

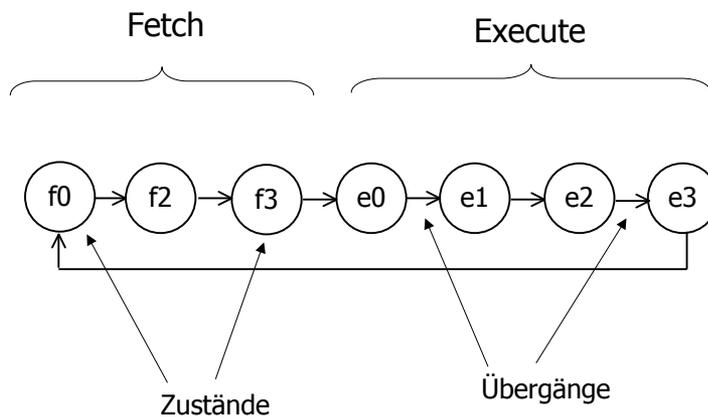
## 15.2.2 Zustandsdiagramme

Aufgaben:

- Erzeuge Uhrzeit in Abhängigkeit von Befehlen und /ack-Signal
- Erzeuge für jeden Speicher eine Kontrolleinheit

→ Zustandsdiagramme

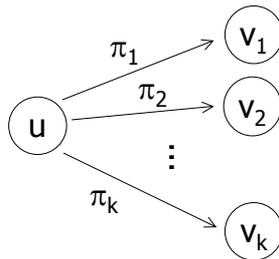
## Uhrzeit bisher:



## Allgemeinere Zustandsdiagramme:

- Knoten evtl. mehr als eine ausgehende Kante
- Übergang von Knoten  $u$  zu Knoten  $v_i$  in

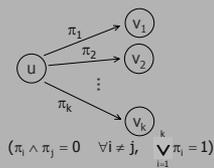
Abhängigkeit von Bedingung  $\pi_i$  :



$$(\pi_i \wedge \pi_j = 0 \quad \forall i \neq j, \quad \bigvee_{i=1}^k \pi_i = 1)$$

## Allgemeinere Zustandsdiagramme:

- Zustandsdiagramme können als Schaltung realisiert werden



- Wir gehen darauf näher ein

# Zustandsdiagramme

Darstellungsform für  
sogenannte „endliche Automaten“

## Exkurs zu endlichen Automaten und ihrer Realisierung

Wir werden damit unsere Kontrolllogik bauen

# Halbautomat

## Definition:

Das Quadrupel  $H = (I, S, S_0, \delta)$  heißt deterministischer,  
endlicher Halbautomat. Dabei bezeichnet:

$I$  eine endliche Menge von erlaubten *Eingabesymbolen*  
("Eingabealphabet")

$S$  eine endliche Menge von *Zuständen*

$S_0 \subseteq S$  ist eine endliche Menge von erlaubten  
*Anfangszuständen*

$\delta: S \times I \rightarrow S$  eine *Übergangsfunktion*.

# Halbautomat

Wir interessieren uns hier für Automaten  
mit Ausgaben

## Moore- und Mealy-Automaten

# Mealy-Automat

## Definition:

Eine *Mealy-Automat*  $M = (I, O, S, S_0, \delta, \lambda)$  ist ein  
endlicher, deterministischer Halbautomat  $H$  erweitert  
um:

- eine endliche Menge  $O$  von *Ausgabesymbolen*  
("Ausgabealphabet")
- eine Ausgabefunktion  $\lambda : S \times I \rightarrow O$ .

# Mealy-Automat

Beim Mealy-Automaten ist

- die Ausgabe abhängig vom aktuellen Zustand und der aktuellen Eingabe
- der Folgezustand abhängig vom aktuellen Zustand und der aktuellen Eingabe.

# Moore-Automat

## Definition:

Eine *Moore-Automat*  $M = (I, O, S, S_0, \delta, \lambda)$  ist ein endlicher, deterministischer Halbautomat  $H$  erweitert um:

- eine endliche Menge  $O$  von *Ausgabesymbolen* ("Ausgabealphabet")
- eine Ausgabefunktion  $\lambda : S \rightarrow O$ .

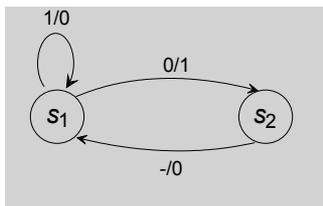
# Moore-Automat

- Ein Moore-Automat ist ein spezieller Mealy-Automat, bei dem die Ausgabe nur vom aktuellen Zustand und nicht von der Eingabe abhängt.
- Moore- und Mealy Automat kann man ineinander überführen.

