***
  zum Aufbau einer Interrupt-Kaskade (IEI, IEO) für die Generierung des Interrupt-Signals /INT und zum Bereitstellen des Interrupt-Vektors **IV** ;

- ***DMA-Steuerung***
  zur Entgegennahme des Ressourcenbereitschaftssignals RDY und des *Peripherieauswahlsignals* (Geräteadresse) /CE(A7-0) im Slavemode und des *Wartesignals* /WAIT im Mastermode ;

- ***byte compare control***
  zur Auslösung eines Interruptsignals /INT bei Koinzidenz zwischen dem Eingabebyte und einem internen "compare-Byte",
  vorausgesetzt: Die Interruptkaskade gestattet einen Interrupt,
  das Busanforderungssignal /BUSRQ ist passiv.

- ***priority control***
  zur Priorisierung des DMA als Busmaster mittels /BUSRQ=high, wenn die Busanforderungskaskade einen Buszugriff gewährt.

Die **Kanal-Logik** ist untergliedert in

- ***E- und A-Adreßregister*** *(16 Bit)*;

- ***word counter***
  zur Überwachung bzw. Protokollierung des Transfers;

- ***Blocklängenregister***
  zur Eintragung der zu transferierenden Blocklänge.

Die **Übertragung** zwischen Sender und Empfänger wird nach Schaltkreisinitialisierung

---

- **gestartet** mit **RDY=high** und der Implizierung /BUSRQ=low ... und
- **beendet** entsprechend der initialisierten Betriebsart: **Ü**bertragung
  oder
  Suchen.

Die **DMA -Betriebsarten**

- ***continuous mode*** bzw. ***Blocküübertragung*** (Abb. 9/2)
  Ohne Unterbrechungsmöglichkeit (d.h. RDY ohne Bedeutung) wird ein Block übertragen, der entweder durch seine *Länge* oder durch ein *compare Byte* limitiert ist (Speicher <===> Speicher).

- ***burst mode*** (Abb. 9/2)
  Die Übertragung erfolgt adäquat der Blocküübertragung, jedoch nur solange RDY=high (Speicher <===> E/A).

- ***transparent mode*** (Abb. 10/2)
  Die Byte-Übertragung erfolgt innerhalb des *refresh*-Zyklus', was eine externe Logik erfordert (Speicher <===> Speicher).

- ***byteweise Übertragung***
  Die Rückgabe der Busmasterschaft an die CPU erfolgt nach jeder Einzelbyteübertragung (PIO-Emulation) (Speicher <===> E/A).

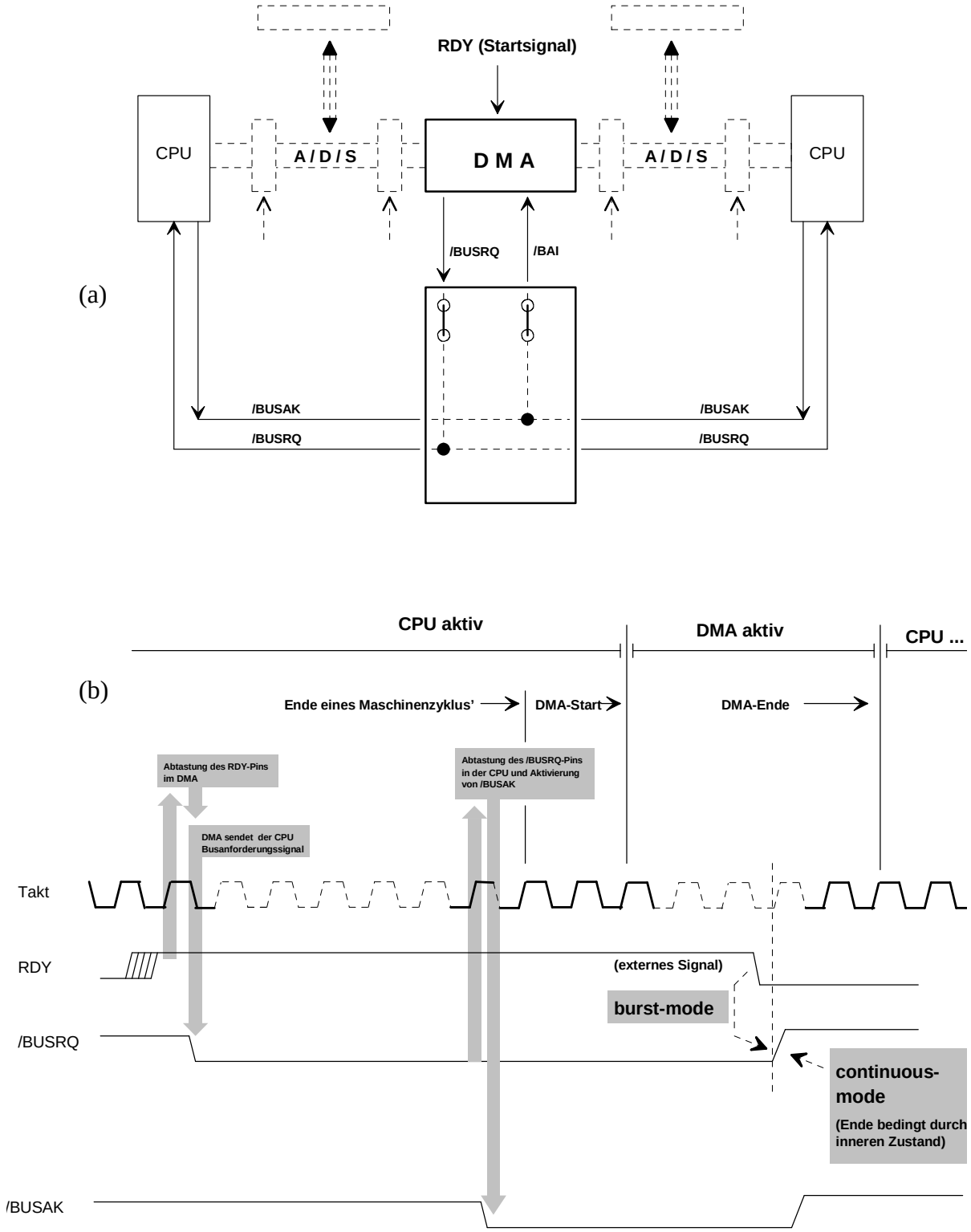
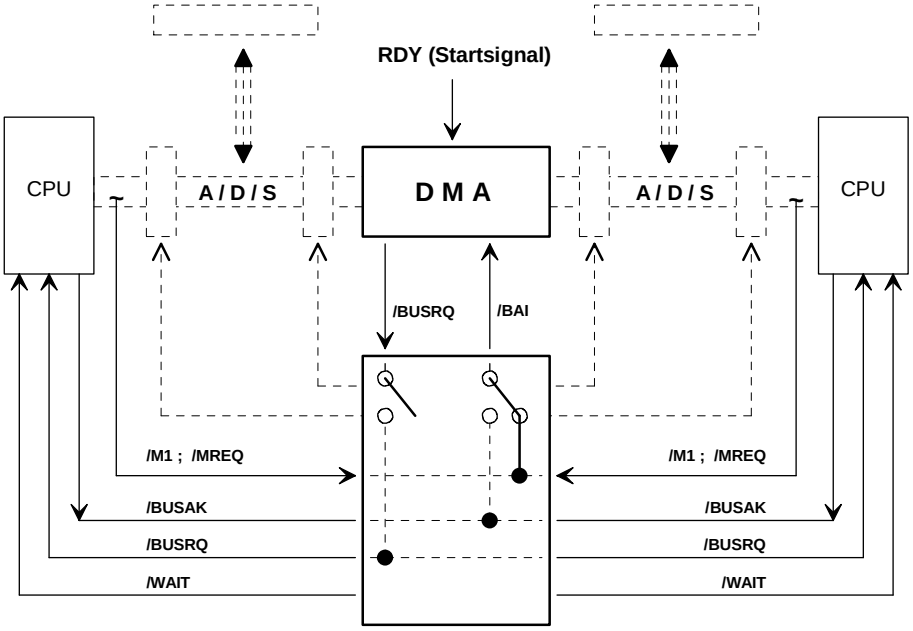


Abb. 9/2 Signalisierungen zwischen CPU und DMA-Schaltkreis in den Betriebsarten *continuous mode* bzw. *Blockübertragung* und *burst mode*
(a) Signalstruktur, (b) Signalverlauf

(a)



M1-Zyklus im DMA-*transparent_mode*

(b)

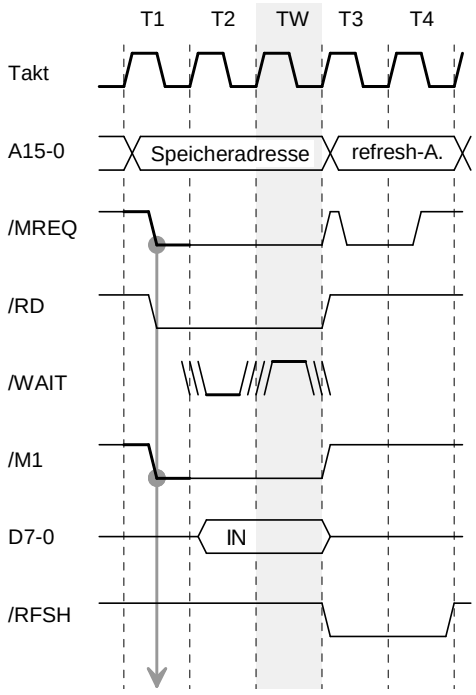
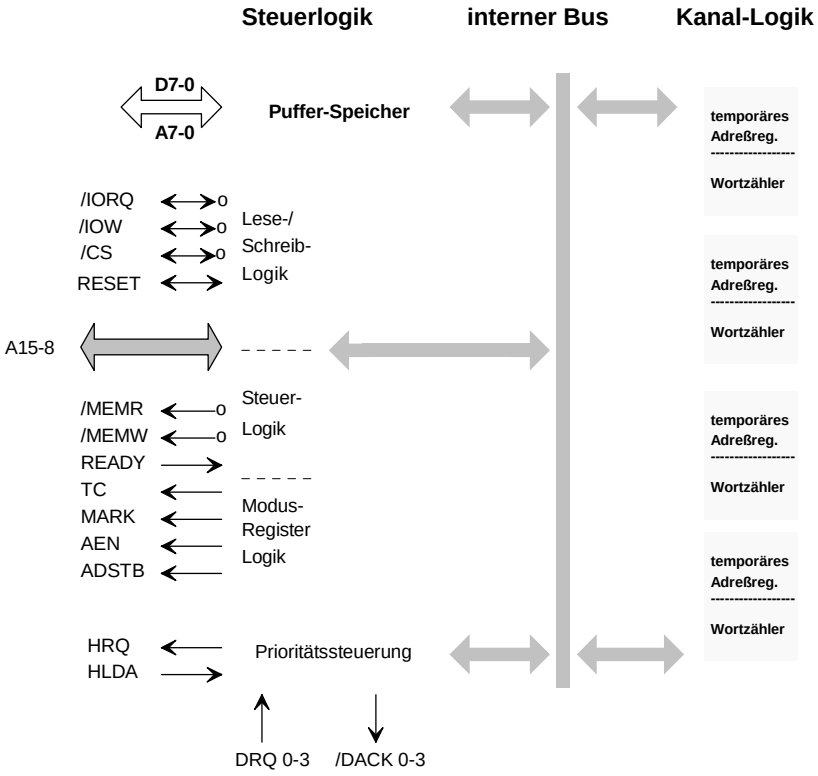


Abb. 10/2 Signalisierungen zwischen CPU und DMA-Schaltkreis in der Betriebsart *transparent mode* (a) Signalstruktur, (b) Signalverlauf

2.4.2.2 INTEL-DMA



Im Gegensatz zum Z80-DMA verläuft der Datenfluß außerhalb des DMA-Schaltkreises. Der DMA-Schaltkreis liefert nur **Lese-/Schreib-Adressen** sowie **Steuersignale.**

Die **DMA -Betriebsarten**

- *DMA-read*
  Daten werden vom DMA-Adressaten (Speicher, E/A-Ressource)
  zum DMA-Requester transferiert

- *DMA-write*
  ..... vom DMA-Requester
  zum DMA-Adressaten .....

- *DMA-verify*
  Funktionen wie vorstehend,
  jedoch ohne Generierung von I/O read/write,
  aber mit Bestätigungssignal für read/write an die Peripherie

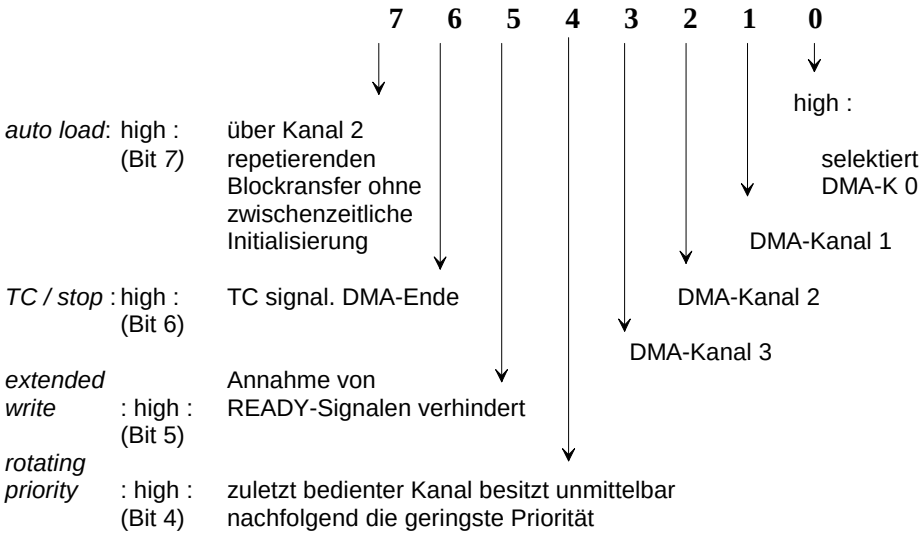
  (Betriebsart geeignet zur Durchführung von Zählvorgängen, für *refresh* und für Überwachungsaufgaben.)

Der **Registersatz des i8257** ist untergliedert in

- ***Datenpuffer***
  ausgeführt als *tri-state bidirectional 8-bit-register*
  zur Kopplung des i8257 an den System-Daten-Bus

- **mode set register**

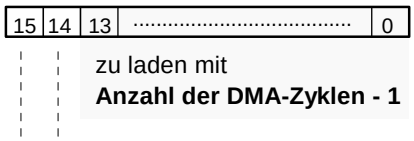
  zur Selektion von 1 aus 4 Betriebsmodi und 1 aus 4 DMA-Kanälen



4 DMA-Kanäle mit
je **2 Register** :

- ***Adreß-Register***
  zur Aufnahme der Startadresse für den zu lesenden bzw. zu schreibenden Speicherplatz

- ***Zähler-Register***



0	0	verify - DMA-Zyklus
1	0	DMA-Schreib-Zyklus
0	1	DMA-Lese-Zyklus
1	1	(unzulässige Bitkombination)

zzgl. • ***Status-Register***
zur Protokollierung jener Kanäle, welche die *Blockendebedingung* erreicht haben.
Das Erreichen wird mit dem Setzen der 4. Bitposition (*update flag*) angezeigt.

