technische universität
dortmund

fakultät für informatik
informatik 12

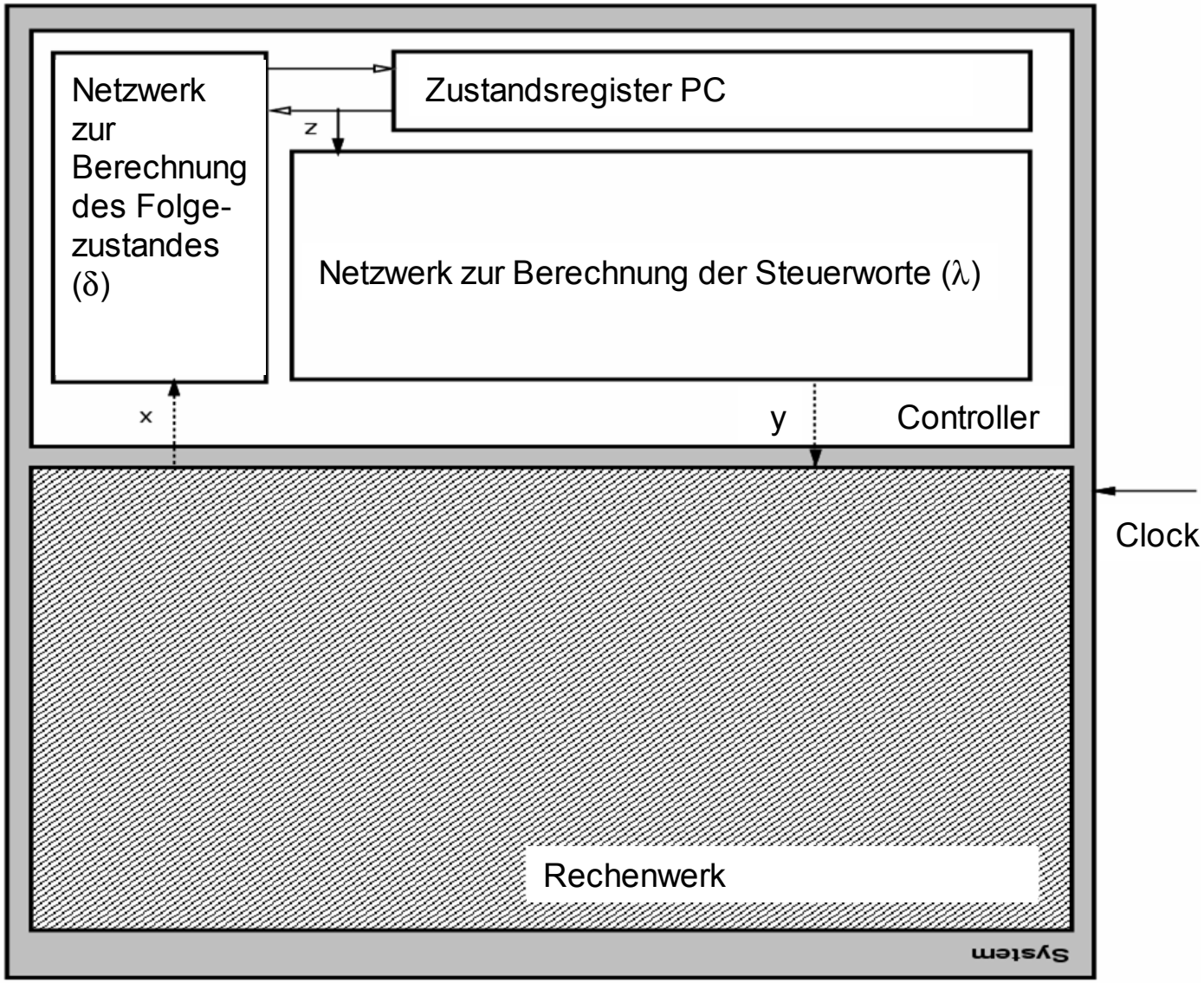
Controllersynthese

Peter Marwedel
Informatik 12

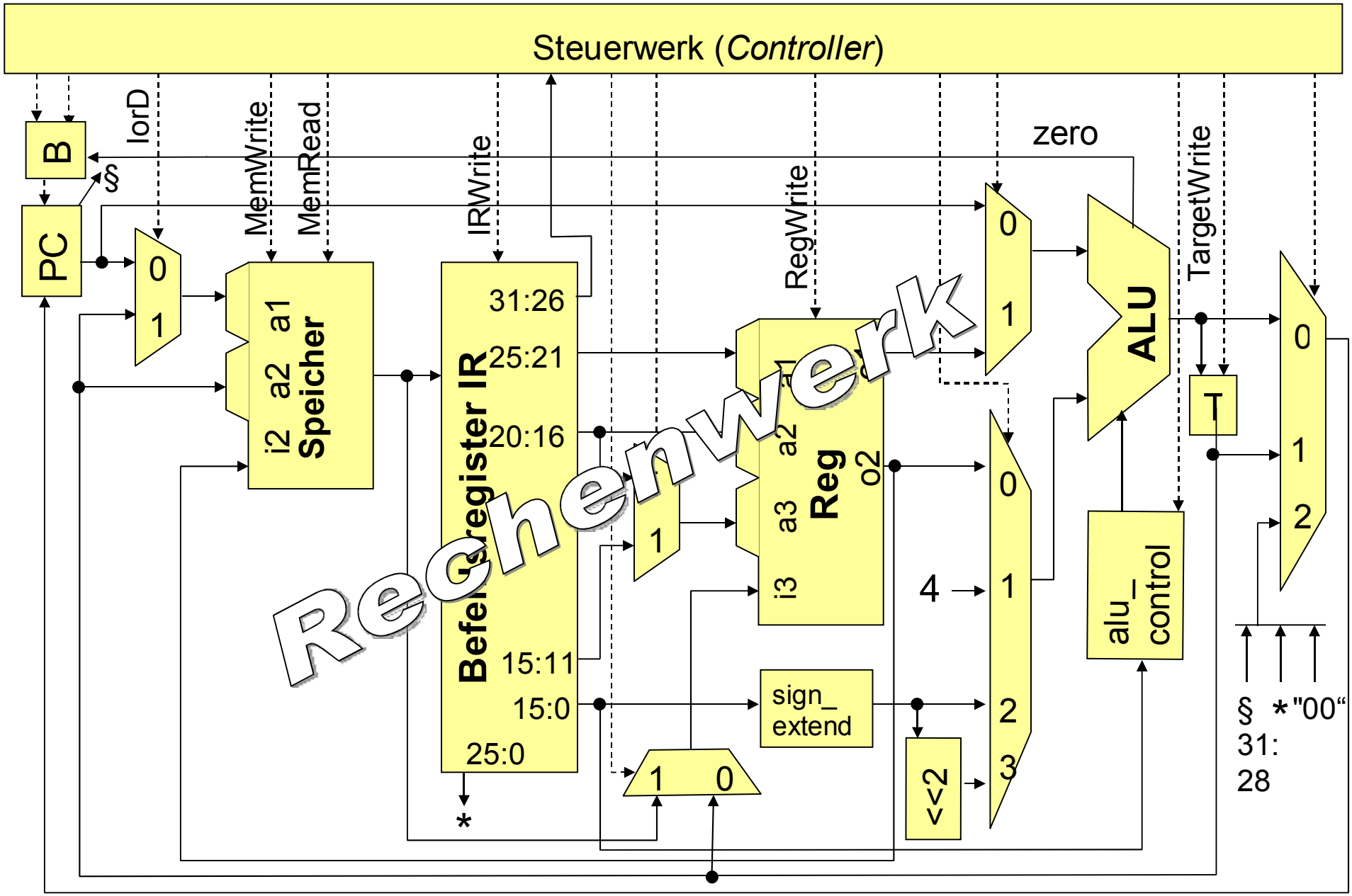
Aufteilung in Rechenwerk und Controller

Die Bausteine eines Rechenwerks müssen in jedem Kontrollschritt kontrolliert werden.