# Darstellungen

x	state	next-state	y
1	s <sub>1</sub>	s <sub>1</sub>	0
0	s <sub>1</sub>	s <sub>2</sub>	1
-	s <sub>2</sub>	s <sub>1</sub>	0

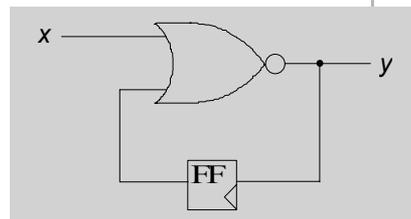
a) Darstellung als Zustandstafel



c) Darstellung als Zustandsdiagramm

	x = 0, y	x = 1, y
s <sub>1</sub>	s <sub>2</sub> , 1	s <sub>1</sub> , 0
s <sub>2</sub>	s <sub>1</sub> , 0	s <sub>1</sub> , 0

b) Darstellung als Flusstafel



d) Darstellung als synchrones Schaltwerk

# Darstellungen

## Wichtig für uns

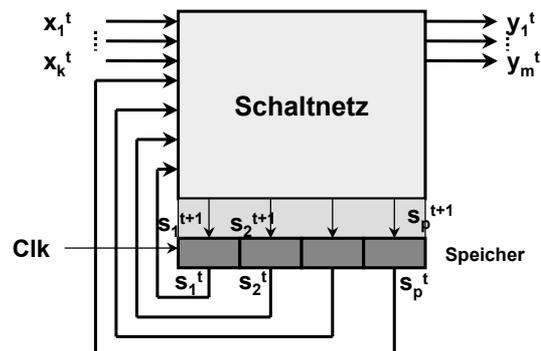
Darstellung als synchrones Schaltwerk

und der Weg

vom Zustandsdiagramm zum Schaltwerk

## Definition: Schaltwerke

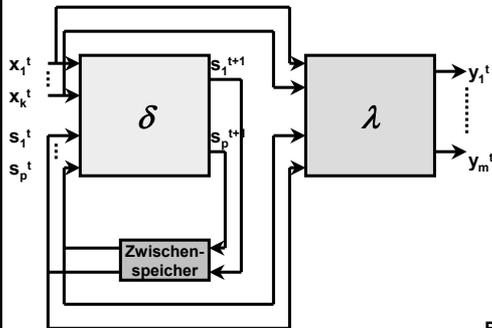
- Aufteilung eines synchronen Schaltwerks in Schaltnetz und speichernde Elemente:



$$y_i^t = f_i(x_1^t, x_2^t, \dots, x_k^t, s_1^t, s_2^t, \dots, s_p^t)$$

$$s_i^{t+1} = g_i(x_1^t, x_2^t, \dots, x_k^t, s_1^t, s_2^t, \dots, s_p^t)$$

## Darstellung eines Mealy-Automaten



Eingabevektor:  $\underline{X} = (x_1, x_2, \dots, x_k)$

Ausgabevektor:  $\underline{Y} = (y_1, y_2, \dots, y_m)$

Zustandsvektor:  $\underline{S} = (s_1, s_2, \dots, s_p)$

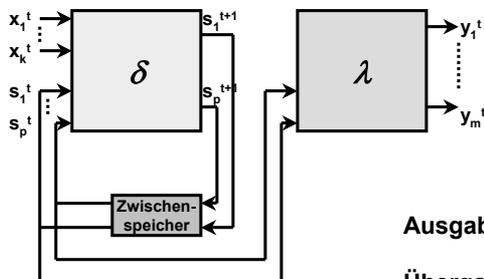
Ausgabefunktion:  $\underline{Y}^t = \lambda(\underline{X}^t, \underline{S}^t)$

Übergangsfunktion:  $\underline{S}^{t+1} = \delta(\underline{X}^t, \underline{S}^t)$

Beim Mealy-Automaten ist

- die Ausgabe abhängig vom aktuellen Zustand und der aktuellen Eingabe
- der Folgezustand abhängig vom aktuellen Zustand und der aktuellen Eingabe.