Interne Steuerlogik und *Kanal-Logik* generieren folgende Signale für den

DMA-Zyklus: ***Busanforderung → DMA-Transfer → DMA-Ende***

- *Busanforderungssteuerung*

  HRQ : Anforderung der Busmasterschaft;
  wird vom ix86-Prozessor am HOLD-Pin entgegengenommen

  HLDA: Bestätigung der Busmasterschaft durch den Prozessor

- *Lese-/ Schreib-Steuerung*
  zur Initialisierung (*slave mode)* — zur Generierung *(master mode)*

  RESET: zur Versetzung des Schaltkreises in den Anfangszustand

  READY : zur Synchronisation des i8257 mit der E/A-Ressource

  A3-0 : Registerauswahladressen
  A7-0 : zur Ergänzung der Speicheradresse

  /ADSTB : Strobe-Signal für high- Adreß-Byte[3]
  AEN : zur Signalisierung einer gültigen Adreßbelegung

  /IOW : Register mit Steuerwörtern schreiben
  /IOR : aus Register ..... lesen
  /CS : zur Schaltkreisselektion

  /MEMR
  /MEMW : zur Speichersteuerung

- *DMA-Ende-Steuerung*

  TC : zur Ankündigung eines letzten DMA-Zyklus'

  MARK : modulo-128-Signal

---

[3] (high- Adreß-Byte entstammt dem Prozessor, wird im Pufferschaltkreis 8212 zwischengespeichert und über /ADSTB mit dem low- Adreß-Byte aus DMA-Schaltkreis zur 16-Bit Adresse kombiniert.

Die ***Prioritäts-Steuerung*** favorisiert einen von vier Kanälen zum DMA-Transfer mit den Signalen:

- DRQ 0-3 : DRQ... sind konkurrierende Anforderungssignale an den DMA-Schaltkreis, falls nicht *rotating priority* vereinbart wurde (gesetztes Bit 4 im *mode set register*), ist Kanal 0 höchst und Kanal 4 niedrigst priorisiert

  Das gesetzte DRQ... muß während des laufenden Transfers beibehalten werden.

- /DACK0-3 : Aktivitätsverteilung :
  1-aus-4-Bestätigungssignal an die DMA-fordernde Ressource zur Signalisierung der DMA-Gewährung.

  Das Bestätigungssignal wird mit jedem transferierten Byte zurückgenommen und neu gesetzt.

Die **Übertragung** wird (nach erfolgter DMA-Initialisierung)

- gestartet mit DRQ ... [→ HRQ → HLDA → AEN]

und

- beendet mit TC → high bzw. mit DRQ...→ low.

Die gerichtete Verkopplung von Mikrorechnern erfolgt über Interface-Schaltkreise. Das nachfolgende Schaltungsbeispiel zeigt die Verkopplung zweier Mikrorechner mittels PIO- und DMA-Interface (Abb. 11/2) und den Ablauf einer Datenübertragung vom sendenden zum empfangenden Mikrorechner (Abb. 12/2).

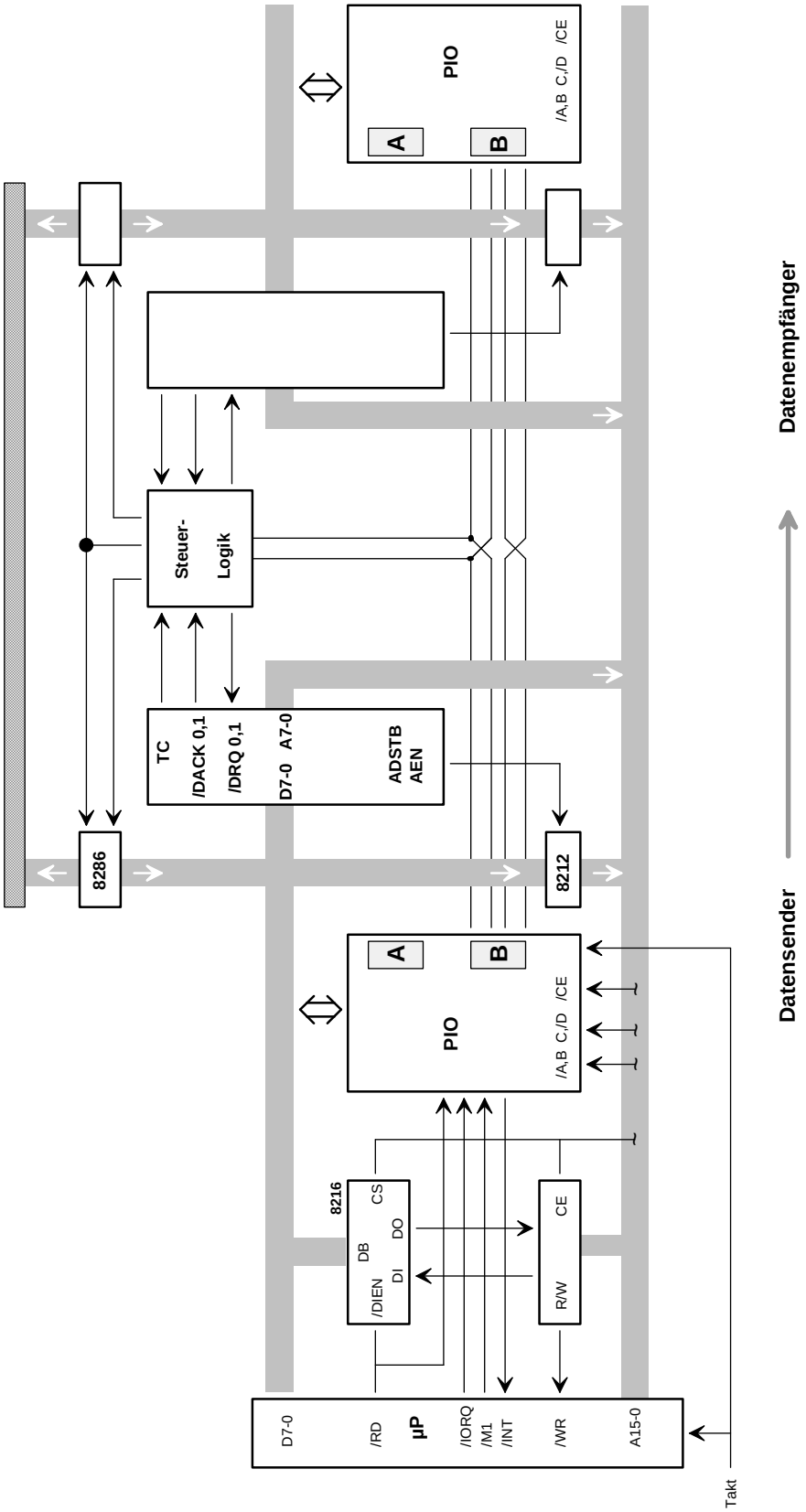


Abb. 11/2 Verkopplung zweier Mikrorechner mittels PIO- und DMA-Interface
Funktionsablauf einer Datenbyteübertragung

vom **SENDER** zum **EMPFÄNGER**

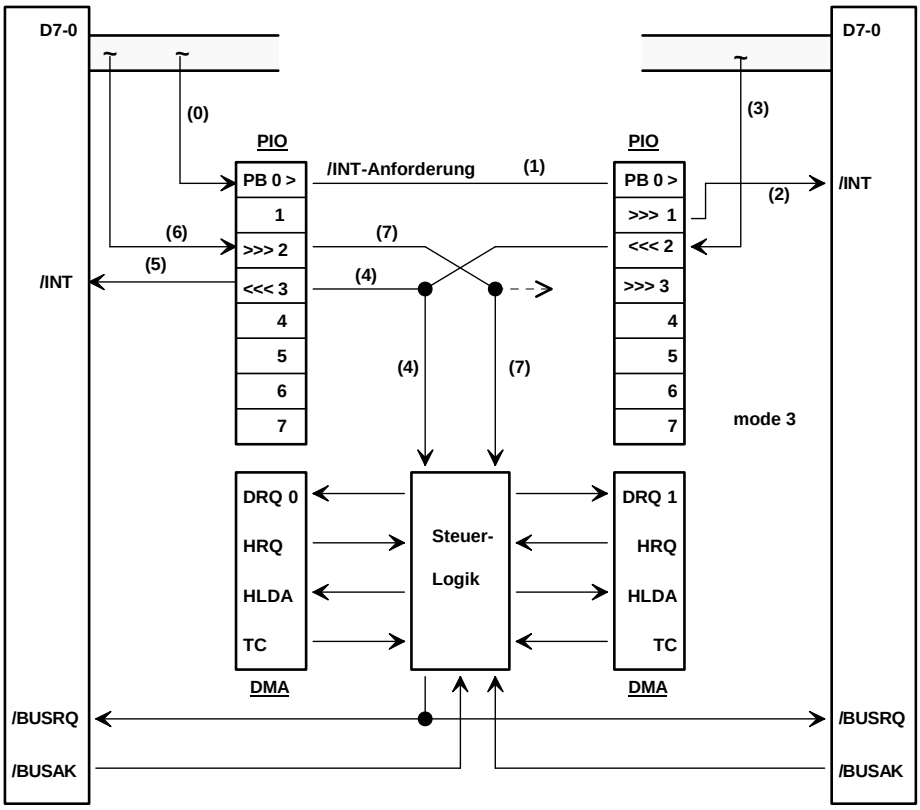


Abb. 12/2 Ablauf einer gerichteten Datenübertragung

Sender und Empfänger arbeiten autonom

(0) Ein bestimmtes Datenbyte setzt im Sender PB0

(1) PB0 generiert /INT-Anforderung auf Seite des Empfängers

(2) Empfänger nimmt /INT-Anforderung entgegen,
Abarbeitung einer Interruptroutine auf der Empfängerseite

(3) Empfänger quittiert die abgearbeitete Interruptroutine

(4) Protokollierung der Interruptabarbeitung in der Steuer-Logik,
DRQ**1**: low → high

(5) **und** /INT-Anforderung auf der Senderseite

(6) Sender quittiert die abgearbeitete Interruptroutine

(7) Protokollierung der Interruptabarbeitung in der Steuer-Logik,
DRQ**0**: low → high

Steuer-Logik HRQ_{Sender} & $HRQ_{Empfänger}$ → $/BUSRQ_{Sender\ \&\ Empfänger}$

Steuer-Logik $/BUSAK_{Sender}$ & $/BUSAK_{Empfänger}$ → $HLDA_{Sender\ \&\ Empfänger}$

Steuer-Logik TC_{Sender} or $TC_{Empfänger}$ → DRQ1 : high → low
DRQ1=low → HRQ : high → low
HRQ=low → /BUSRQ : low → high
/BUSRQ=high → /BUSAK : low → high → high

4. Ressourcen zur Bussteuerung

Mikroprozessorsysteme waren bis Ende der 70er Jahre durch die Komponenten *CPU* und *Speicher* charakterisiert. Sowohl die Verarbeitungsbreite auch der zu adressierende Speicherraum waren so gering, daß der Prozessor sowohl Verabeitungs- als auch Steuerfunktionen realisieren konnte.

Die Erhöhung der Verabeitungsbreite im Mikroprozessor wie auch die Vergrößerung des physischen Speichervolumens zu Beginn der 80er Jahre ging mit einer Zunahme des *systembedingten overheads* im Mikroprozessor einher. Beispiel dafür ist die Adreßverwaltung oder auch die Verwaltung der Befehls-Pipeline. Um die dafür notwendige auf dem Siliziumkern zu plazieren, mußten bestimmte Prozessorfunktionen extern gestützt oder auch ganz ausgelagert werden in

Spezialschaltkreise:

- ***INTERRUPT-CONTROLLER***
  zur Priorisierung von Unterbrechungsanforderungen an die CPU inkl. Bereitstellung der entsprechenden Interruptvektoren,

- ***BUS-CONTROLLER***
  zur Bereitstellung des Interrupt-Acknowledge-Signals INTA und zur Generierung von Systemsteuersignalen beim Speicher- und E/A-Zugriff (nicht zu verwechseln mit der prozessorinternen Speicherverwaltung zur Abbildung des virtuellen in den realen Adreßraum)

- ***ARBITER***
  zur Priorisierung von Zugriffen mehrerer Prozessoren und DMA-Geräte auf ein gemeinsames Bussystem

2.5.1 BUS-CONTROLLER

---

hier: - *CONTROLLER i..88*

PROZESSOR → **CONTROLLER** → **Co-Prozessor / SYSTEMBUS**

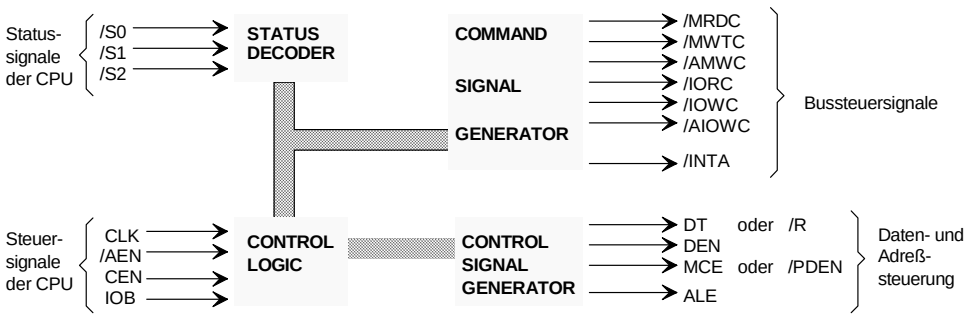


Abb. 13/2 Komponenten des BUS-CONTROLLERS i..88

Zur Steuerung der Datenübertragung existiert ein zum Bussystem **MULTIBUS** von INTEL kompatibler BUS-CONTROLLER (Abb. 13/2). Dieser ermöglicht den Durchgriff des Prozessors auf den **MULTIBUS**. Der Prozessor nimmt an diesem Durchgriff nicht aktiv teil, sondern veranlaßt ihn lediglich dazu durch die Statussignale /S2...0 (Abb. 14/2).