☞ Aufgabe der Controller-synthese.

Asynchrone Rückkopplung wenn beides Mealy-Automaten sind.

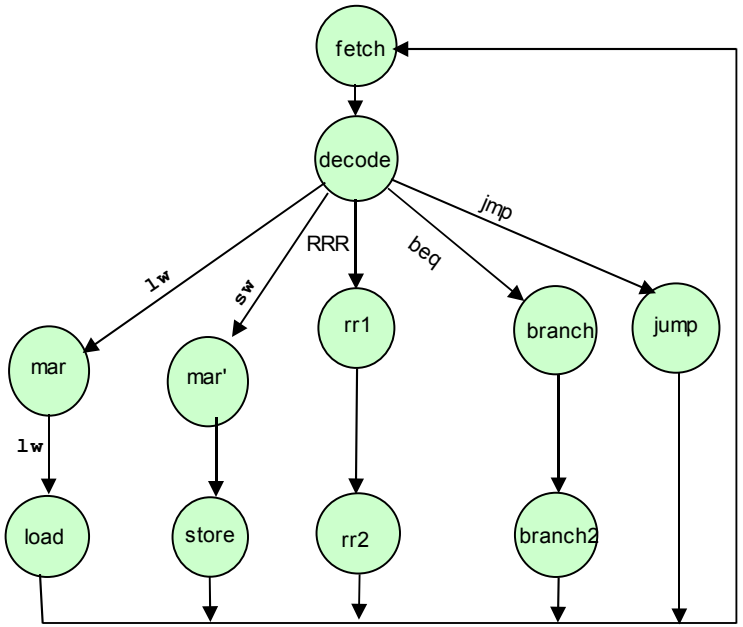


Beispiel: MIPS µArchitektur gemäß RS

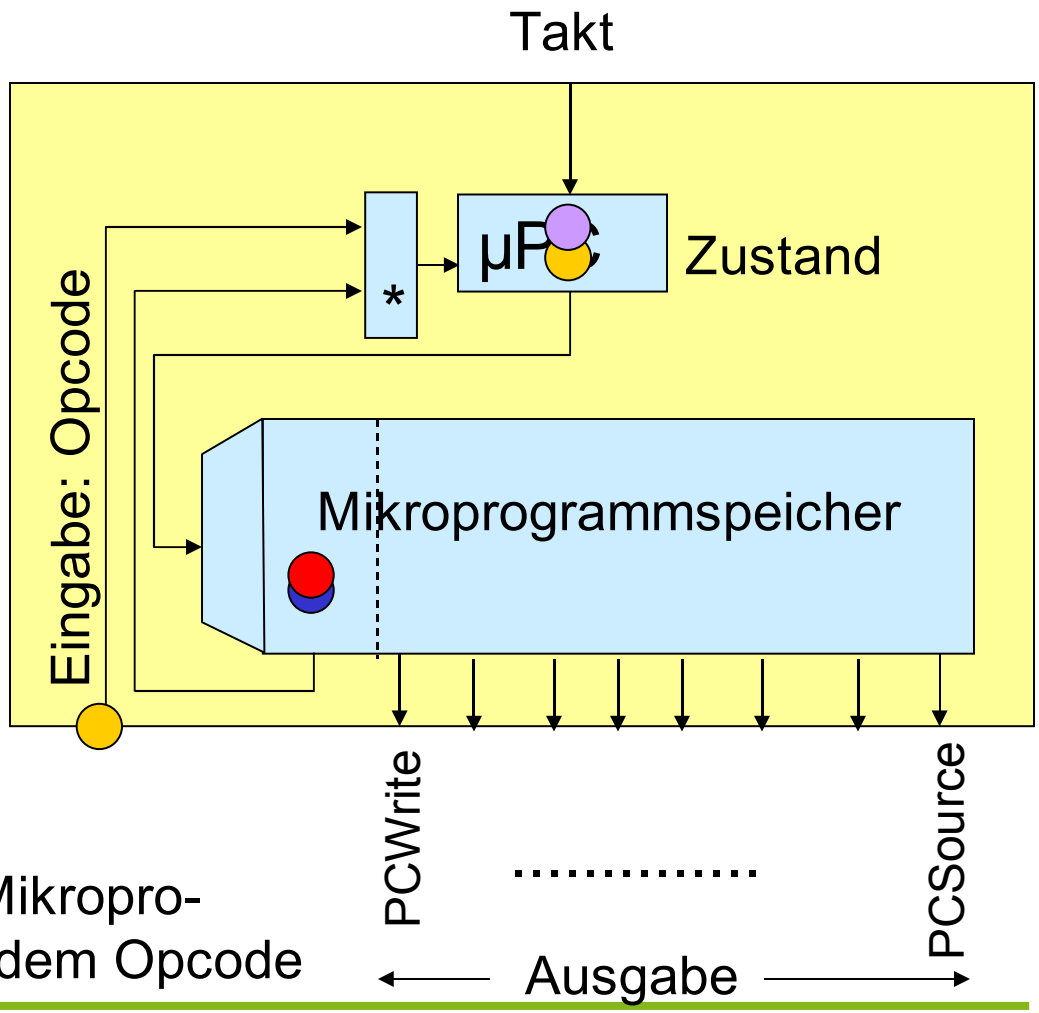


Das Steuerwerk (Moore-Automat)