## Moore-Automat



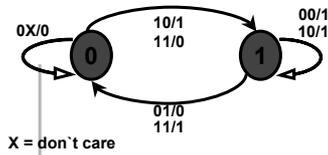
Ausgabefunktion:  $\underline{Y}^t = \lambda(\underline{S}^t)$

Übergangsfunktion:  $\underline{S}^{t+1} = \delta(\underline{X}^t, \underline{S}^t)$

- Ein Moore-Automat ist ein spezieller Mealy-Automat, bei dem die Ausgabe nur vom aktuellen Zustand und nicht von der Eingabe abhängt.

# Zustandsdiagramme und der Weg zum Schaltwerk

## Zustandsdiagramm:



- Knoten: Zustände des Schaltwerks
  - Kanten: Zustandsübergänge
  - Kantenmarkierung: Eingabe/Ausgabe (Mealy-Automat)
- Es wird immer genau *ein* Zustand angenommen.  
 Moore-Automat analog mit Ausgabe als Beschriftung der Zustände.  
 Aufgabe: Bestimme synchrones Schaltwerk mit dem angegebenen Zustandsübergangsgraphen.

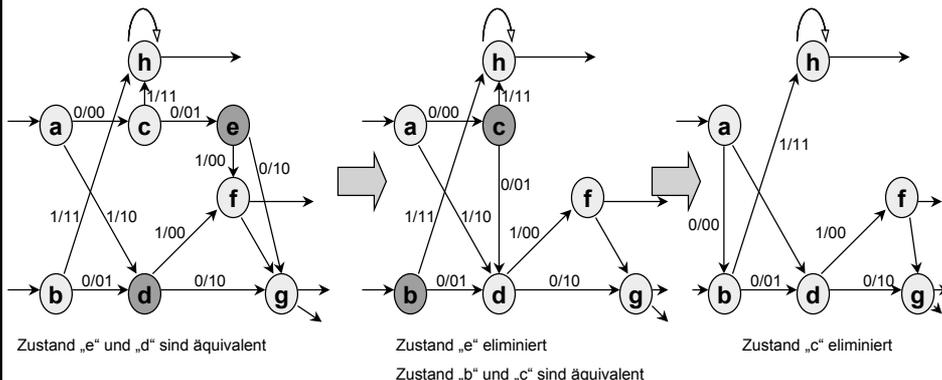
## Zustands- und Ausgangstabelle:

$s^t$	$x_1^t$	$x_2^t$	$s^{t+1}$	$y^t$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

# Einsparung redundanter Zustände

Zwei Zustände eines Schaltwerks heißen äquivalent, wenn sie bei gleichem Eingangsvektor stets den selben Ausgangsvektor erzeugen und einen äquivalenten Folgezustand annehmen.

Äquivalente Zustände können durch einen einzigen Zustand ersetzt werden.



# Schaltwerksentwurf: Beispiel

## 1. Aufgabenstellung:

Modulo-4 Vorwärts/Rückwärtszähler

Der Zähler soll von 0 bis 3 zählen können. Ist der Steuereingang  $x$  auf 1 gesetzt, so soll vorwärts gezählt werden, d.h. die Zahlenfolge 0,1,2,3 durchlaufen werden, ist  $x = 0$ , so soll rückwärts gezählt werden, d.h. die Zahlenfolge 3,2,1,0 durchlaufen werden. Am Ausgang ist der Zählerstand anzugeben (Ausgabevektor  $y_0, y_1$ ). Der Zähler ist als Ringzähler zu realisieren.

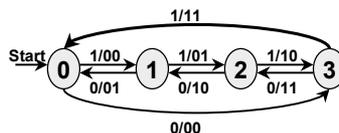
## 2. Zustandsdefinition:

Vier Zustände erforderlich: 0, 1, 2, 3

BB - TI II 15.2/19

# Schaltwerksentwurf: Beispiel

## 3. Zustandsgraph:



## 4. Eliminieren von Zustandsredundanzen

## 5. Zustandskodierung:

0 00, 1 ~ 01, 2 ~ 10, 3 ~ 11

BB - TI II 15.2/20

# Schaltwerksentwurf

## 6. Zustandstabelle:

	x	$z_1^t$	$z_0^t$	$z_1^{t+1}$	$z_0^{t+1}$	$y_1$	$y_0$
Vorwärtszählen	1	0	0	0	1	0	0
	1	0	1	1	0	0	1
	1	1	0	1	1	1	0
	1	1	1	0	0	1	1
Rückwärtszählen	0	1	1	1	0	1	1
	0	1	0	0	1	1	0
	0	0	1	0	0	0	1
	0	0	0	1	1	0	0