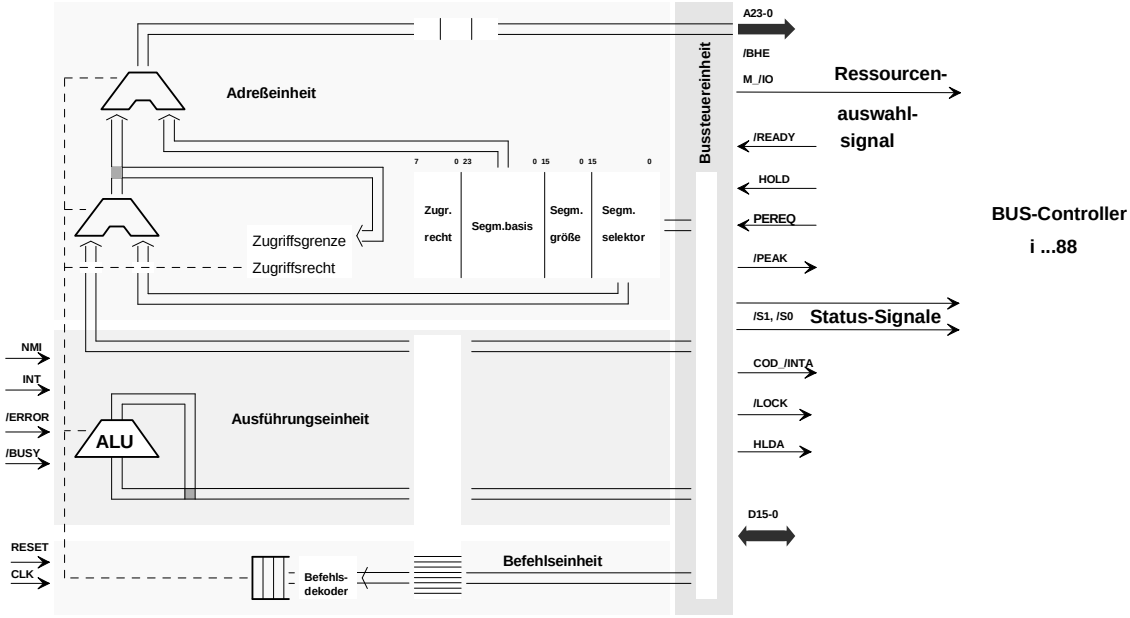


Abb. 14/2 Zusammenwirken von Prozessor und CONTROLLER gem. Abb. 13/2

Zusammengeschaltet mit einem Gleitkommaprozessor ...87 von INTEL konfiguriert der BUS-CONTROLLER ein Mikrorechnersystem zur beschleunigten Abarbeitung numerischer Operationen (Abb. 15/2).

Die *Bussteuereinheit* realisiert auf Anforderung der *Adreßeinheit* bzw. auf Anforderung durch den Co-Prozessor die Steuerung und Ausführung alle Arbeitsspeicherzugriffe.

Liegt keine Anforderung vor, erfolgt die Auffüllung der *Befehlswarteschlange* durch die *Bussteuereinheit* .

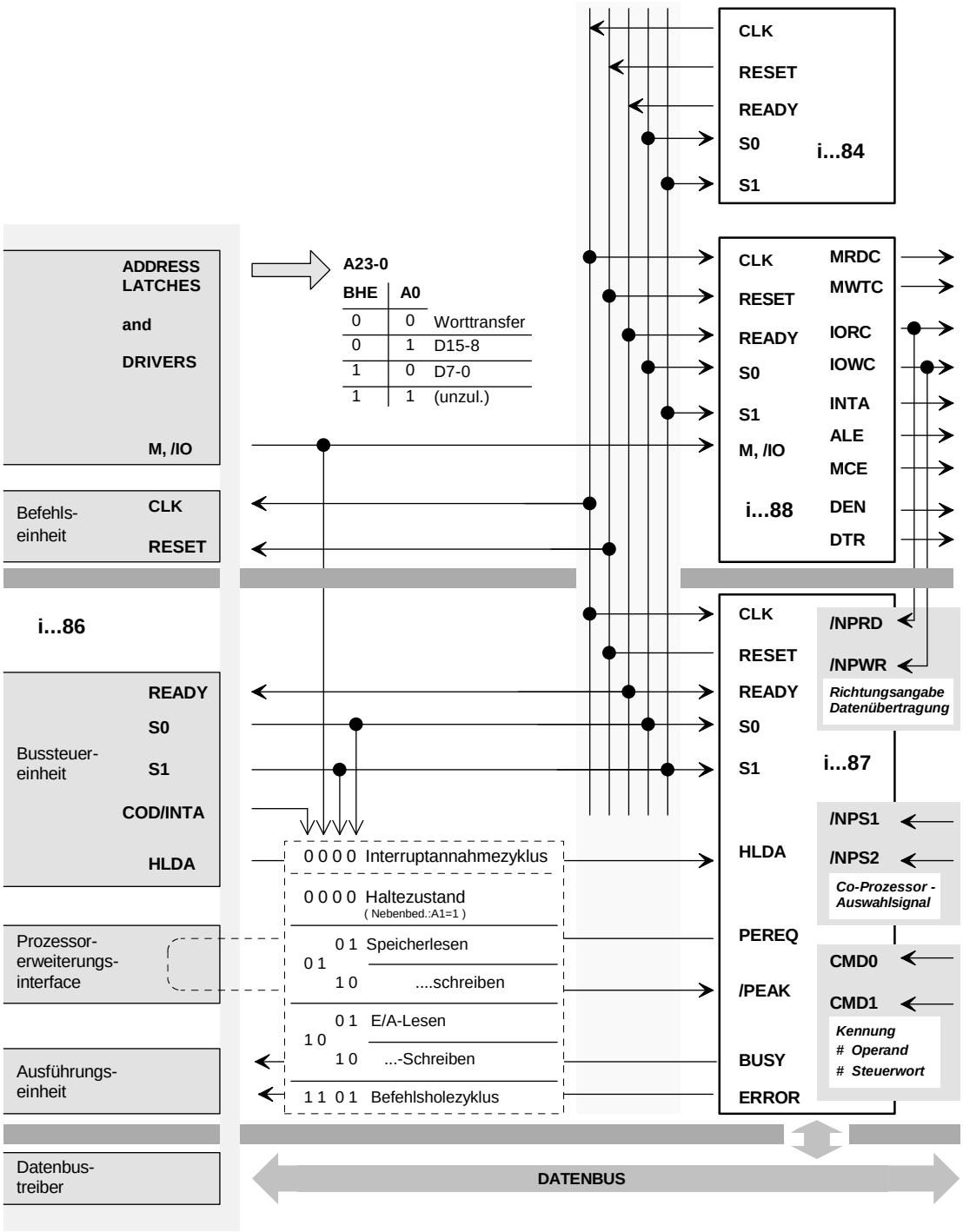


Abb. 15/2 Zusammenwirken von Prozessor, CONTROLLER und Gleitkommaprozessor

2.5.2 BUS-ARBITER

---

hier: - *ARBITER i...89*

PROZESSOR → ARBITER → SYSTEMBUS

Zur Steuerung der Busvergabe existiert ein spezieller Schaltkreis, bezeichnet als *ARBITER.*

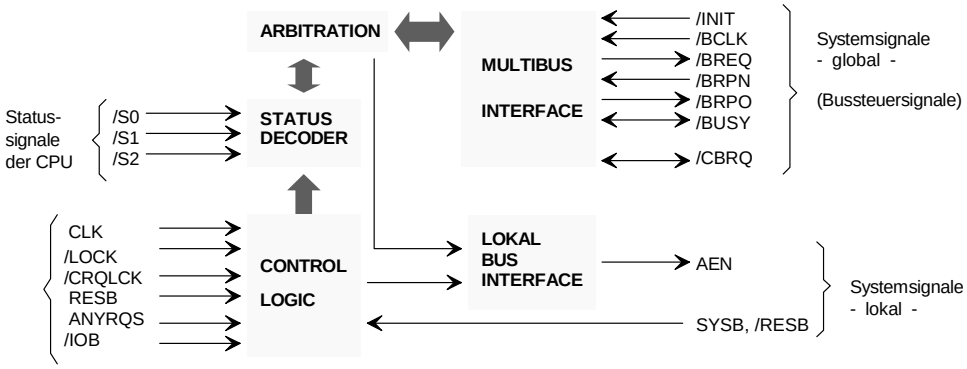


Abb. 16/2 Komponenten des ARBITERS i...89

Für den Prozessor ist der ARBITER (Abb. 16/2) eine tranparente Ressource; es gibt keine unmittelbare Rückwirkung vom ARBITER auf den Prozessor. Damit ist der ARBITER eine *passiv exklusive* Ressource, was heißt:

> Der Prozessor wird beim Ressourcenzugriff infolge Nicht-Priorisierung durch die Prioritätslogik in den Wartezustand versetzt.
> Der Ressourcenzugriff wird durch den ARBITER i...89 angemeldet und der Wartezustand des Prozessors durch den TIMER (Taktgeber i...84) veranlaßt (Abb. 17/2).

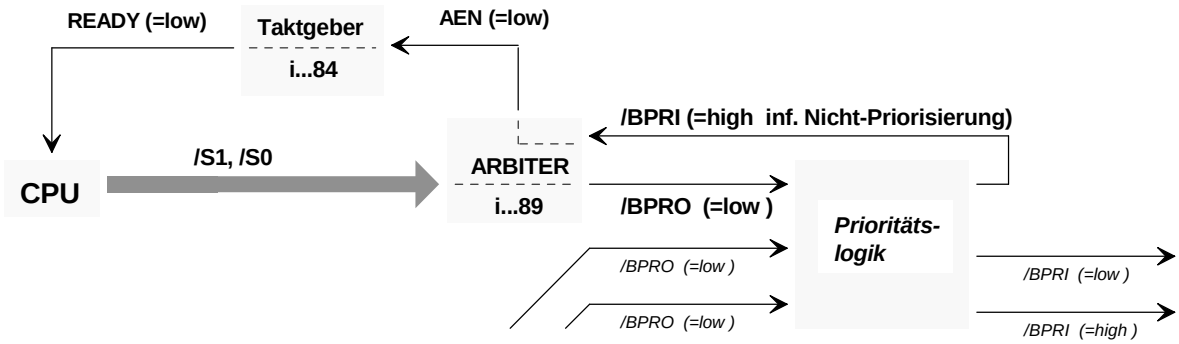


Abb. ... Zusammenwirken von Prozessor und ARBITER gem. Abb. ...

Zusammengeschaltet bilden *Prozessor*, *BUS-CONTROLLER* und *ARBITER* den komplexen Kern eines Mikrorechnersystems, in dem die *Verarbeitung der Information* und die *Steuerung* der Speicher- und Transfer-Ressourcen erfolgt (Abb. 18/2).

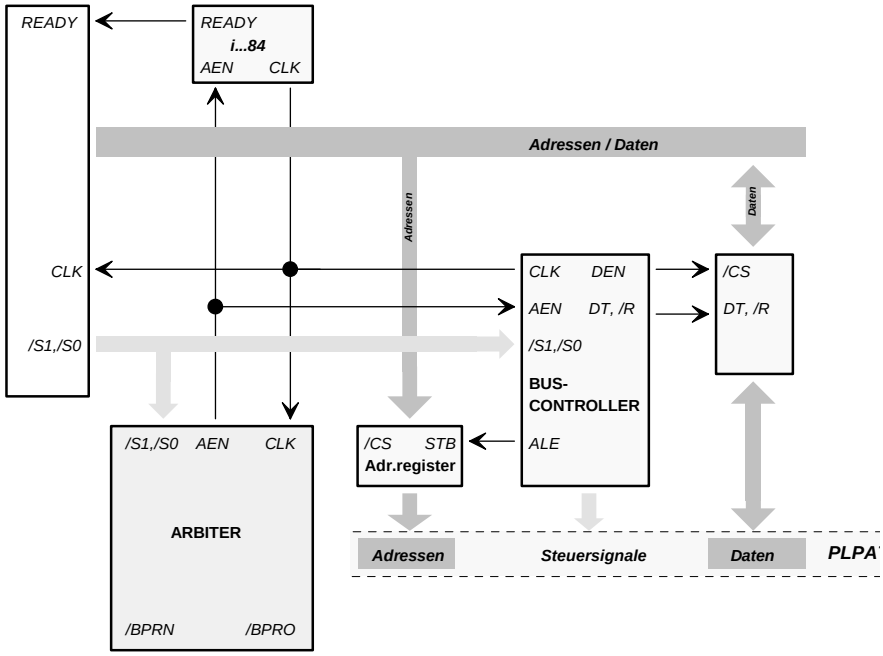


Abb. 18/2 Komplexer Kern eines Mikrorechnersystems, bestehend aus Prozessor, CONTROLLER, ARBITER

Der Mikroprozessor generiert Daten und Adressen im (Zeit-)Multiplex. Diese müssen getrennt, getrieben, gepuffert und über den PLATINEN-BUS verteilt werden.

Zur Trennung von Daten und Adressen erzeugt der BUS-CONTROLLER Steuersignale zur **Datenübertragungssteuerung**

- **Pufferung** des vom Prozessor generierten und für die Adressierung der Speicher- und E/A-Ressourcen bestimmten Adreßwortes,

- zur **richtungsabhängigen Durchschaltung** von Daten über einen *Datentreiber* im *tri-state*

Der PLATINEN-BUS verteilt die getrennten Daten und Adressen an die Systemressourcen einschließlich der vom BUS-CONTROLLER zur Steuerung dieser Ressourcen generierten Signale.

Während der einzelne(!) Mikroprozessor über den PLATINENBUS mit seinen Ressourcen verbunden ist und mit diesen ein *Mikroprozessorsysstem* konfiguriert, sind mehrere Mikroprozessorsysteme über einen SYSTEMBUS miteinander verbunder und konfigurieren mit diesem ein *Multiprozessorsystem*. Für die SYSTEMBUS-strukturierung und -organisation gibt es eine Reihe von Standards. Nachfolgend wird auf den MULTIBUS von INTEL Bezug genommen.

Der **ARBITER** eines Mikroprozessorsystems

- verwaltet den Zugriff des Prozessors auf den Systembus,
- erzeugt **Steuersignale zur Systembusvergabesteuerung**:

  - Entgegennahme der Zugriffsanforderung eines Prozessors auf den SYSTEMBUS ;
  - Eintragung der entgegengenommenen Forderung in eine Prioritätslogik ;
  - Versetzung des Prozessors in den Wartezustand, falls dessen Zugriffsanforderung auf Grund von Anforderungen konkurrenter Prozessoren durch die Prioritätslogik nicht bedient werden kann ;
  - Aufrechterhaltung der Zugriffsanforderung auf den SYSTEMBUS
    für die Dauer des Wartezustandes des Prozessors,
    der Zugriffsgewährung auf den SYSTEMBUS ;
  - Zurücknahme der Zugriffsanforderung auf den SYSTEMBUS nach erfolgter Zugriffsgewährung.

Grundsätzlich gilt:

- Zugriffsanforderungen müssen(!) *erhoben* und können(!) *gewährt* werden!
- Die *Forderungserfassung* kann *seriell* oder *parallel* erfolgen.

2.5.3 Prinzipien der Forderungserfassung

Serielle Forderungserfassung

(Abb. 19/2)

- **alle ARBITER längs *daisy chain verkettet***
  - stationäres Prioritätsgefälle
  - *ARBITER*, dessen /BRPN-Eingang bedingungslos low-Pegel führt, besitzt die höchste Priorität.

- **höchstpriorierter ARBITER signalisiert Zugriffs-Verzicht** auf den SYSTEMBUS durch
  low-Pegel an /BPRO-Ausgang

  entlang der *daisy chain* belegt dieser Pegel den Pegel den /BPRN-Eingang des in Flußrichtung unmittelbar benachbarten ARBITERs.