Verhalten, vereinfacht



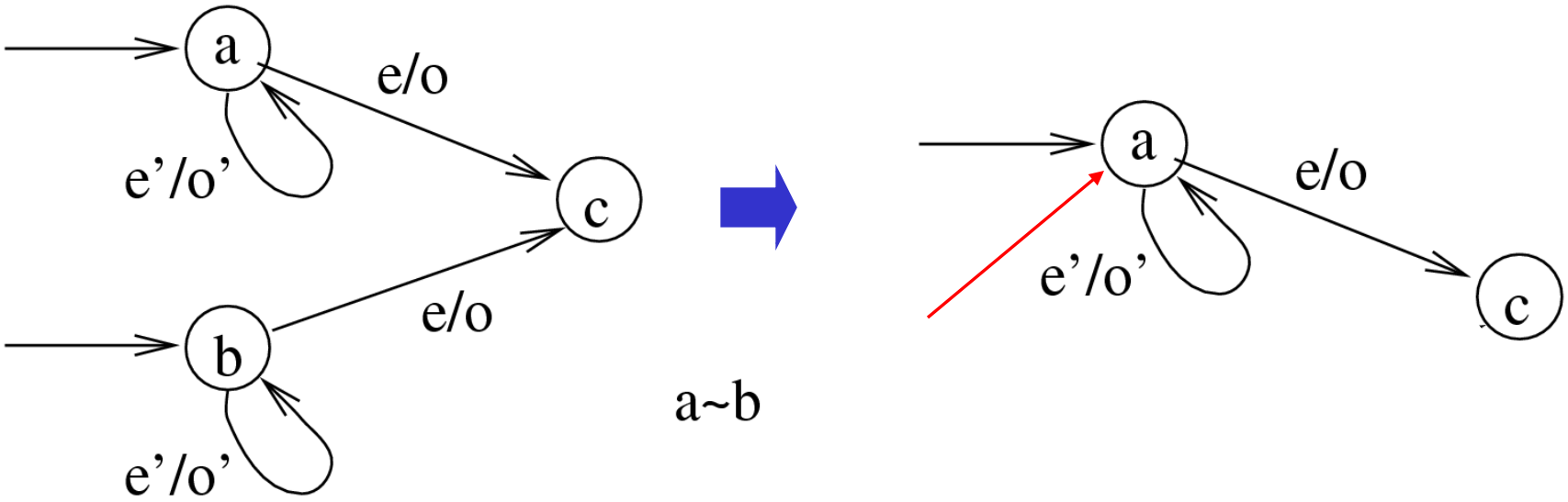
Struktur



* Folgezustand bestimmt durch Mikroprogrammspeicher, bei decode aus dem Opcode

Zustandreduktion

Idee: Zustände sind äquivalent, wenn sie für alle Eingaben dieselben Ausgaben erzeugen und dieselben Nachfolger haben: Äquivalente Zustände können durch einen Zustand ersetzt werden. Beispiel:



Theorie äquivalenter Zustände bekannt

Erinnerung: GTI; hier: Sieling, SS 2005

Minimierung

1. Schritt: Überflüssige Zustände streichen

2. Schritt: Äquivalenzklassenautomaten berechnen.

Fragen:

- Wie berechnet man $\equiv$?
- Ist der resultierende Automat wirklich minimal?

391

Erinnerung: GTI; hier: Sieling, SS 2005

Idee: $\not\equiv$ statt $\equiv$ berechnen

Es ist $p \not\equiv q \Leftrightarrow \exists w: \delta(p,w) \in F \wedge \delta(q,w) \notin F$ oder
$\delta(p,w) \notin F \wedge \delta(q,w) \in F$

Zeuge für die Nichtäquivalenz von p und q

1. Idee: Suche nichtäquivalente Paare von Zuständen, wobei mögliche Zeugen mit wachsender Länge probiert werden.

392

Erinnerung: GTI; hier: Sieling, SS 2005

Idee eines Algorithmus

Eingabe: DFA ohne überflüssige Zustände mit Q={1,...,n}.

Initialisierung: Alle Paare $\{i,j\}$ (mit $i<j$) sind unmakiert (d.h., $i \not\equiv j$ noch nicht erkannt).

- Markiere alle Paare $\{i,j\}$, bei denen ε Zeuge ist (d.h., $i \in F$, $j \notin F$ oder $i \notin F$, $j \in F$)

Problem: Vermeide, alle möglichen Zeugen der Längen 1,2,3,...,?? auszuprobieren.

393

Erinnerung: GTI; hier: Sieling, SS 2005

Idee:

Bearbeitung von $\{i,j\}$ (Reihenfolge beliebig):

Für alle $a \in \Sigma$:

Falls es ein a gibt, so dass $(\delta(i,a),\delta(j,a))$ markiert, folgt $i \not\equiv j$; also $\{i,j\}$ markieren.

Sonst: Für alle a speichere zu $\{\delta(i,a),\delta(j,a)\}$, dass bei deren Nichtäquivalenz auch $\{i,j\}$ zu markieren ist.

Datenstruktur: Zu jedem Paar $\{i,j\}$ Liste $L\{i,j\}$.

Wenn $\{i,j\}$ markiert wird, werden auch die Elemente von $L\{i,j\}$ markiert.

394

Hier: Betrachtung der Ausgabe

Two FSMs are equivalent if and only if every input sequence yields identical output sequences

Goal: Given an FSM, find the simplest equivalent FSM with a minimum number of states

- Two states s_1 and s_2 in an FSM are equivalent if and only if each input sequence beginning from s_1 yields an output sequence identical to that obtained by starting from s_2

© S. Devadas (MIT): http://csg.csail.mit.edu/u/d/devadas/public_html/6.373/lectures/l08/P011.html

FSM Minimierung

Look at each pair of states (s_1, s_2) in the FSM

If s_1 produces different outputs from s_2, for any input, mark them non-equivalent