# Schaltwerksentwurf

## 7. Boolesche Ausdrücke für Ausgangs- und Übergangsfunktion:

Ausgangsfunktion:  $y_0^t = z_0^t, y_1^t = z_1^t$

Übergangsfunktion:  $z_0^{t+1} = xz_1^t \bar{z}_0^t + xz_1^t z_0^t + \bar{x}z_1^t \bar{z}_0^t + \bar{x}z_1^t z_0^t$

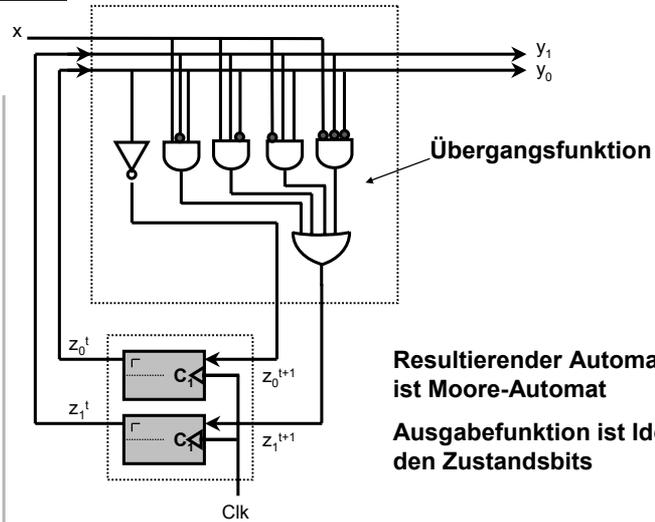
$$z_1^{t+1} = xz_1^t \bar{z}_0^t + xz_1^t z_0^t + \bar{x}z_1^t \bar{z}_0^t + \bar{x}z_1^t z_0^t$$

Minimierung:  $z_0^{t+1} = \bar{z}_0^t$

$$z_1^{t+1} = xz_1^t \bar{z}_0^t + xz_1^t z_0^t + \bar{x}z_1^t \bar{z}_0^t + \bar{x}z_1^t z_0^t$$

# Schaltwerksentwurf

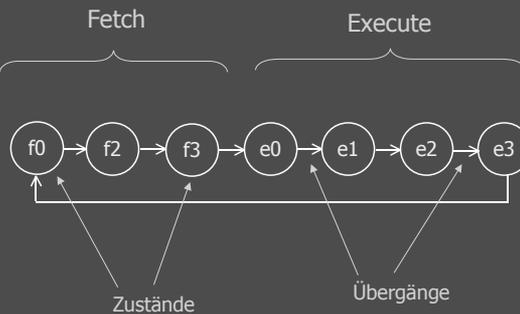
Schaltbild:



BB - TI II 15.2/23

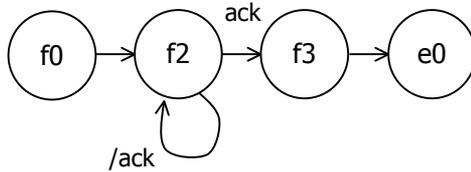
# Zurück zu unserem Rechner ReTi II

Uhrzeit bisher:

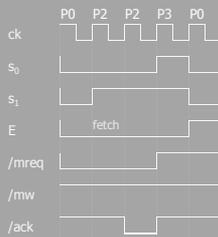


BB - TI II 15.2/41

# Zustandsdiagramm für Uhrzeiten: Fetch



## Wait-Zyklen für Fetch



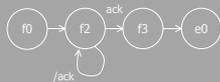
## Kodierung der Zustände für Fetch

Zustand	E	s <sub>1</sub>	s <sub>0</sub>
f0	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

# Spezifikation der Übergänge

- $E := /E * s_1 * s_0$  ; Übergang von f3 nach e0
- $s_1 := /E * /s_1 * /s_0$  ; Übergang von f0 nach f2  
     +  $/E * s_1 * /s_0$  ; Übergang ausgehend von f2
- $s_0 := /E * s_1 * /s_0 * ack$  ; Übergang von f2 nach f3

## Zustandsdiagramm für Uhrzeiten: Fetch



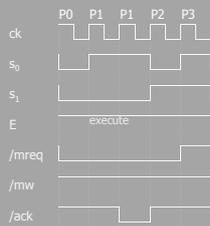
## Kodierung der Zustände für Fetch

Zustand	E	s <sub>1</sub>	s <sub>0</sub>
f0	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

## Zustandsdiagramm für Uhrzeiten: Execute

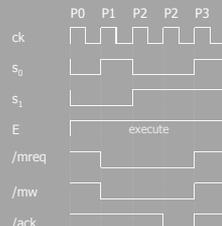
- Wir unterscheiden zwischen Compute memory – Befehlen, Load/Store – Befehlen (außer LOADI, MOVE) und CPU-internen Befehlen
- Dementsprechend gibt es drei Bedingungen cm, ls und in, die die Übergänge bestimmen.