  Berechtigung auf Zugriffsanforderung

- **berechtigter und fordernder ARBITER führt**
  high-Pegel am /BPRO-Ausgang

  Diesem ARBITER

  - wird Zugriff auf SYSTEMBUS *gewährt*
  - aber (!) *Busmasterschaft* erst dann, wenn der momentan stattfindende Zugriff (signalisiert durch /BUSY=low) beendet worden ist (signalisiert durch /BUSY=high).

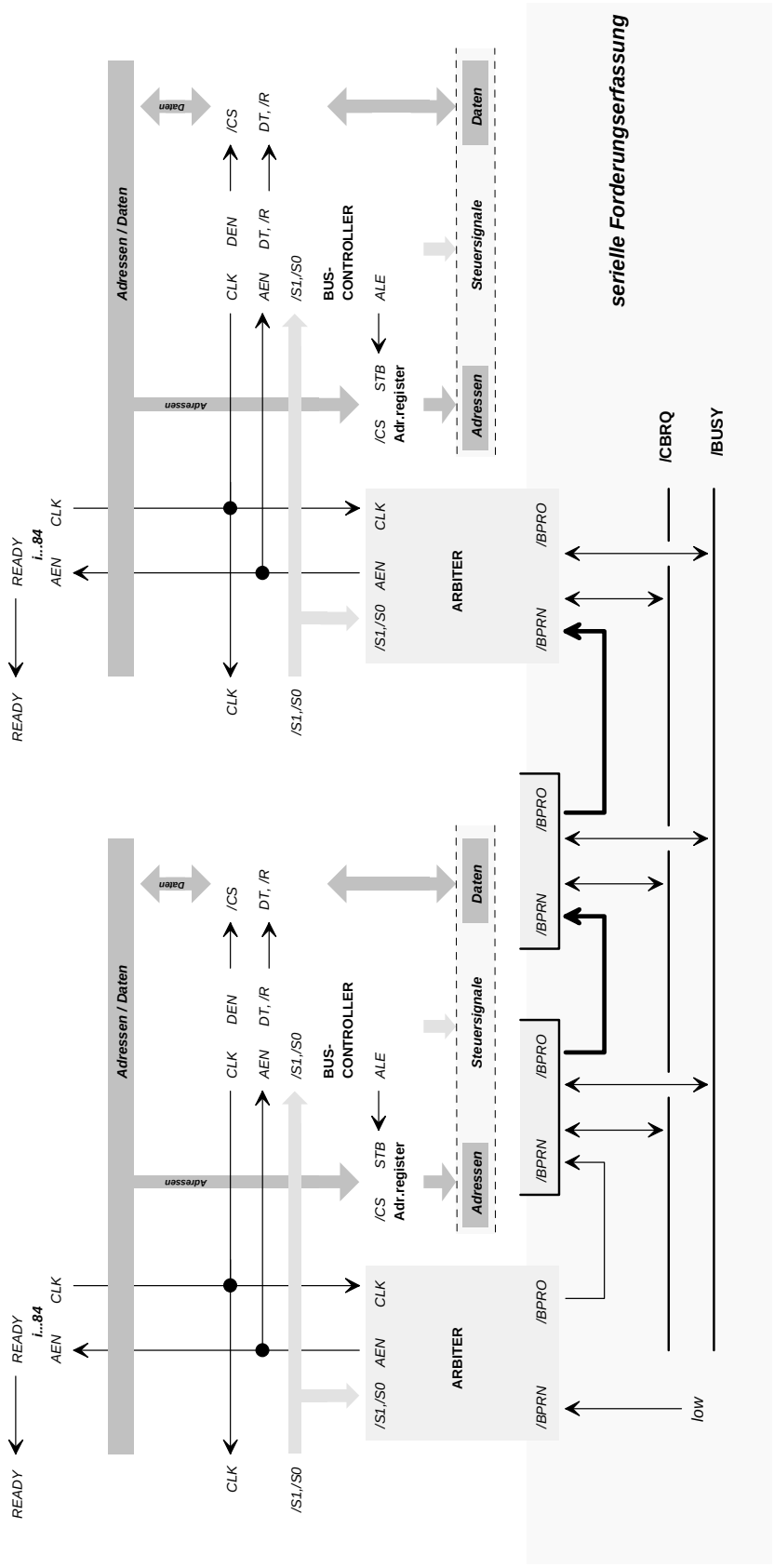
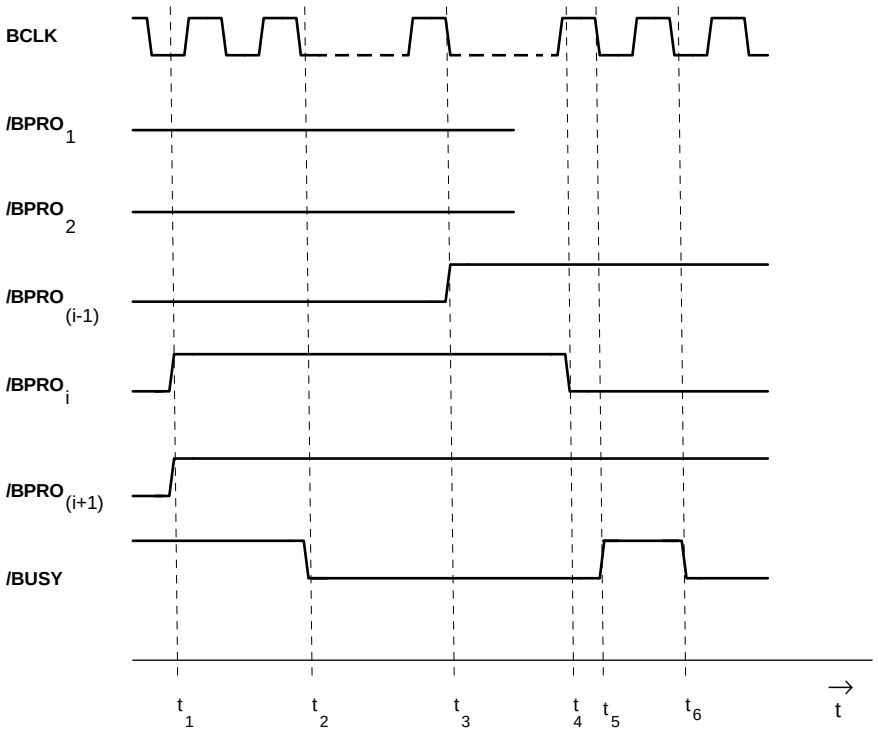


Abb. 19/2 Schaltungsanordnung zur seriellen Forderungserfassung

- *Ereignisfolge bei serieller Forderungserfassung* -



$t < t_1$ SYSTEMBUS ohne Zugriffsanforderung (/BUSY=high)

$t = t_1$ Prozessor (i) und Prozessor (i+1) erheben gleichzeitig eine Zugriffsanforderung auf den SYSTEMBUS

$t_1 < t < t_2$ Zeitintervall zur Priorisierung einer Anforderung und Vergabe der Busmasterschaft

$t = t_2$ Prozessor (i) erhält [weil längs der *daisy chain* höher priorisiert als Prozessor (i+1)] die Busmasterschaft (/BUSY=low)

$t = t_3$ Prozessor (i-1) erhebt eine Zugriffsanforderung auf den SYSTEMBUS

$t = t_4$ Prozessor i nimmt Zugriffsanforderung auf den SYSTEMBUS zurück

$t_4 < t < t_5$ Beendigung der Busvergabe für Prozessor i

$t_5 < t < t_6$ Zeitintervall zur Vergabe einer Busmasterschaft

$t_6 < t$ Wahrnehmung der Busmasterschaft durch Prozessor (i+1)

Parallele Forderungserfassung

(Abb. 20/2)

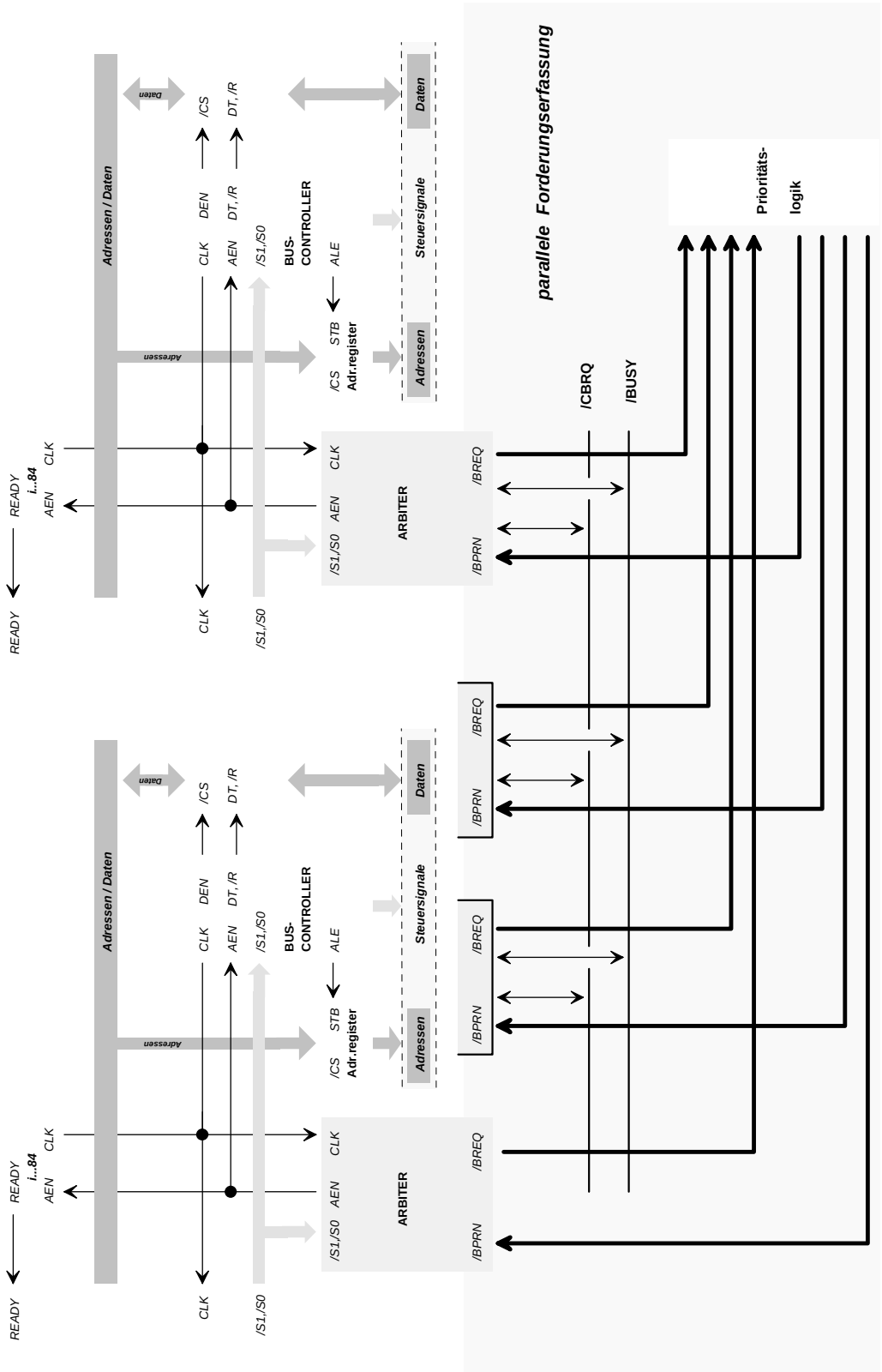
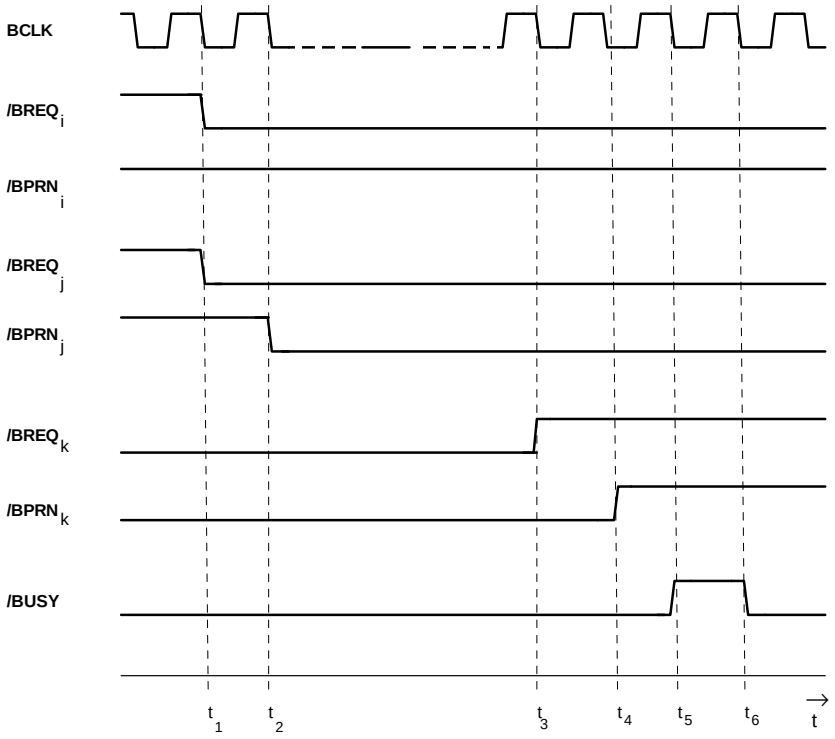


Abb. 20/2 Schaltungsanordnung zur parallelen Forderungserfassung

- Ereignisfolge bei paralleler Forderungserfassung -

Bei *paralleler Forderungserfassung* sind alle *ARBITER* mit einer *zentralen Prioritätslogik* verbunden. Die Prioritätslage ist somit nicht durch die Ortslage des *ARBITERs* bestimmt.

angenommene Prioritätsverteilung: Proz(k) < Proz (i) < Proz. (j)



$t < t_4$ Prozessor k besitzt die Busmasterschaft (/BUSY=low)

$t = t_1$ Prozessor (i) und Prozessor (j) erheben gleichzeitig eine Zugriffsanforderung auf den SYSTEMBUS

$t_1 < t < t_2$ Zeitintervall zur Priorisierung einer Anforderung und Vergabe der Busmasterschaft

$t = t_2$ Prozessor (j) wird gem. aktueller Forderungslage als nächster Busmaster priorisiert

$t = t_3$ Prozessor (k) nimmt Zugriffsanforderung auf den SYSTEMBUS zurück

$t = t_4$ Prozessor k bekommt Busmasterschaft entzogen

$t_4 < t < t_5$ Beendigung der Busvergabe für Prozessor k

$t_5 < t < t_6$ ARBITER-interne Reaktionszeit

$t_6 < t$ Wahrnehmung der Busmasterschaft durch Prozessor (j)

3. Architekturen parallel strukturierter Rechnersysteme

Innovationen der Mikrorechentechnik beruhen darauf, sowohl durch Vervielfachung der Prozessoren als auch durch Raumteilung des Speichers die Rechenleistung zu erhöhen.