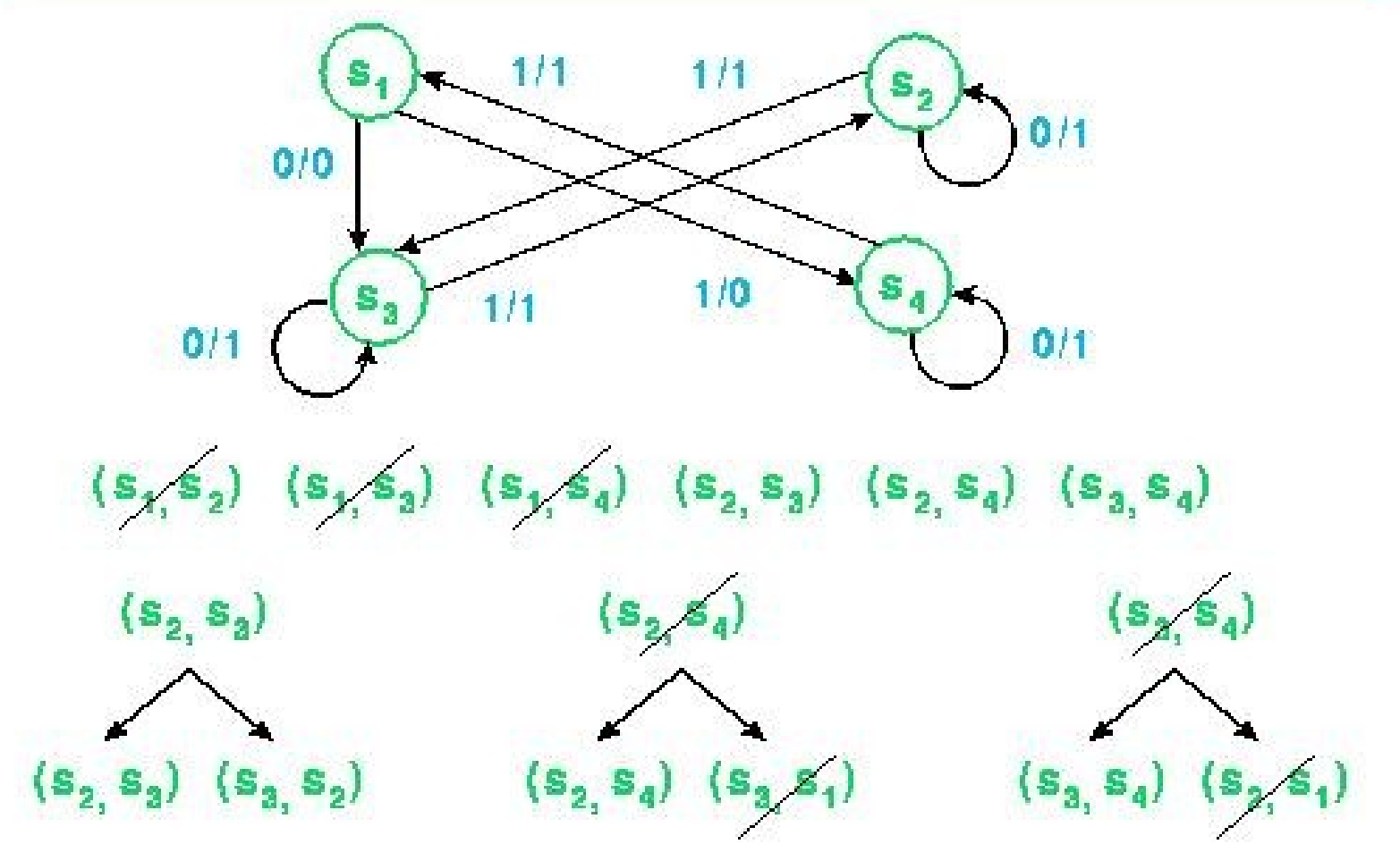
For each state pair (s_1, s_2) not yet marked, for each input i, find state pair $(T(s_1, i), T(s_2, i))$

If $(T(s_1, i), T(s_2, i))$ are marked non-equivalent for any i, mark (s_1, s_2) non-equivalent

Iterate until no more marking is possible.
Unmarked state pairs are equivalent, simplify FSM accordingly

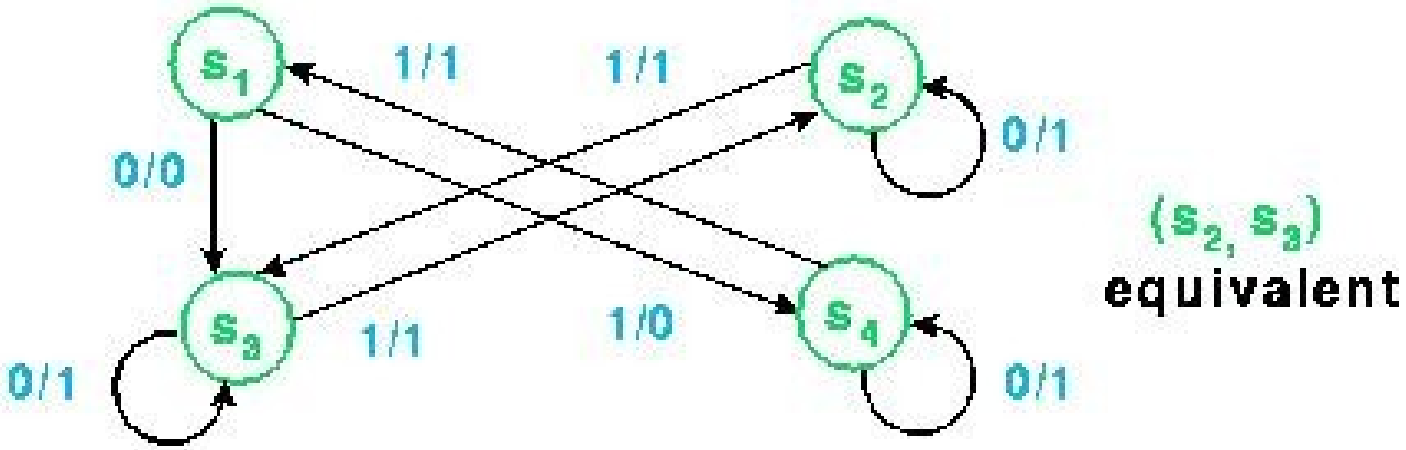
© S. Devadas (MIT): http://csg.csail.mit.edu/u/d/devadas/public_html/6.373/lectures/l08/P011.html

Beispiel-Minimierung

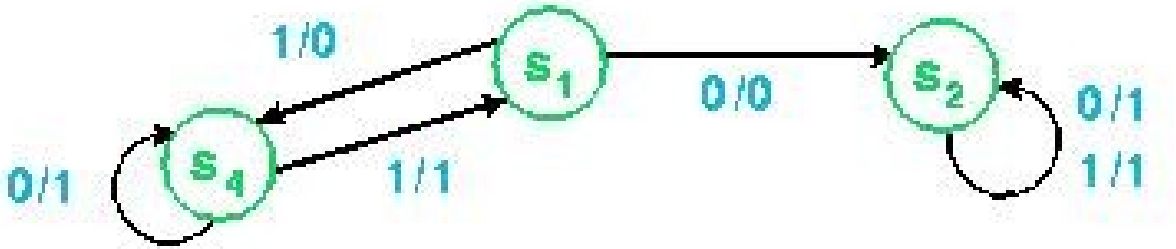


© S. Devadas (MIT): http://csg.csail.mit.edu/u/d/devadas/public_html/6.373/lectures/l08/P011.html