Wait-Zyklen für Compute memory



Wait in P1

Wait-Zyklen für Store

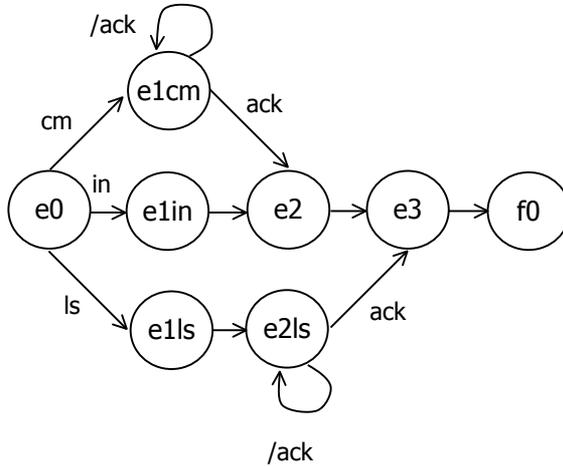


Wait in P2

## Erklärung zu den Zustandsdiagrammen

- $cm = 1 \Leftrightarrow$  Es handelt sich um  
Compute memory – Befehle (waits in P1)
- $ls = 1 \Leftrightarrow$  Es handelt sich um  
Load/Store – Befehle (außer LOADI, MOVE)  
(waits in P2)
- $in = 1 \Leftrightarrow$  Es handelt sich um  
CPU-internen Befehl (keine waits !)

# Zustandsdiagramm für Uhrzeiten: Execute



## Erklärung zu den Zu...

Bestimme cm, ls, in  
folgendermaßen

■ cm =  $/I31 * /I30 * I29$

■ in = 
$$\begin{aligned} & /I31 * /I30 * /I29 \\ & + /I31 * I30 * I29 * I28 \\ & + I31 * /I30 * I29 * I28 \\ & + I31 * I30 \end{aligned}$$

Jump →

■ ls = 
$$\begin{aligned} & /I31 * I30 * /I29 \\ & + /I31 * I30 * /I28 \\ & + I31 * /I30 * /I29 \\ & + I31 * /I30 * /I28 \end{aligned}$$

### Compute - Befehle (ff)

Typ	MI	F	Befehl	Wirkung
0 0	0	0 1 0	SUBI i	[ACC]:=[ACC] - [i]    <PC> := <PC> + 1
		0 1 1	ADDI i	[ACC]:=[ACC] + [i]    <PC> := <PC> + 1
		1 0 0	OPLUSI i	ACC := ACC ⊕ 0 <sup>8</sup> i    <PC> := <PC> + 1
		1 0 1	ORI i	ACC := ACC ∨ 0 <sup>8</sup> i    <PC> := <PC> + 1
		1 1 0	ANDI i	ACC := ACC ∧ 0 <sup>8</sup> i    <PC> := <PC> + 1
0 0	1	0 1 0	SUB i	[ACC]:=[ACC] - [M(<i>)]    <PC> := <PC> + 1
		0 1 1	ADD i	[ACC]:=[ACC] + [M(<i>)]    <PC> := <PC> + 1
		1 0 0	OPLUS i	ACC := ACC ⊕ M(<i>)    <PC> := <PC> + 1
		1 0 1	OR i	ACC := ACC ∨ M(<i>)    <PC> := <PC> + 1
1 1 0	AND i	ACC := ACC ∧ M(<i>)    <PC> := <PC> + 1		

0 1	1 0	0	LOADINZ i	$M(\langle i \rangle + [i])$	$\langle PC \rangle := \langle PC \rangle + 1$
0 1	1 1	0	LOADI i	$ACC := 0^8 i$	$\langle PC \rangle := \langle PC \rangle + 1$