Eine elementare Klassifikation von *Flynn* legt die Vielfachheit der Daten[1]- und Befehlsströme zugrunde. Von Bedeutung für parallele Informationsverarbeitungssysteme sind

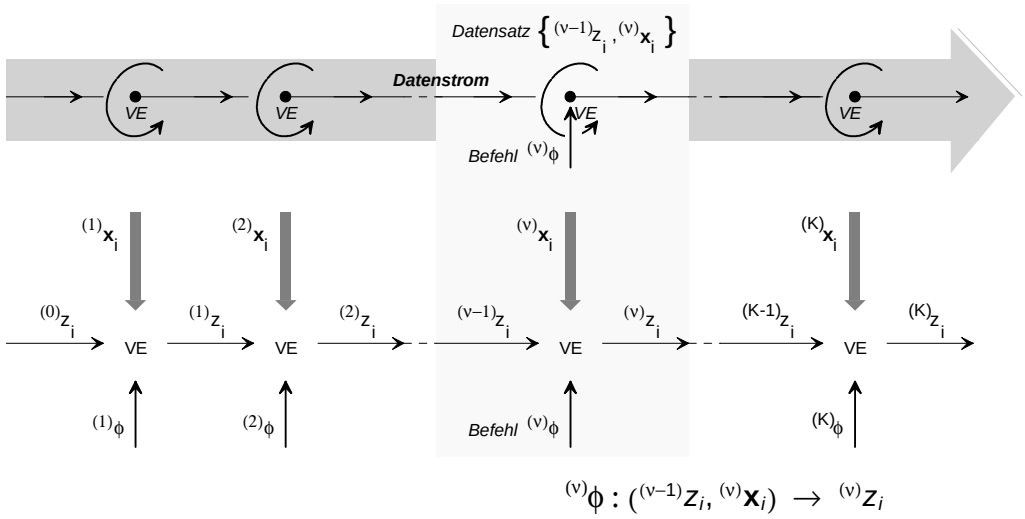
- SIMD-Architekturen (**S**ingle **I**nstruction **M**ultiple **D**ata)
  - Vektorrechner (Abb. 23/2)
  - Feldrechner (Abb. 24/2)
- MIMD-Architekturen (**M**ultiple **I**nstruction **M**ultiple **D**ata) (Abb. 25/2)

In einer SIMD-Architektur /GW 93/ wird ein und derselbe Befehl ϕ in einer Verarbeitungseinheit VE über mehrere *Datensätze* implementiert. *Datensätze* enthalten sowohl die einer Verarbeitungseinheit VE zugewiesenen *primären Operandenmenge* $\mathbf{x}$ als auch *Resultate* z zuvor implementierter Befehle. Das von einem Befehl erzeugte Resultat wird von einem nachfolgenden Befehl verarbeitet. Seien K Befehle über N Datensätze zu implementieren, dann entstehen insgesamt KN Resultate, klassifiziert in Zwischen- und Endresultate. Weiter im Detail wie folgt.

[1] unter *Daten* sind hier konkret *Operanden* zu verstehen

3.1 SIMD-Architekturen

Ein ***Vektorrechner*** treibt durch eine ***Kaskade von Verarbeitungselementen*** einen *Datenstrom*, bestehend aus *Datensätzen*.



Jedes Verarbeitungselement leistet eine bestimmte Verarbeitung (Multiplikation, Addition, ...) über jeden Datensatz längs des getriebenen Datenstromes und erzeugt ein Resultat. Ihr Architekturprinzip favorisiert ***Vektorrechner zur Implementierung von Vektoroperationen***. Die Bezeichnung *Vektorrechner* beruht auf der Vorstellung, daß über einen Satz von Vektoren elementeweise und ohne Wiederholung ein und dieselbe Operation ausgeführt wird (Beispiel a). Darüber hinaus kann ein *Vektorrechner* aber auch iterative Algorithmen implementieren, wird die Kaskade ringförmig geschlossen (Beispiel b).

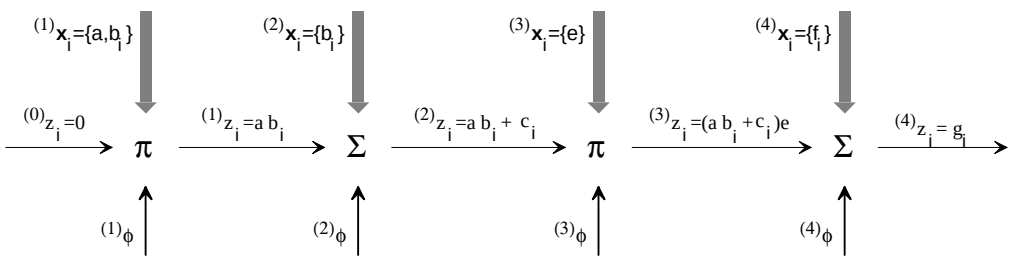
Beispiel a : Zu berechnen ist mit einem Vektorrechner der Ausdruck

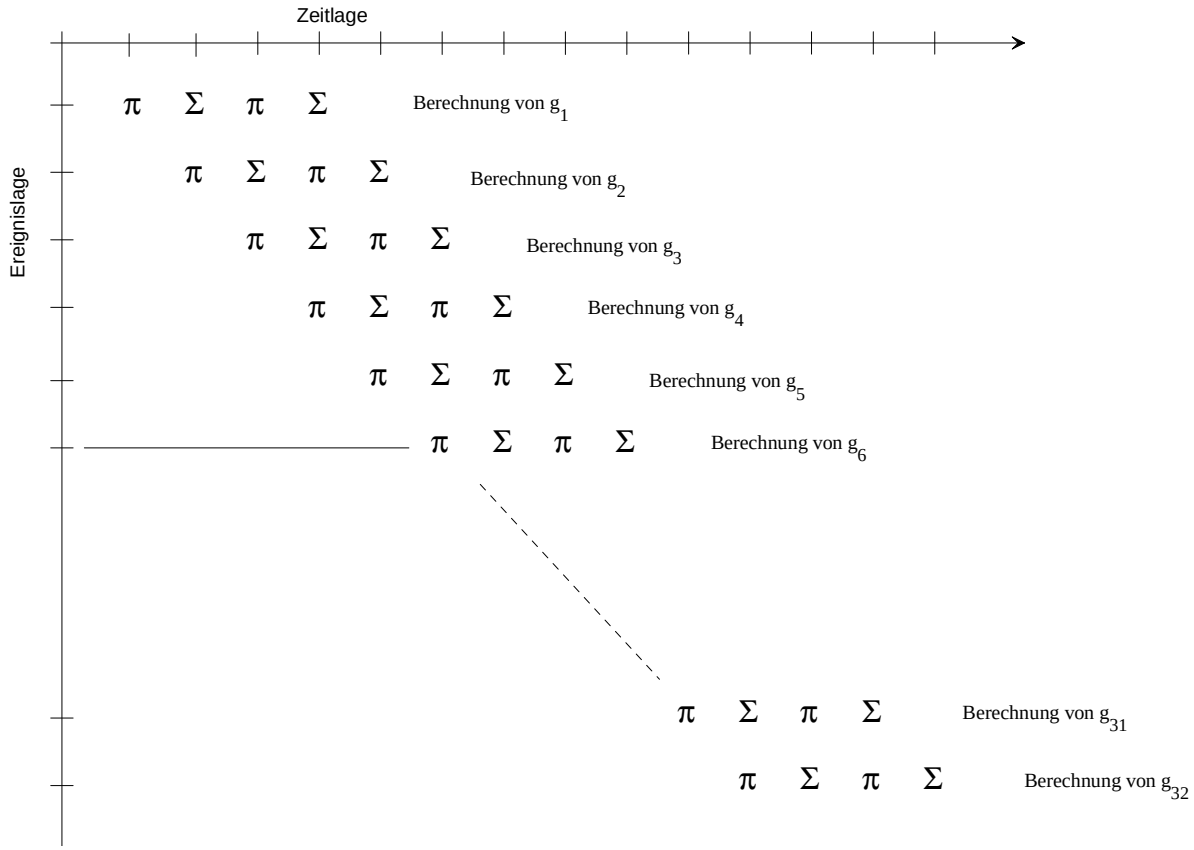
$$\vec{G} = \left(a \bullet \vec{B} + \vec{C} \right) \bullet e + \vec{F}$$

mit $\vec{B} = (b_1, b_2, ..., b_i, ..., b_N)$ dto. $\vec{C}, \vec{F}, \vec{G}$ für N=32

(a,e : skalare Parameter)

Der Ausdruck bedingt K=4. Die Implementierung von g_i geschieht wie dargestellt über 2 Summier- und 2 Multipliziereinheiten VE.





Beispiel b : Zu berechnen ist auf einem Vektorrechner fortlaufend die Iteration

$$x_{n+1} = a\left(x_n - \frac{b}{x_n}\right)$$

für a=b=0.5, x_0=1.5

Zur Einbeziehung von Zwischenresultaten längs der Kaskade von Verarbeitungselementen wird die Iteration wie folgt sequentialisiert:

$^{(0)}z = x_0$ (als Initialwert)

$^{(0)}z = {}^{(2)}z$ (zur Laufzeit)

$^{(1)}z = a\ {}^{(0)}z$

$^{(2)}z = {}^{(1)}z - \frac{a^2 b}{{}^{(1)}z}$

Nach dem n-ten Durchlauf des Datenstromes durch die Kaskade präsentiert $^{(2)}z$ den Iterationswert x_n. Wie Abb. 21/2 dargestellt, wird dieser Iterationswert auf den Eingang der Kaskade rückgeführt, durchläuft die Kaskade und erscheint ausgangs der Kaskade als Iterationswert x_{n+1}, repräsentiert durch $^{(2)}z$

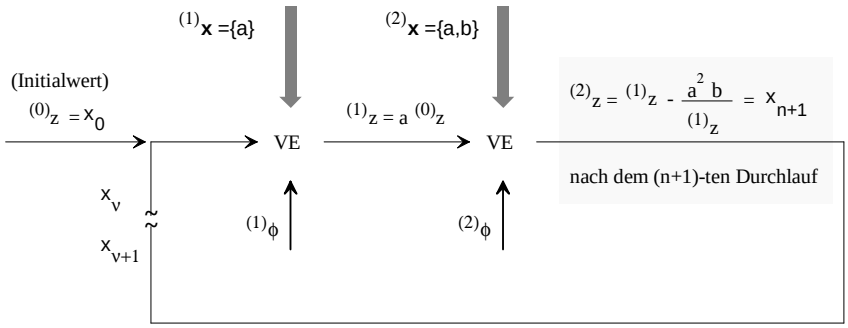


Abb. ... Prinzip der Implementierung von $x_{n+1} = a\left(x_n - \frac{b}{x_n}\right)$ in einem Vektorrechner

Den Verlauf von x_n für n=1, 2, ..., 100 zeigt Abb. 22/2.

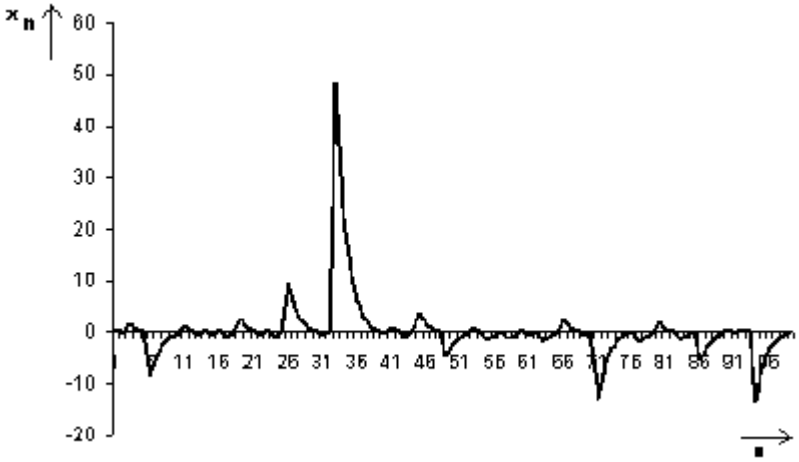


Abb. 22/2 Verlauf von x_n in der Iteration $x_{n+1} = a\left(x_n - \frac{b}{x_n}\right)$ für a=b=0.5 und x_0=1.5

Die Architektur eines Vektorrechners zeigt Abb. 23/2. Eine Skalareinheit liest einen Befehl aus dem Hauptspeicher ein, dekodiert diesen Befehl als Vektorbefehl und übergibt ihn zur Implementierung an die dafür vorgesehene Vektoreinheit. Die Vektoreinheit enthält K Verarbeitungseinheiten, die über eine Vektorsteuereinheit zu einer Kaskade gekoppelt sind. Abb. 23/2 zeigt eine Ausführungsvariante, bei der die Verarbeitungsinheiten durch die Skalareinheit entprechend dem zu impementierenden Befehl und damit problemorientiert spezifiziert sind. Die einzelnen Verarbeitungseinheiten haben sowohl lesenden als auch schreibenden Zugriff auf entsprechende Vektorregister, haben jedoch keinen direkten Zugriff auf den Hauptspeicher. Sämtlicher Datentransfer zur Implementierung des Vektorbefehls, d.h. sämtlicher Operanden- und Resultattransfer, erfolgt über die Vektorlade- und -speichereinheit inerhalb der Vektoreinheit.