Vereinfachter Automat



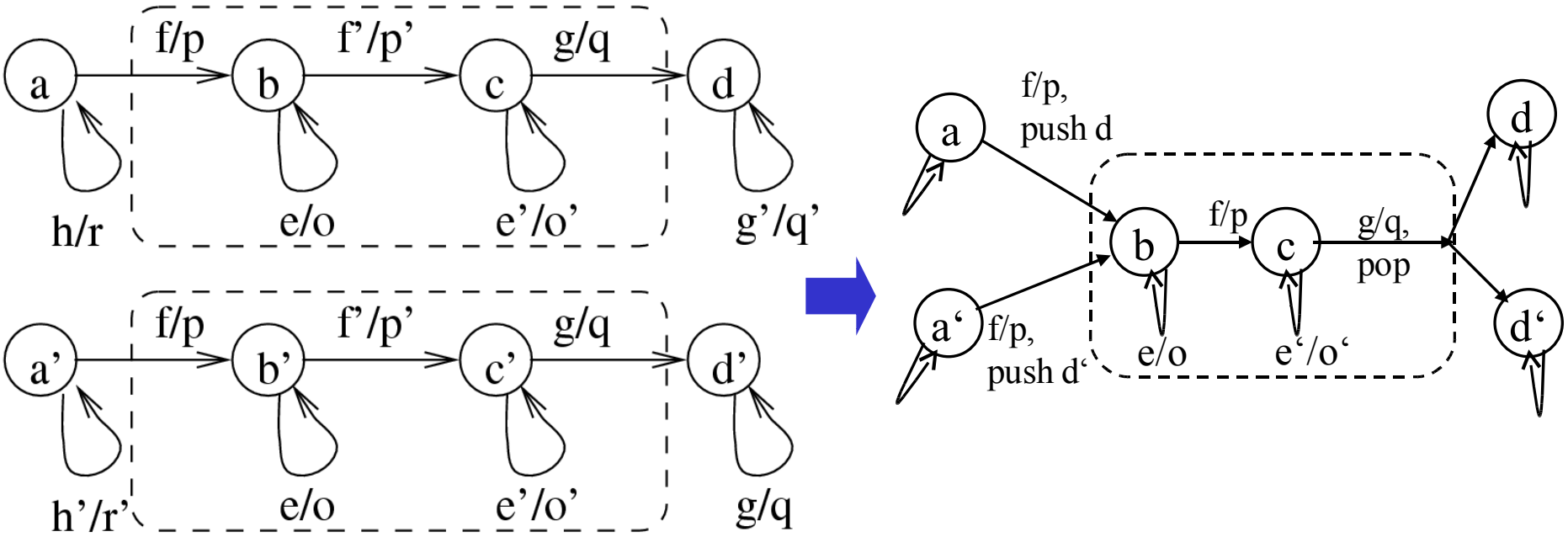
Pick s_2 and move incident edges of s_3 to s_2
Delete s_3 and any outgoing edges



© S. Devadas (MIT): http://csg.csail.mit.edu/u/d/devadas/public_html/6.373/lectures/l08/P011.html

Erweiterung: Controller mit einem *Stack*

Stapel (*stack*) erlaubt parameterlose Unterprogramme

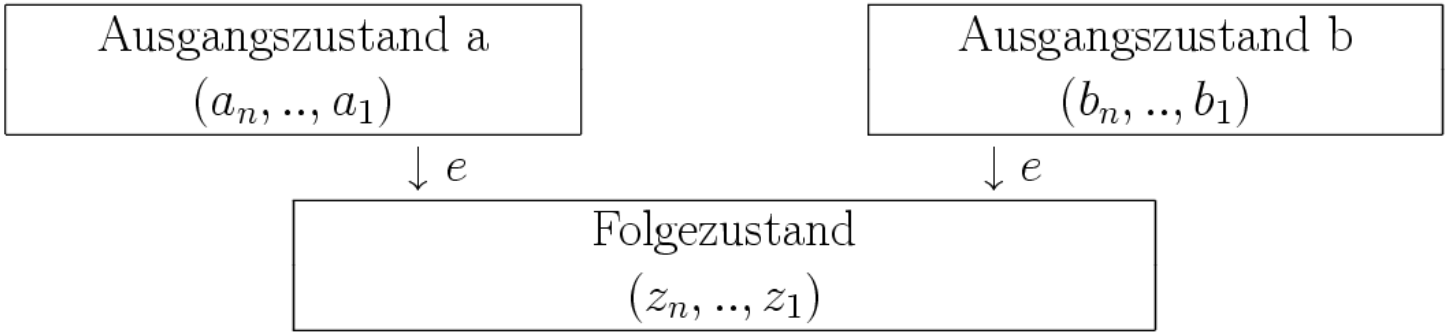


Zustandskodierung

- Aufwand für die Implementierung von λ und δ hängt von der Kodierung ab.
- Regelbasierte Kodierungsmethode in ASYL [Saucier et al.]:
  3 Klassen von Regeln:

1. *Join*-Regeln (1)

Betrachte Fälle, in denen für Eingabe *e* Übergänge zu demselben Folgezustand erfolgen.



Sei *i* Bit des Folgezustands mit z_i =1.

Um das Bit bei Eingabe von *e* zu setzen, wird der Ausdruck

$$z_i = e\,(a_n \ldots a_2\, a_1 \vee b_n \ldots b_2\, b_1)$$

benötigt.

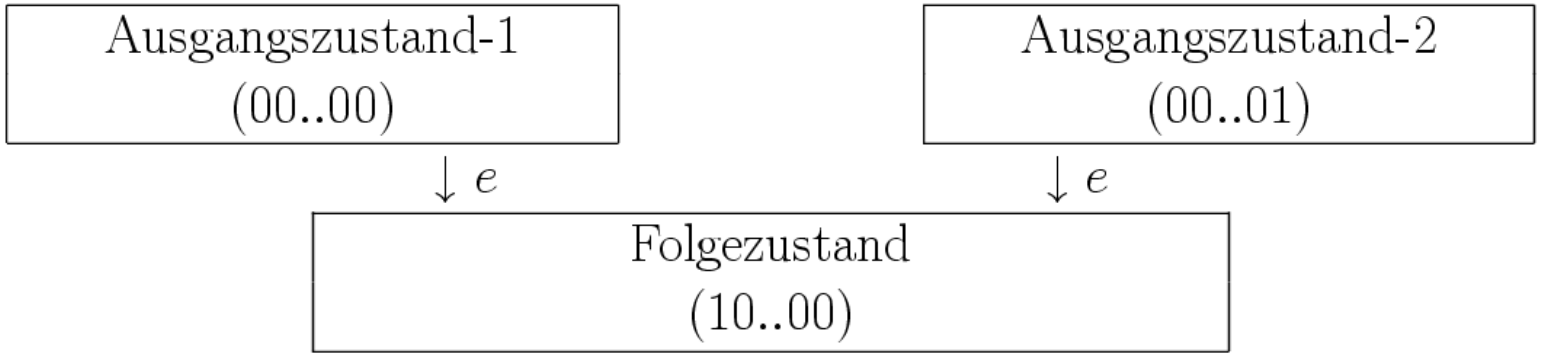
Join-Regeln (2)

Um das Bit bei Eingabe von *e* zu setzen, wird der Ausdruck

$$z_i = e\ (a_n \ldots a_2\ a_1 \vee b_n \ldots b_2\ b_1)$$

benötigt.