### Store, Move - Befehle (ff)

Typ	Modus	Befehl	Wirkung
1 0	0 0	STORE i	$M(\langle i \rangle) := ACC$ <PC> := <PC> + 1
1 0	0 1	STOREIN1 i	$M(\langle IN1 \rangle + [i]) := ACC$ <PC> := <PC> + 1
1 0	1 0	STOREIN2 i	$M(\langle IN2 \rangle + [i]) := ACC$ <PC> := <PC> + 1
1 0	1 1	MOVE S D	D := S    <PC> := <PC> + 1

außer bei D = 0 0 (PC)

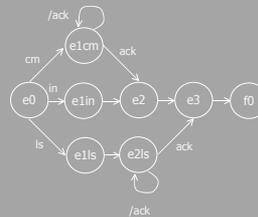
# Erklärung zu den Zustandsdiagrammen

Nach Binärcodierung der Zustände kann Zustandsdiagramm ( = endlicher Automat ) z.B. durch Register-PAL realisiert werden.

e1 zerlegt  
in 3 Zustände e1cm, e1in, e1ls

e2 aufgeteilt in 2 Zustände  
e2, e2ls

Zustandsdiagramm für Uhrzeiten:  
Execute



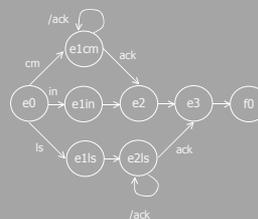
# Zustandskodierung

Erweiterung der Kodierung vom Fetch-Diagramm,  
zwei neue Signale r1, r0  
zur Unterscheidung der e1 und e2 Teil-Zustände

Kodierung der Zustände für  
Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

Zustandsdiagramm für Uhrzeiten:  
Execute



# Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

## Realisierung durch PAL-Gleichungen

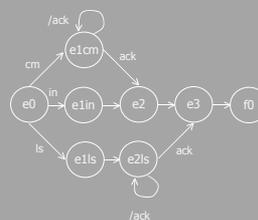
Hier nur einige Kommentare:  
r0 ergibt sich durch Ersetzen  
von in und Ausmultiplizieren  
bei

$$\begin{aligned}
 r0 := & E * /s1 * /s0 * in \\
 + & E * /s1 * s0 * r1 * /r0 \\
 + & E * s1 * /s0 * r0 * /ack
 \end{aligned}$$

### Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

### Zustandsdiagramm für Uhrzeiten: Execute



# Realisierung durch PAL-Gleichungen

Bei r1 ersetze ls

$$r1 := E * /s1 * /s0 * ls$$

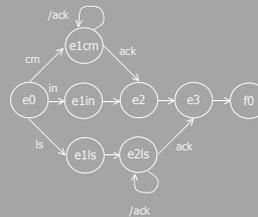
E wird aktiviert nach f3 und gehalten bis e3

s1 wird aktiviert nach f0, e1cm mit ack, e1in und e1ls; gehalten nach f2, e2 und e2ls

## Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

## Zustandsdiagramm für Uhrzeiten: Execute



BB - 118 | 15:248

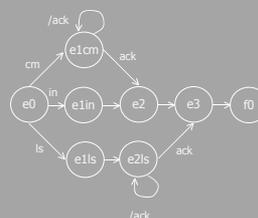
# Realisierung durch PAL-Gleichungen

s0 wird aktiviert nach f2 mit ack, nach e2ls mit ack und nach e2; gehalten nach e1cm mit /ack

## Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

## Zustandsdiagramm für Uhrzeiten: Execute



BB - 118 | 15:248

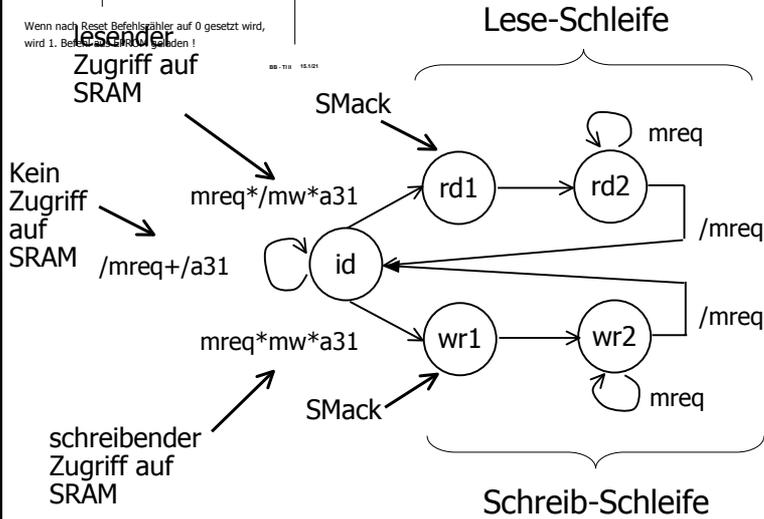
### Memory Map

Festlegung, welcher Speicher unter welcher Adresse angesprochen wird durch Memory Map.