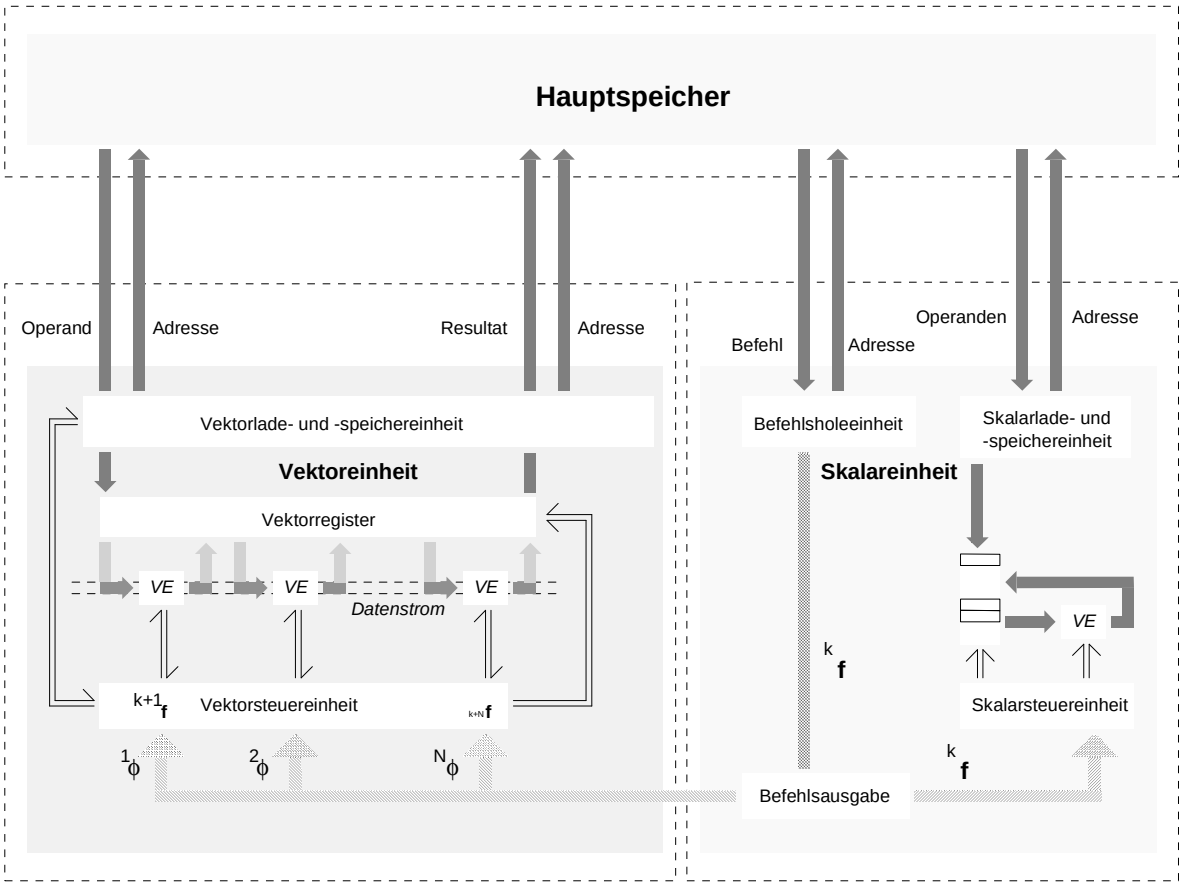


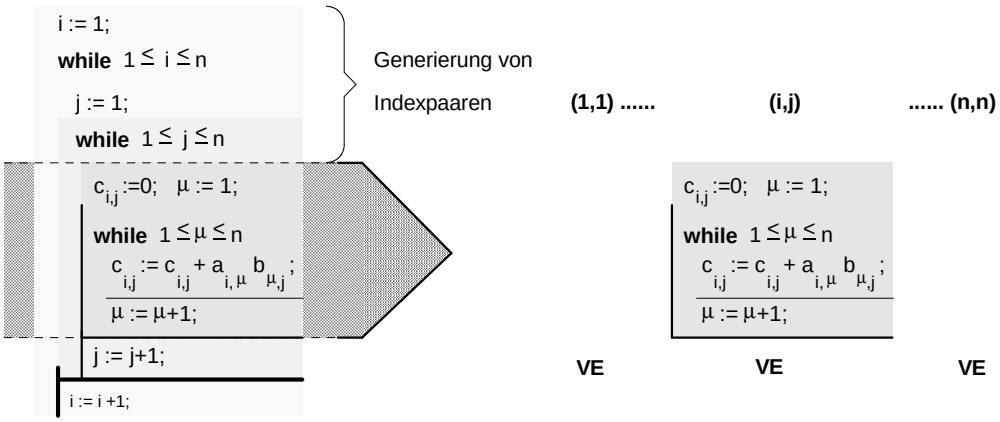
Abb. 23/2 Architektur eines Vektorrechners, bestehend aus *Vektor-* und *Skalareinheit*. In der Ereignislage k überweist die Skalareinheit der Vektoreinheit den Vektorbefehl ^{k}f zur Implementierung. Die dafür notwendigen Verarbeitungseinheiten VE werden durch die Steueranweisungen ϕ entsprechend ^{k}f spezifiziert.

Ein ***Feldrechner*** treibt durch ein ***Feld von Prozessoren*** voneinander unterschiedliche Datenströme, über die ein und derselbe Befehl implementiert wird. Auch im Feldrechner besteht ein *Datenstrom* aus mehreren *Datensätzen*. Im Gegensatz zum Vektorrechner, wo längs einer Kaskade von Verabeitungselementen jedes Verarbeitungselement einen einzigen Datensatz verarbeitet und alle Datenströme konsekutiv die Kaskade durchlaufen, implementiert in einem Feldrechner jeder Prozesor alle Datensätze des zugewiesenen Datenstroms. Mit anderen Worten heißt das: Ein Vektorrechner besteht aus wenigen, aber spezifischen Verarbeitungselementen, die konsekutiv von allen Datenströmen durchlaufen werden. Ein Feldrechner hingegen besteht aus multivalenten Prozessoren, die parallel und synchron zueinander über alle Phasen ein und desselben Befehls unterschiedliche Datenströme verarbeiten.

Gemäß ihrem Architekturprinzip sind ***Feldrechner zur Implementierung von Matrixoperationen*** geeignet und damit zur Implementierung von Algorithmen, die sich auf solche Operationen zurückführen lassen bzw. solche Operationen enthalten. Nachstehend wird die Eignung des Feldrechners zur Matrixmultiplikation $\mathfrak{C}=\mathfrak{AB}$ dargestellt.

Beispiel c : Seien $\mathfrak{A} = ||a_{i,j}||_{n,n}$ und $\mathfrak{B} = ||b_{i,j}||_{n,n}$ zwei miteinander zu mutiplizierende Matrizen und $\mathfrak{C}=\mathfrak{AB} = ||c_{i,j}||_{n,n}$ dessen Ergebnis mit $c_{i,j} = \sum_{\mu=1}^{n} a_{i,\mu} b_{\mu,j}$.

Wie dargestellt im Struktogramm zur Berechnung von $\mathfrak{C}=\mathfrak{AB}$ erzeugen die zwei äußeren Schleifen nacheinander (!) die Indexpaare (1,1), ..., (n,n). In der inneren Schleife erfolgt die Brechnung von c.



Die Bildung von c ist für alle Indexpaare $(i,j) \in \{(1,1), ..., (n,n)\}$ ein sowohl identischer als auch voneinander unabhängiger Prozeß und kann demzufolge für alle Indexpaare gleichzeitig erfolgen. Jedem Prozessor eines Feldrechners wird ein Indexpaar zugewiesen, dessen zugeordneter Wert c zu berechnen ist. Die Berechnung von c erfolgt im zugewiesenen Prozessor iterativ über jeweils n Phasen. Alle Prozesoren starten gleichzeitig die Brechnung von c und beenden, da es sich um identische Prozesse handelt, auch gleichzeitig die Berechnung von c.

Eine Ausführungsform eines Feldrechners zeigt Abb. 24a/2. Eine Skalareinheit weist jedem Prozessor ein und denselben Befehl zu; jeder Prozessor besitzt über das Verbindungsnetzwerk einen Zugriff auf den Hauptspeicher zur Einholung der entsprechenden Datensätze. Zum Beíspiel sind das zur Berechnung von $c_{i,j}$ die Datensätze $(a_{i,1}b_{1,j})$, ..., $(a_{i,i}b_{j,j})$,..., $(a_{i,n}b_{n,j})$. Der Prozeß beginnt in jedem Prozessor mit dem Einholen der Datensätze. Wurden alle Datensätze eingeholt, beginnt in allen Prozessoren (gleichzeitig !) die Brechnung von c. Prinzipiell können die Datensätze auch in einem zum Prozessor gehörenden Lokalspeicher bereits eingetragen sein. Dadurch entfällt das Einholen der Datensätze (Abb. 24b/2).

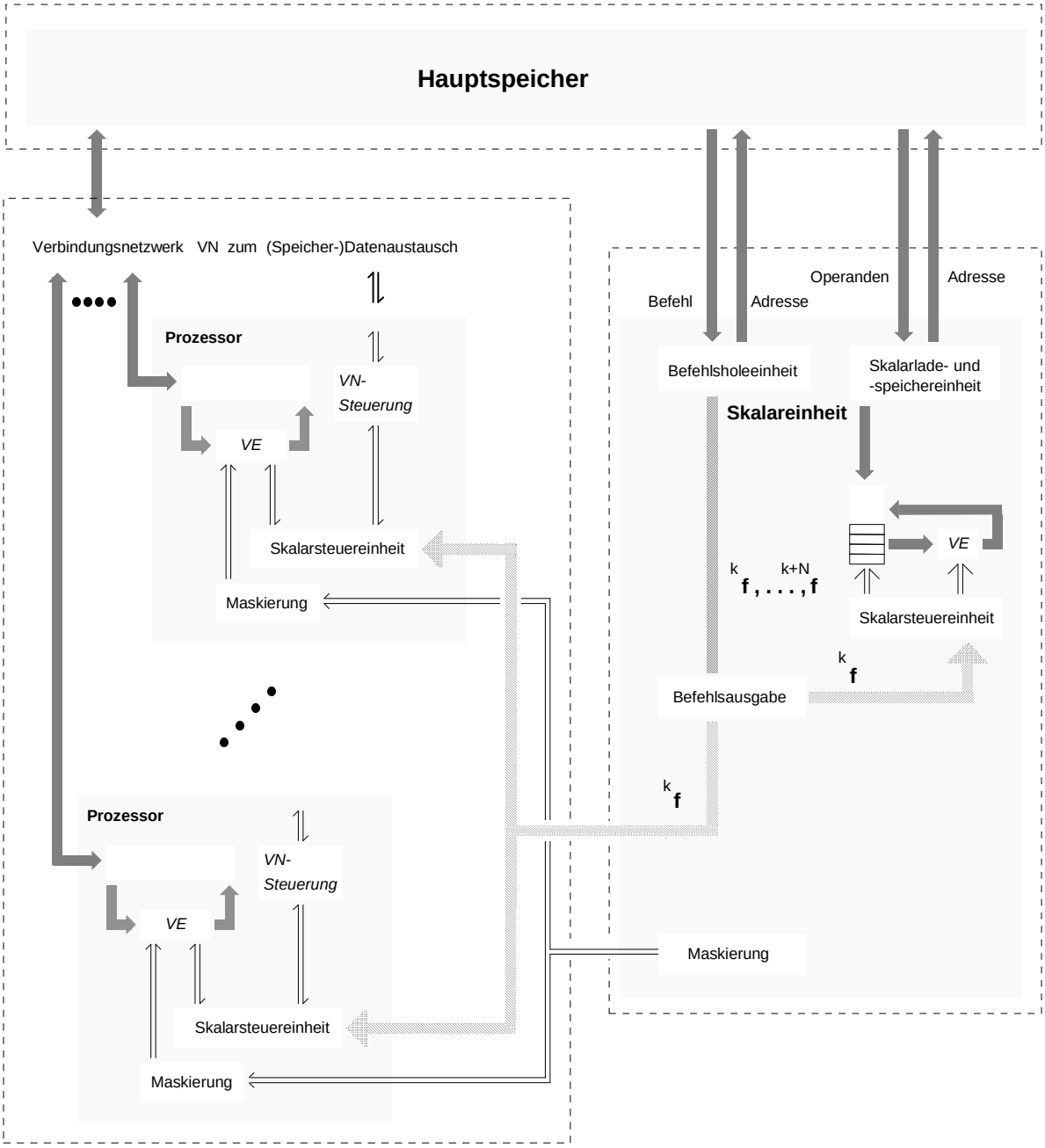


Abb. 24a/2 SIMD-Architektur eine Feldrechners mit einem gemeinsamen Speicher

Über das Verbindungsnetzwerk VN erlangen alle Prozessoren Zugriff auf den Adreßraum eines gemeinsamen Haupt-Speichers, über den auch der Datenaustausch stattfindet.

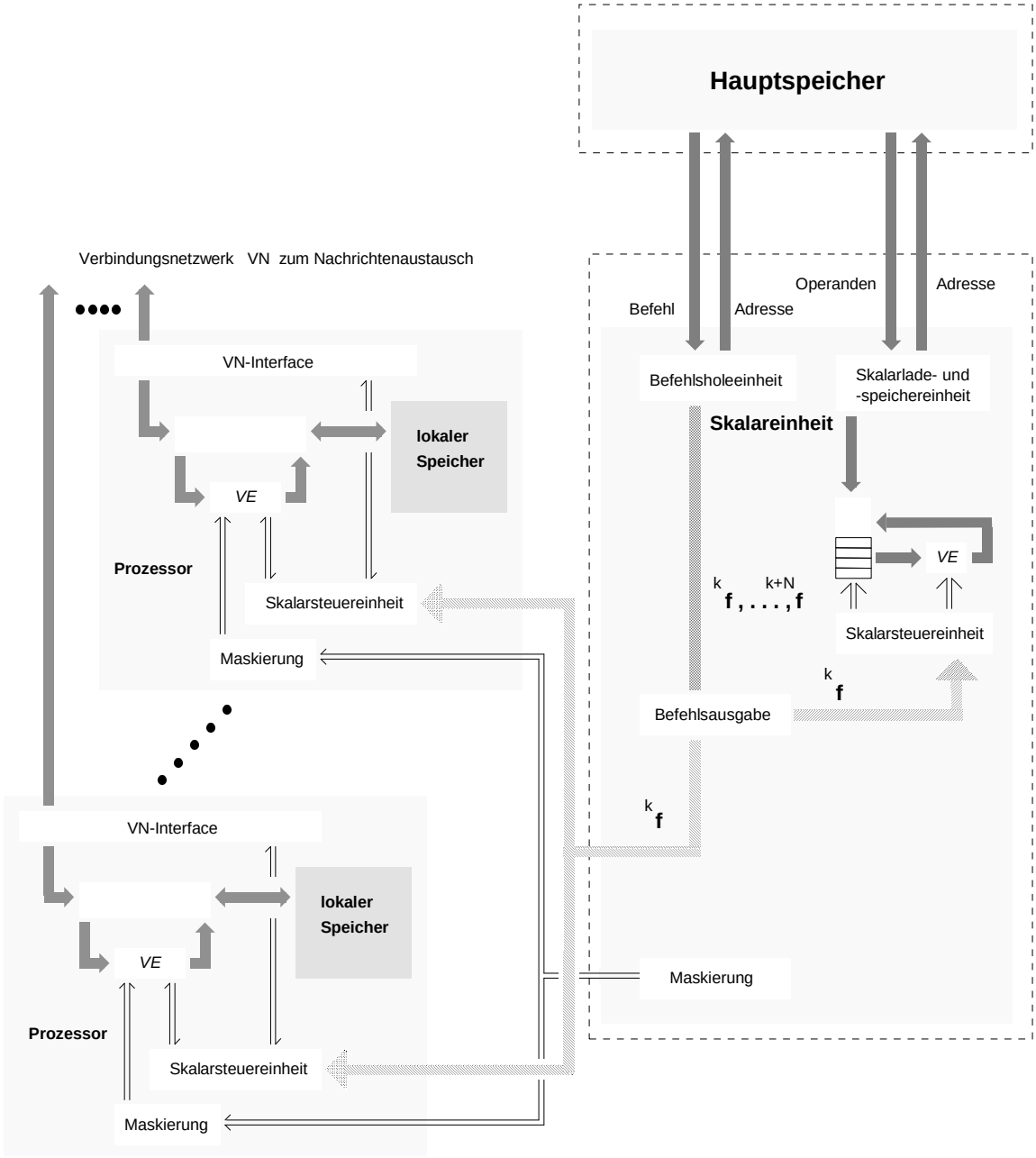


Abb. 24b/2 SIMD-Architektur eines Feldrechners mit lokalen Speichern

Die Interprozessorkommunikation erfolgt *nachrichtenbasiert* über das Verbindungsnetzwerk VN.