Sei der Hamming-Abstand der Kodierung der Ausgangszustände =1, also z.B:

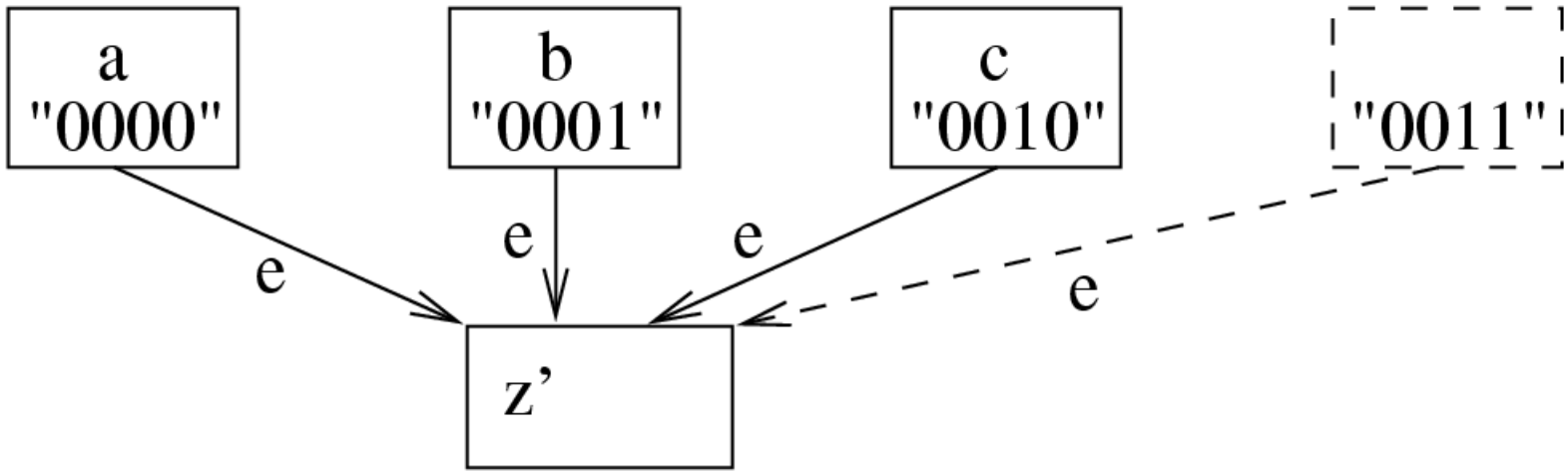


Dann berechnen sich die Zustandsbits des Folgezustands mit einem einzigen Term, also z.B:

$$z_i = e\ (a_n \ldots a_2).$$

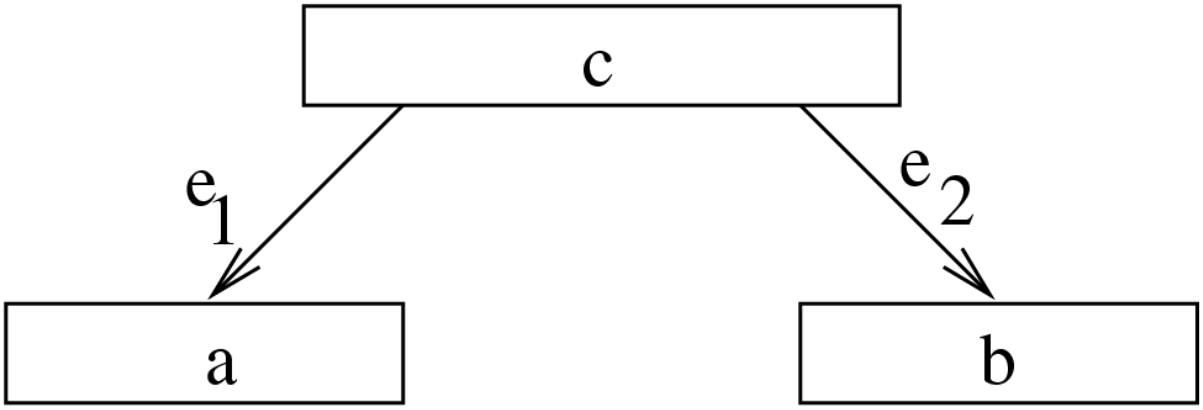
Join-Regeln: >2 Ausgangszustände

Im Allgemeinen lassen sich bei *m* Ausgangszuständen max. *m*-1 Produktterme einsparen.
Sofern *m* keine 2-er Potenz: 2^k-*m* mit $k = \lceil \log_2(m) \rceil$ nicht ausgenutzte Codes von einer weiteren Verwendung ausschließen.



2. *Fork*-Regeln

Analog zu 1. ist die Berechnung des Folgezustands bei Verzweigungen am einfachsten, wenn sich die Folgezustände in möglichst wenigen Bits unterscheiden.