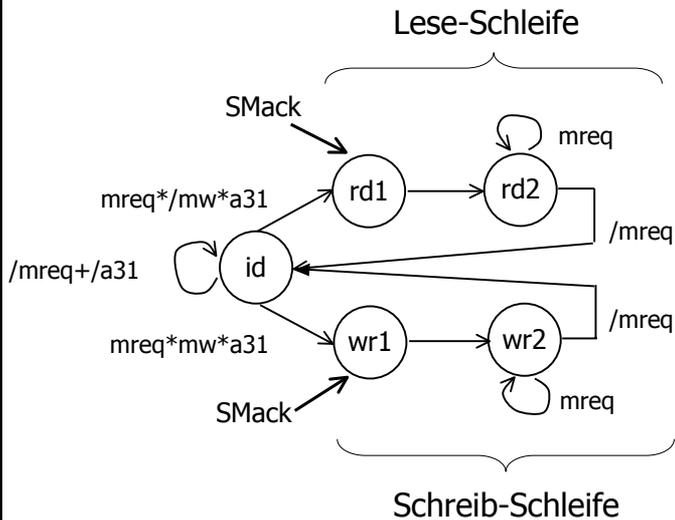
A[31:30]	Einheit
1 *	SRAM
0 1	UART
0 0	EPROM

Wenn nach Reset Befehlszähler auf 0 gesetzt wird, wird 1. Befehl aus SRAM geladen!

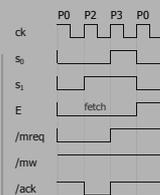
## Programm für Speicher:



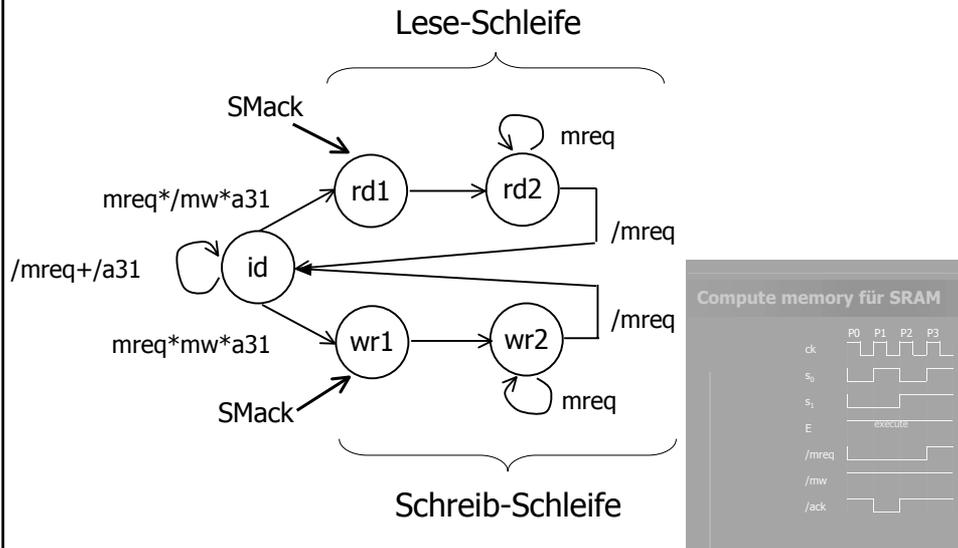
## Korrespondenz zu Fetch auf CPU-Seite



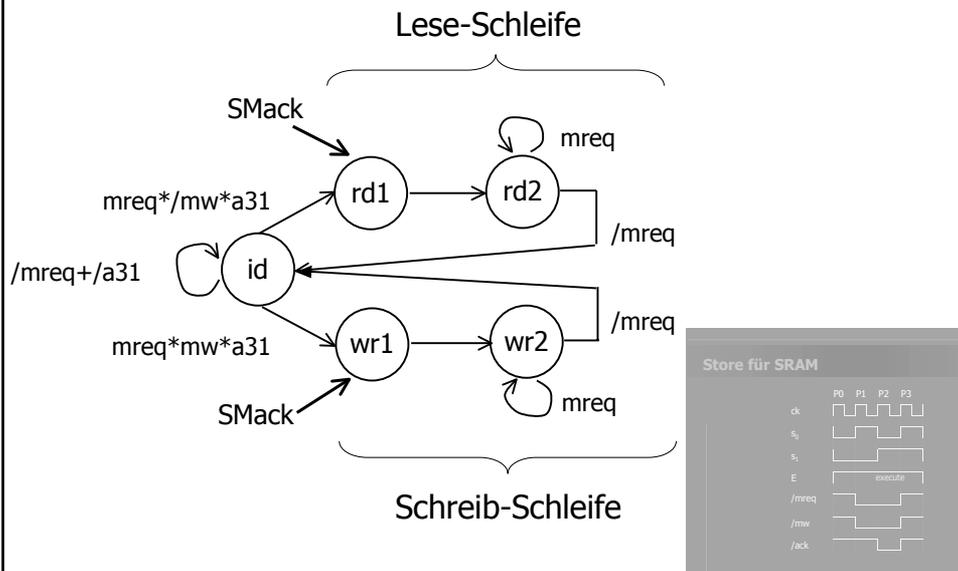
### Fetch für SRAM



# Korrespondenz zu Compute memory auf CPU-Seite



# Korrespondenz zu Store auf CPU-Seite



## Kontrollsignale aufgrund der Speicherkontrolle

SMDdœ, SMw, SMack werden aufgrund der Zustände der Speicherkontrolle berechnet:

SMack aktiviert in rd1, wr1 für einen Takt

SMDdœ aktiviert in rd1, rd2, disabled im Zustand nach rd2

SMw aktiviert in wr1 für einen Takt

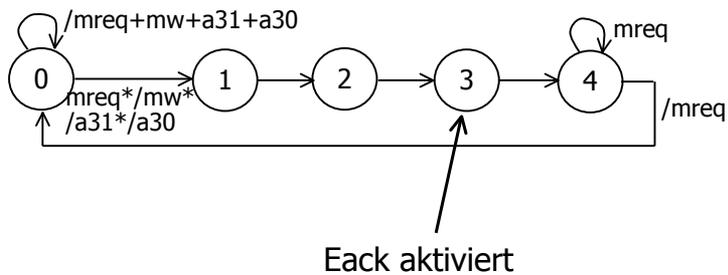
## Realisierung des Automaten und der Kontrollsignale

Benutze P-PALs und 3 Signale sr0, sr1, sr2 zur Kodierung der 5 Zustände

→ einfache Übung



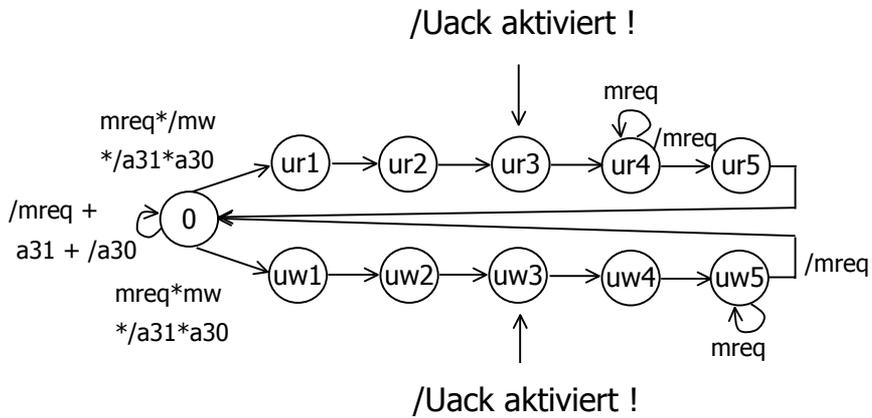
## Zustandsdiagramm für Speicher: EPROM



## EPROM

wie SRAM, aber nur Lesen und 2 Wait-Zyklen,  
da Eack erst in Zustand (3) aktiviert wird  
(Eack aktiv von (1) bis (4) )

## Zustandsdiagramm für Speicher: UART



## UART

$/Uack$  aktiviert in ur3 bzw. uw4

→ bei Lesen 2 Wait-Zyklen,  
bei Schreiben 3 Wait-Zyklen

(ur5 wird gebraucht, um Lesezykluszeit zu garantieren)