3.2 MIMD-Architekturen

Alle Datensätze können verschieden voneinander sein, und über alle Datensäze können verschiedene Befehle implementiert werden. In dieser Definition besitzt die MIMD-Architektur den größten Grad an Verallgemeinerung, die SIMD-Architektur stellt eine Untermenge dar. Ausführungsformen eines parallel strukturierten Informationsverabeitungssystem mit MIMD-Architektur zeigt Abb. 25/2

.

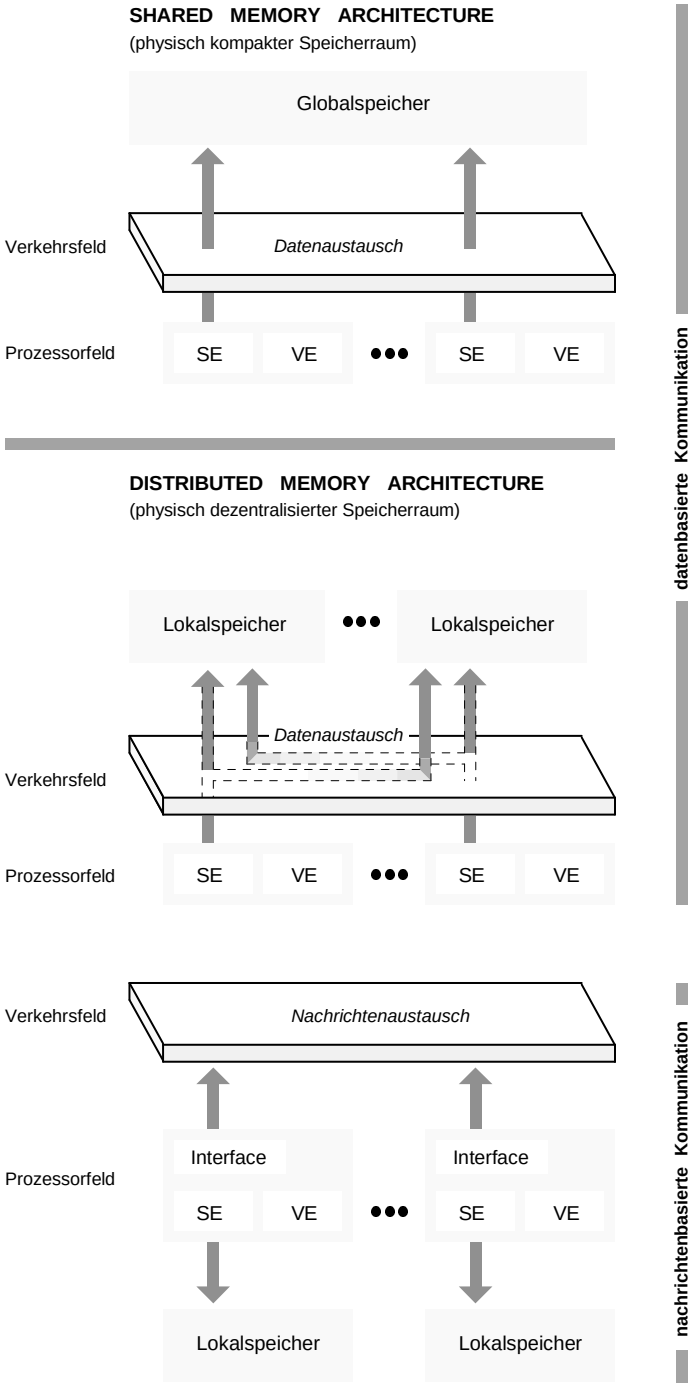


Abb. 25/2 Ausführungsformen eines parallel strukturierten Informationsverabeitungssystem mit MIMD-Architektur

In einem *parallel strukturierten Informationsverabeitungssystem* mit MIMD-Architektur werden entweder
- *Daten*

oder
- *Nachrichten* (*messages*) ausgetauscht.

Dieser *Austausch* erfolgt

entweder
- über einen gemeinsamen Speicher (*shared memory*)

oder
- über ein Verbindungsnetzwerk (*message passing*) /WK 98/.

Ein Verbindungsnetzwerk nimmt einen *Nachrichtenstrom* entgegen, transportiert diesen Strom und übergibt ihn am Zielort dem Empfänger. Zur Klassifizierung des Übertragungsprozesses sind die

Verbindungsnetzwerke nach ihrer
- *Verbindungsart* und ihrer
- *Topologie* einzuteilen.

Bezüglich *Verbindungsart* wird unterschieden zwischen
- *Leitungsvermittlung* und
- *Paketvermittlung* /KW 98/.

Leitungs- und Paketvermittlung

Bei der *Leitungsvermittlung* erfolgt der Verbindungsaufbau über einen physischen Kanal, der für die gesamte Dauer der Datenübertragung exklusiv reserviert bleibt.
Bei der *Paketvermittlung* nimmt der Nachrichtenstrom laufzeitaktuell nur einen Teilabschnite eines physischen Kanals in Anspruch. Im Idealfall stehen die zum Nachrichtentransport erforderlichen Teilabschnitte eines physischen Kanal genau dann zur Verfügung, wenn es die Fortschrittslage des Nachrichtenstroms längs seiner Bewegungsrichtung erfordert. Damit garantiert die *Paketvermittlung* eine intensivere Kanalauslastung als die *Leitungsvermittlung*, erfordert jedoch eine wesentlich kompliziertere Vermittlungsstrategie.

Bei der *Paketvermittlung* wird der zu übertragende *Nachrichtenstrom* in einzelne *Nachrichtenpakete* zergliedert, bestehend aus *Nachrichtenkopf* und *Nachrichtenkörper*. Der *Nachrichtenkopf* enthält die Information über den Empfänger des *Nachrichtenpaketes*, über dessen Länge und über dessen Zuordnung in dem zu übertragenden *Nachrichtenstrom*. Diese *Pakete* werden in das Verbindungsnetz eingetragen, werden zueinander asynchron durch das Verbindungsnetzwerk von Verkehrsknoten zu Verkehrsknoten transportiert und nach

erfolgtem Durchsatz durch das Verbindungsnetzwerk am Zielort wieder zum ursprünglichen *Nachrichtenstrom* rekonstruiert (***a**synchron **t**ransfer **m**ode **ATM***).

*store-and-forward_* und *worm-hole_*Strategie /WK 98/

Man bezeichnet in der *Paketvermittlung* ein Verfahren, bei dem die Nachrichtenpakete in jedem Verkehrsknoten vollständig zwischengespeichert werden, als *store-and-forward_*Strategie. Eine zentrale Steuerungsinstanz wertet die laufzeitaktuellen Ereignislagen der *Nachrichtenpakete* aus und kanalisiert sie dementsprechend durch das *Verbindungsnetzwerk* (*routing*). Alternativ dazu kann an Stelle einer zentralen Steuerungsinstanz auch jeder Verkehrsknoten die Weiterleitung der einzelnen *Nachrichtenpakete* administrieren. Man bezeichnet dieses Verfahren als *worm-hole_*Strategie[2]. Anhand des *Nachrichtenkopfes* bestimmt der *Verkehrsknoten* die Fortschrittsrichtung des durchgesetzten *Nachrichtenpaketes*. Stellt der Verkehrsknoten das Ende des transferierten *Nachrichtenpaketes* fest, dann beendet er für dieses Paket seinen Steuerungsdienstsatz.

Topologien von Verbindungsnetzwerken

Träger der genannten Strategien ist die *Topologie* des *Verbindungsnetzwerkes*. Nachfolgend wird unter *Topologie* die Präsentation des *Verbindungsnetzwerkes* bezeichnet. Abhängig vom Richtungsverhalten der im *Verbindungsnetzwerk* durchzusetzenden *Nachrichtenströme* bestehen zwischen den *Verkehrsknoten* spezifische Referenzmuster. Die Güte ist meßbar sowohl an der Laufzeit eines *Datenstromes* durch das *Verbindungsnetzwerk* als auch an der *Konfliktrate* konkurrierender Datenströme bei der Zuweisung von *Verkehrsknoten*.

In einem *parallel strukturierten Informationsverabeitungssystem* mit MIMD-Architektur werden entweder • *Daten*
oder • *Nachrichten* (*messages*) ausgetauscht.

Bezüglich *Rekonfigurierbarkeit* der Verbindungsnetze ist zu unterscheiden
zwischen • *statischer Topologie*
und • *dynamischer Topologie*.

---

[2] Ähnlich einem 'Wurm' schlängelt sich der in einzelne *Pakete* zergliedert *Nachrichtenstrom* in 'eigener Regie' durch das *Verbindungsnetzwerk*.

Vertreter sind *statischer Topologien* (Abb. 26/2)

- *orthogonal*,
- *hexagonal* und
- *toroidal* strukturierte Gitter in zweiter oder höherer Dimension.

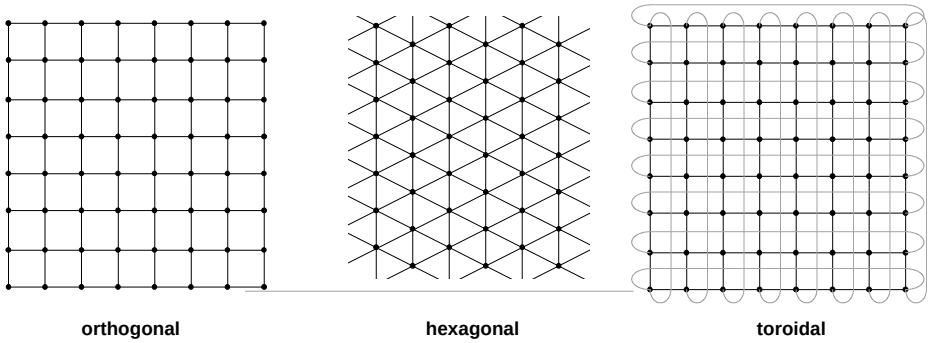


Abb. 26/2 Gitterstrukturen

Eine näherungsweise optimale *Topologie* besitzt der

- *Hypercube.*

Das Netzwerk des *Hypercube* verbindet $N=2^D$ *Verkehrsknoten* miteinander. Es ist D die Anzahl der abgehenden Verbindungskanäle eines *Verkehrsknotens*, gleichbedeutend mit der (ganzzahligen) Dimension des *Hypercube*. Sei

V_x ein mit $x \in \mathbf{X}=\{0,...,(N-1)\}$ indexierter *Sendeknoten*, von dem aus eine Direktverbindung auf D *Empfängerknoten* besteht, markiert als

V_y mit $y \in {}^{(x)}\underline{Y}=\{{}^{(x)}y_1, {}^{(x)}y_1, ..., {}^{(x)}y_i, ..., {}^{(x)}y_D\} \subseteq \mathbf{X}$.

Festzulegen ist die Auswahl der *Empfängerknoten* unter der

Randbedingung:
Jeder der $N=2^D$ *Verkehrsknoten* muß für jeden anderen *Verkehrsknoten* erreichbar sein, wenn aus jedem *Verkehrsknoten* D physische Verbindungen abgehen bzw. in jedem *Verkehrsknoten* D physische Verbindungen ankommen.

Die Frage lautet: Von welchem Knoten zu welchem Knoten ist zur Erfüllung der Randbedingung eine physische Direktverbindung herzustellen? Folgende Vorgehensweise besteht:

Es seien
- $x=x_B^L$ und $y=y_B^L$ *Ganze Zahlen* der Länge L zur (beliebigen) Basis B (i.d.R. B=10)

und
- $K_{B,2}$ eine Konvertierungsvorschrift zur Überführung
  von x_B^L nach $x_2^D=\sum_{j=0}^{D-1} x_j 2^j=(x_{D-1},x_{D-2},...,x_i,...,x_0)$ mit $x_i\in\{0,1\}$,
  d.h. $K_{B,2}: x_B^L \rightarrow x_2^D$

sowie
- $K_{2,B}$ eine Konvertierungsvorschrift zur Überführung
  von $^{(x)}\left(y_2^D\right)_i=\sum_{j=0}^{D-1} {}^{(x)}(y_j 2^j)_i=\left({}^{(x)}(y_{D-1})_i, {}^{(x)}(y_{D-2})_i, ..., {}^{(x)}(y_i)_i, ..., {}^{(x)}(y_0)_i\right)$ mit $y_i\in\{0,1\}$,
  nach $^{(x)}\left(y_B^L\right)_i$
  d.h. $K_{2,B}: y_2^D \rightarrow y_B^L$.

Die Indizes $y=y_B^L$ der *Empfängerknoten* V_y entstehen aus x_2^D durch konsekutive Negation der Stellenziffer x_i. Anschließend wird die so entstandene Dualzahl $^{(x)}\left(y_2^D\right)_i$ nach $^{(x)}y_i$ überführt mittels Konvertierungsvorschrift $K_{2,B}$,

d.h. $K_{2,B}: \left[\left(x_{D-1},x_{D-2},...,x_i\rightarrow\bar{x}_i,...,x_0\right)={}^{(x)}\left(y_2^D\right)_i\right]\rightarrow{}^{(x)}\left(y_B^L\right)_i \quad \forall i\in\{1,2,...,D\}$

Beispiel:

Ein *Verbindungsnetzwerk* bestehe aus N=32 *Verkehrsknoten,* demzufolge gilt D=5 . Es sei B=10. Für den *Verkehrsknoten* V_{23} mit $x=x^2_{B(=10)}=23$ entsteht durch Konvertierung mitels $K_{B,2}$ der korrespondierende Index $x_2^{D(=5)}=(\ 10111\ )$. Die *Empfängerknoten* von V_{23} berechnen sich wie dargestellt.

$K_{2,10}$: [(**1→0,** 0 1 1 1) = $^{(x)}\left(y_2^D\right)_1$] → $^{(x)}\left(y_{10}^2\right)_1$ = 7

$K_{2,10}$: [(1 **0→1,** 1 1 1) = $^{(x)}\left(y_2^D\right)_2$] → $^{(x)}\left(y_{10}^2\right)_2$ = 31

$K_{2,10}$: [(1 0 **1→0,** 1 1) = $^{(x)}\left(y_2^D\right)_3$] → $^{(x)}\left(y_{10}^2\right)_3$ = 19

$K_{2,10}$: [(1 0 1 **1→0,** 1) = $^{(x)}\left(y_2^D\right)_4$] → $^{(x)}\left(y_{10}^2\right)_4$ = 21

$K_{2,10}$: [(1 0 1 1 **1→0,**) = $^{(x)}\left(y_2^D\right)_5$] → $^{(x)}\left(y_{10}^2\right)_5$ = 22

Für alle *Verkehrsknoten* V_0, ..., V_{31} als *Quellknoten* existieren die in Abb. 27a/2 durch ■ markierten *Zielknoten*. Die mit unterschiedlichen Grautönen belegten Flächen in Abb. 27b/2 markieren die topologische Weglänge von einem ausgewählten *Quellknoten* zu einem ausgewählten *Zielknoten*; je dunkler der Grauton, desto ausgedehnter die topologische Weglänge. Über höchstens (D=)5 physische Kanäle hinweg erreicht jeder *Verkehrsknoten* jeden anderen *Verkehrsknoten*.