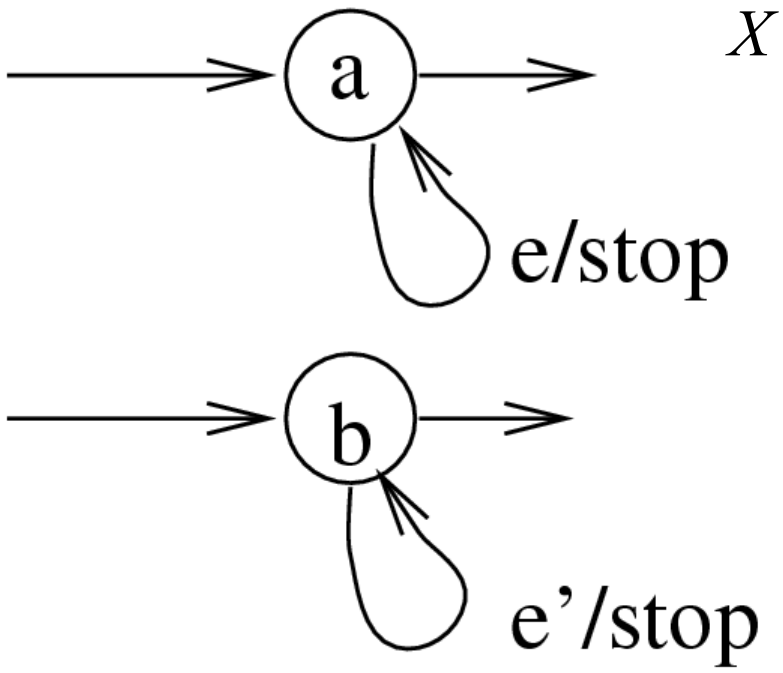
$\exists$ i mit: $a_i = c\ e_1$, $b_i = c\ e_2$

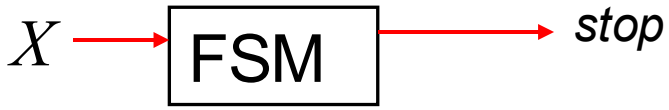
$\forall j \neq i: a_j = b_j$

$\Rightarrow a_j = b_j = c(e_1 \vee e_2)$

3. Ausgabe

Sofern zwei Zustände dieselbe Ausgabe erzeugen, ist die Berechnung der Ausgabe am einfachsten, wenn sich die Kodierung der Zustände in möglichst wenigen Bits unterscheidet.





$$Y_{stop} = a\, e \vee b\, e'$$

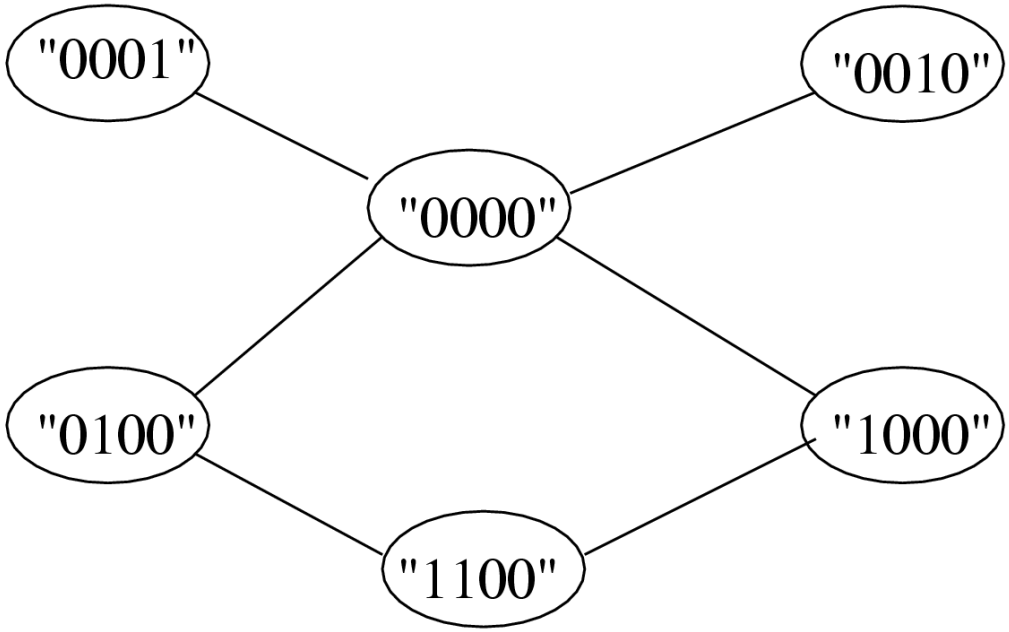
Bei Unterscheidung nur im Bit *i*

$$Y_{stop} = z_{n-1}\, z_{i+1}\, z_{i-1}\, z_0\, (e \vee e')$$

Graph der gewünschten einschrittigen Kodierungen

Nach Aufstellen des o.a. Graphen versucht ASYL, durch Betrachtung dieses Graphen möglichst viele der Forderungen hinsichtlich der Kodierungen zu erfüllen.

Dazu wird zunächst dem Knoten mit den meisten Kanten ein Code zugeordnet. Anschließend wird allen mit diesem Knoten verbundenen Knoten ein Code zugeordnet.



Dieser Prozess wird fortgeführt, bis alle Knoten erreicht sind (*breadth-first search*).

Weitere Ansätze zur Zustandskodierung

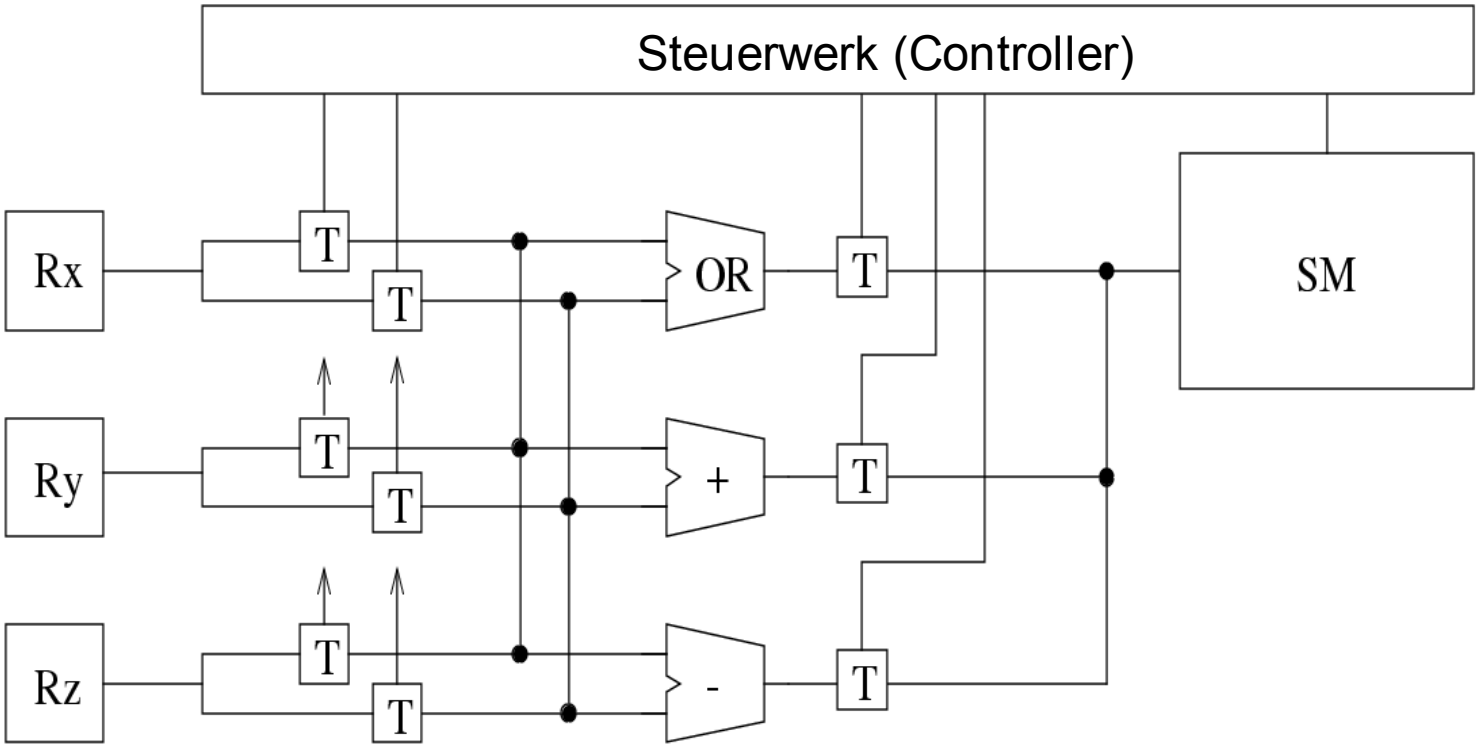
- ASYL: einfach zu erklären
- 1-aus-n (*one-hot-encoding*)
- Gray-Code, wo möglich
- Optimale Verfahren: bekannt (siehe z.B. [Har66]).
- Überblick: [Reu86]; Computing surveys, 1993
- In der Praxis: heuristisch.
- reguläre Ausdrücke: [Ull84].
- NOVA [Vil89]: symbolische Minimierung
- Lange, unverzweigte Folgen von Zuständen: Zähler als Zustandsregister [Ama89].
- MIPRE-System [Franke90].
- Fließbandverarbeitung, u.a.: CATHEDRAL [Man87].
- Hunderte von Verfahren mit speziellen Zielen
  - Minimierung des Energiebedarfs, …