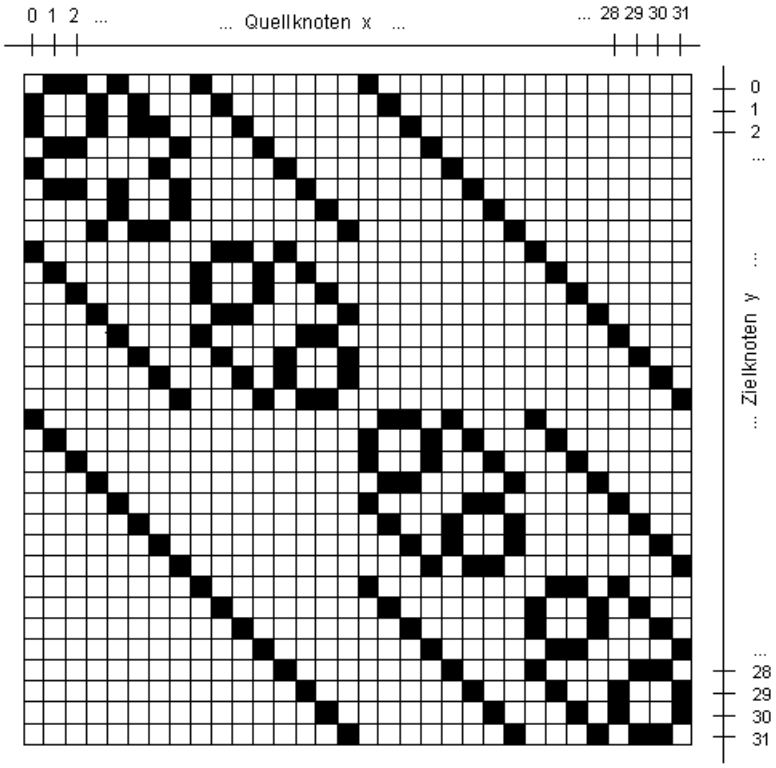


Abb. 27a/2 Matrixdarstellung für die Verknüpfung von Quellknoten mit Zielknoten in einem *Hypercube* mit 32 *Verkehrsknoten*

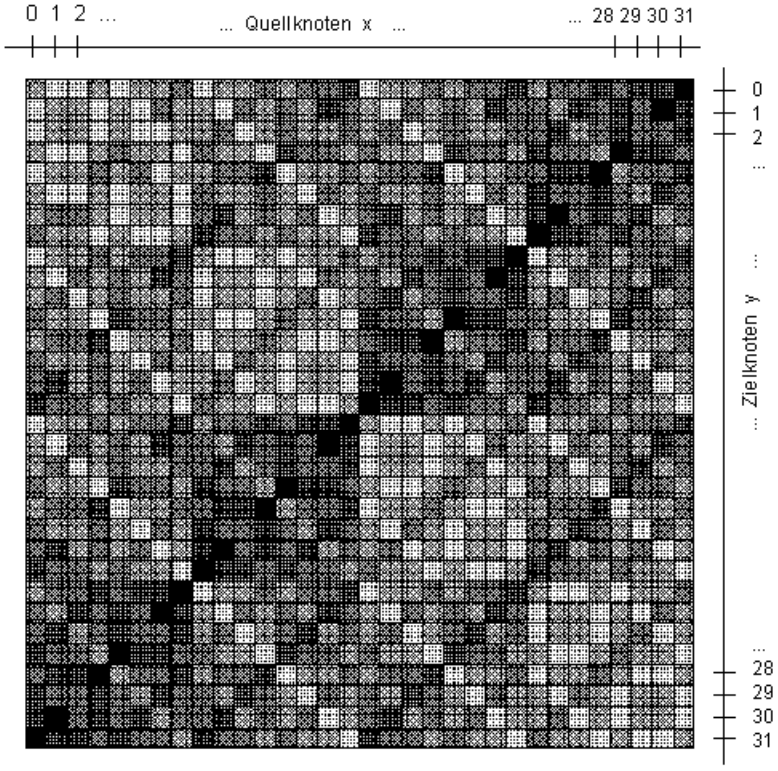


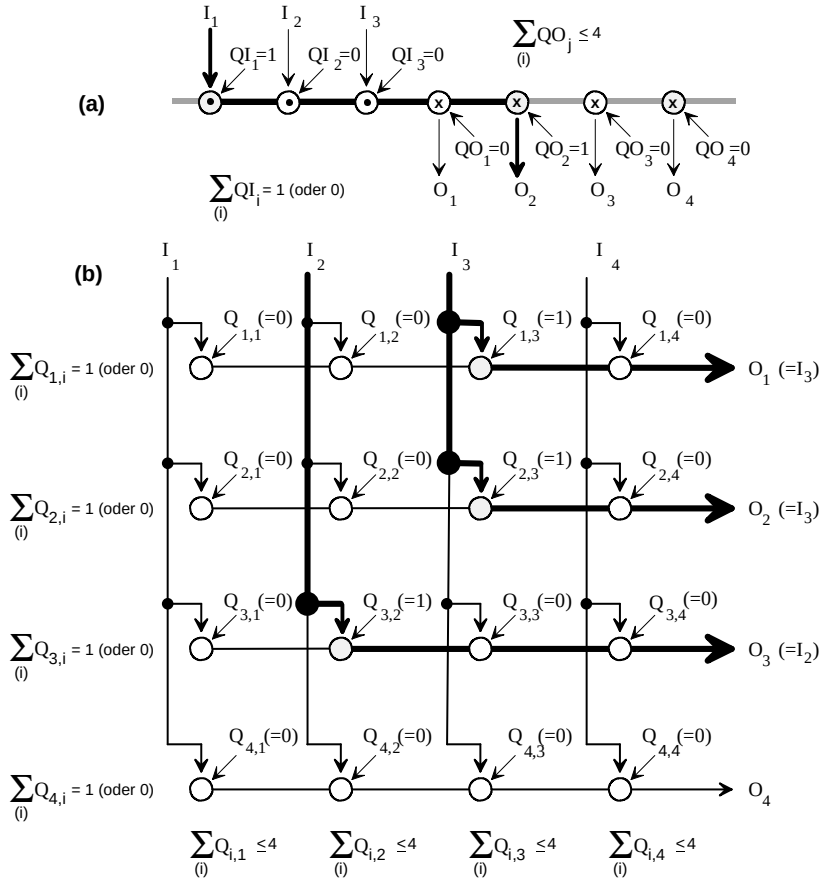
Abb. 27b/2 Matrixdarstellung für die Erreichbarkeit zwischen Quell- und Zielknoten in einem *Hypercube* gem. Bild 27a/2

Je dunkler der Grauton, desto ausgedehnter die topologische Weglänge. Über höchstens (D=)5 physische Kanäle hinweg erreicht jeder *Verkehrsknoten* jeden anderen *Verkehrsknoten*.

Vertreter *dynamischer Topologien*

sind
- *einstufige Netzwerke* (Abb. 28/2)*:*
  - *bus,*
  - *crossbar*

und
- *mehrstufige Netzwerke:*
  - *Banyan-Netzwerke,*
  - *Benes-Netzwerke*.



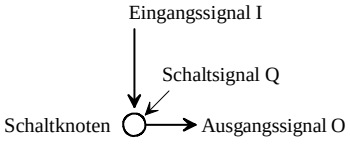


Abb. 28/2 Einstufige Netzwerke

(a) einkanaliges Bussystem (b) crossbar)

---

mit $\sum_{(i)} QI_i = 1\ (oder = 0)$ mit $\sum_{(j)} Q_{i,j} = 1\ (oder = 0) \quad \forall i$

und $\sum_{(j)} QO_j \leq 4$ und $\sum_{(i)} Q_{i,j} \leq 4 \quad \forall j$

als Nebenbedingungen

Komponenten der *dynamischen Topologien* sind schaltbare Netzelemente zur Kanalisierung von Nachrichtenströmen, in ihrem Zusammenwirken hierarchisch gegliedert in ein- und mehrstufige Netzwerke. Die Schaltfunktionen der Netzelemente reduzieren sich entweder auf die Durchschaltung / Blockierung von Nachrichtenströmen oder auf deren Umlenkung / Vervielfachung.

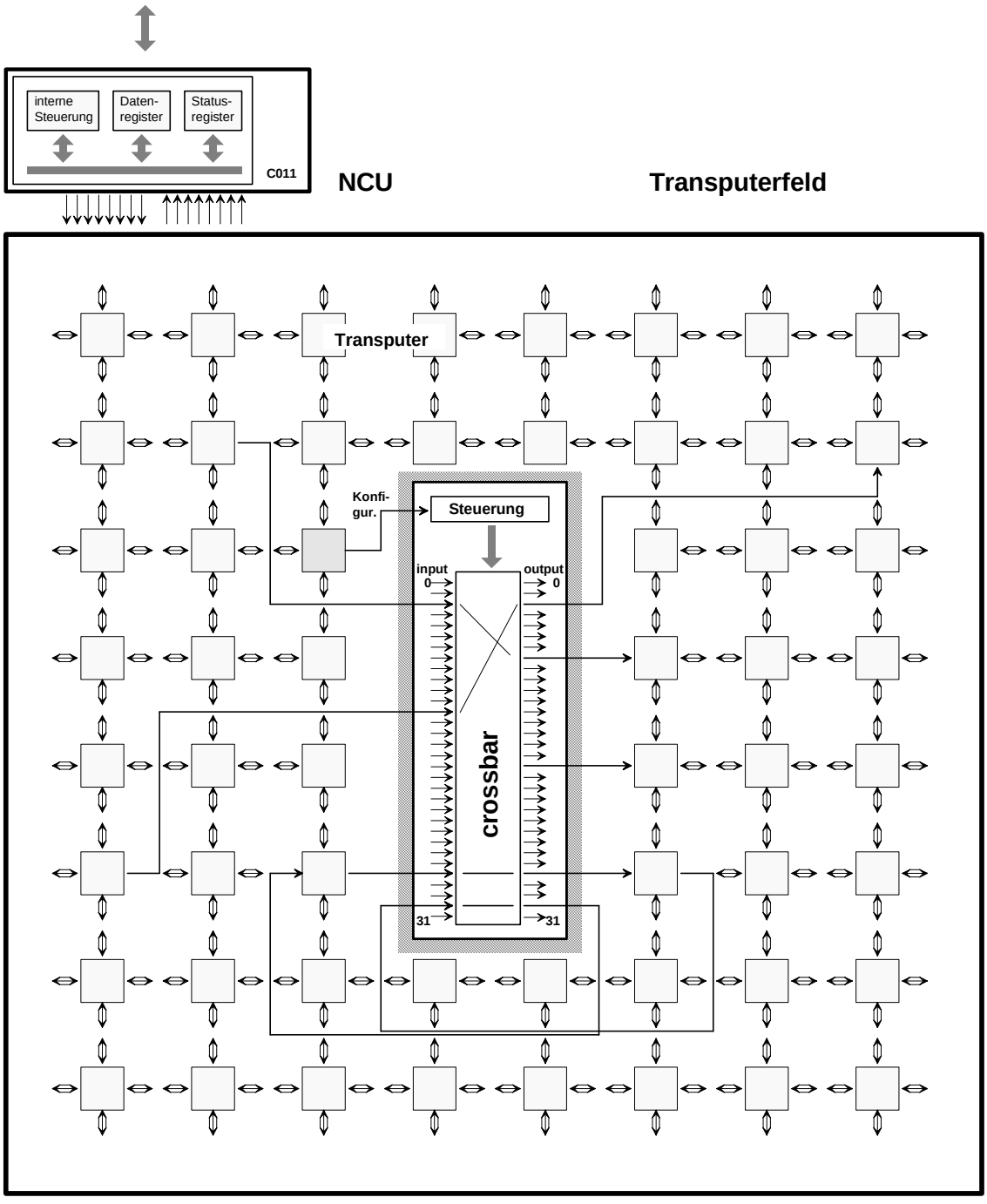


Abb. 29/2 Ereignisabhängige Topologiesteuerung in einem Transputerfeld. Aufgabe der *crossbar* ist es, ausgewählte Direktverbindungen zwischen räumlich weit entfernten Transputern herzustellen.

Der *Bus* und die *crossbar matrix* als Vertreter der einstufigen Netzwerke sind planar strukturiert und stellen über ein singuläres Schaltelement eine Nachrichtenverbindung von einem Sende- zu einem Empfangsknoten im *Verbindungsnetzwerk* her. Der Aufwand an Schaltelementen steigt überproportional zur Anzahl zu transferierender *Nachrichtenströme*. Diesem Trend stehen die mehrstufigen Netzwerke entgegen. Mehrstufige Netzwerke erstrecken sich über mehrere Ebenen und sind aus Netzelementen mit allen genannten Schaltfunktionen konfiguriert. Allen Netzwerken gemein sind die konkurrierenden Nachrichtenströme. Jedes Schaltelement kann jeweils nur einen Nachrichtenstrom kanalisieren, konkurrente Nachrichtenströme müssen zurückgestellt werden.

Mit wachsender Komplexität der *crossbar matrix* ergeben sich verbesserte Möglichkeiten, die Topologie dem Referenzverhalten der Prozessoren anzupassen, wächst andererseits aber auch die Notwendigkeit einer ereignisabhängigen Topologiesynthese. Abb. 29/2 zeigt die Verwendung einer *crossbar matrix* zur Verkopplung von Transputern in einem 8x8-Feld.

Literatur

/AL 92/ H-J.Appelrath, J. Ludewig: **Scriptum Informatik**
B.G.Teubner Stuttgart, 1992 (Kapitel 3)

/BS 90/ J. Blieberger, G.-H. Schildt, U. Schmid, S. Stöcker: **Informatik**.
Springer Verlag Wien New York, 1990

/EH 90/ H. Eichele : **Multiprozesorsysteme**.
B.G.Teubner Stuttgart 1990

/FL 94/ Th. Flik, H. Liebig: **Mikroprozessortechnik**.
Springer-Verlag, 1994

/GW 93/ Giloi, W.K.: **Rechnerarchitektur**.
Springer-Verlag 1993

/GL 90/ L. Goldschlager, A. Lister : **Informatik**.
Hanser Studienbücher, 1990

/WE 95/ D. Werner: **Taschenbuch der Informatik**.
Fachbuchverlag Leipzig, 1995 (Kapitel 2)

/WK 98/ K. Waldschmidt (Hrsg.) : **Parallelrechner**.
B.G.Teubner Stuttgart 1998