Realisierung der Ausgabefunktion: - Einfache Techniken -

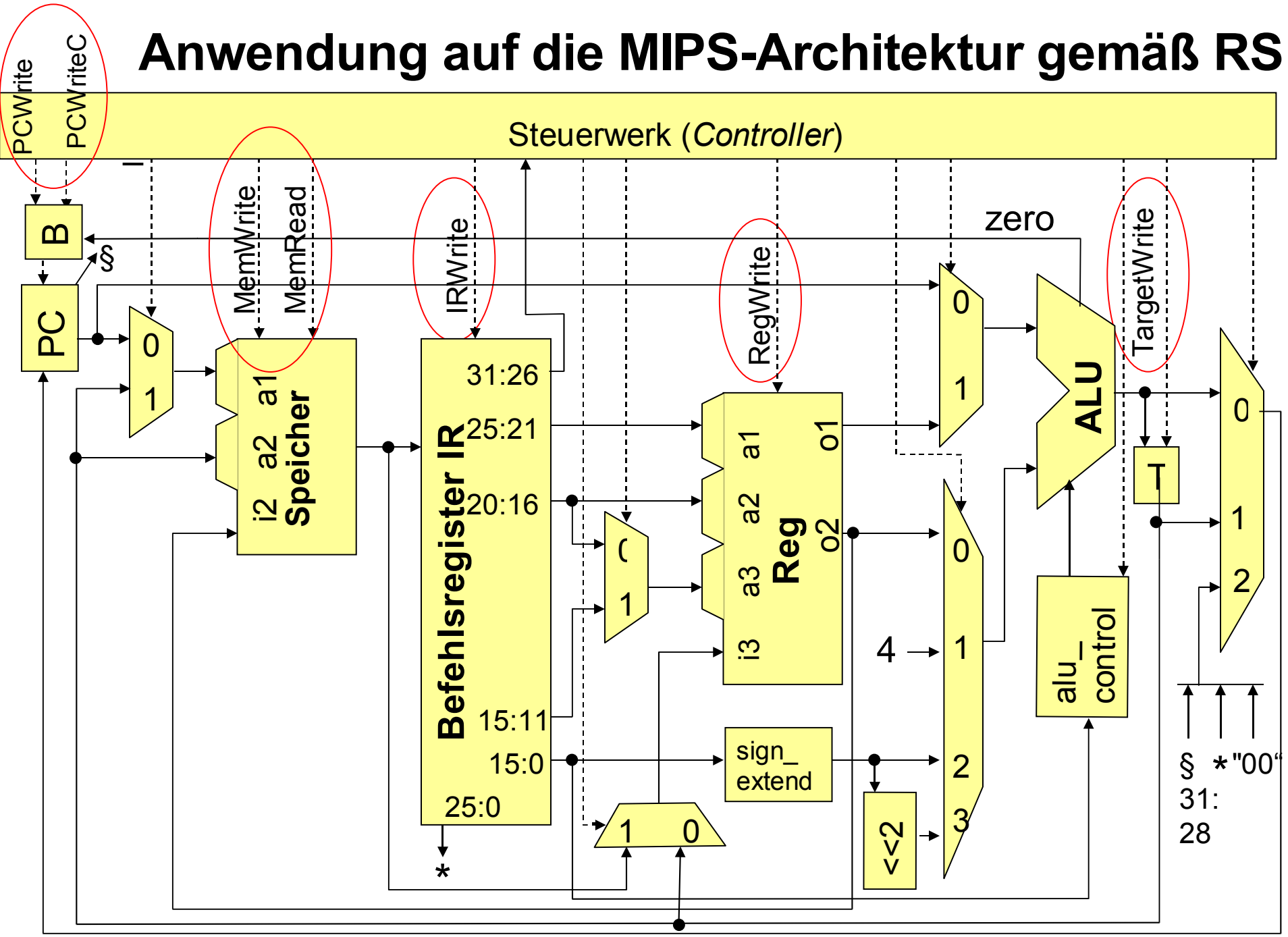
- Logikoptimierung ist auf die Implementierung von λ anwendbar. Z. Tl. berücksichtigen die Methoden der Zustandskodierung das Ziel, diese Funktion kostengünstig zu realisieren.
- Symbolische Minimierungstechniken lassen sich einsetzen, um Ausgabewerte zu kodieren.

Techniken aus der Mikroprogrammierung: - *direct control* -

Jedes Befehlsbit veranlasst die Ausführung einer Operation. Idee, dass Einheiten wie Addierer, Shifter, u.s.w. voneinander getrennt aufgebaut sind und deren Ergebnisse über Gates oder Treiber ausgewählt werden.

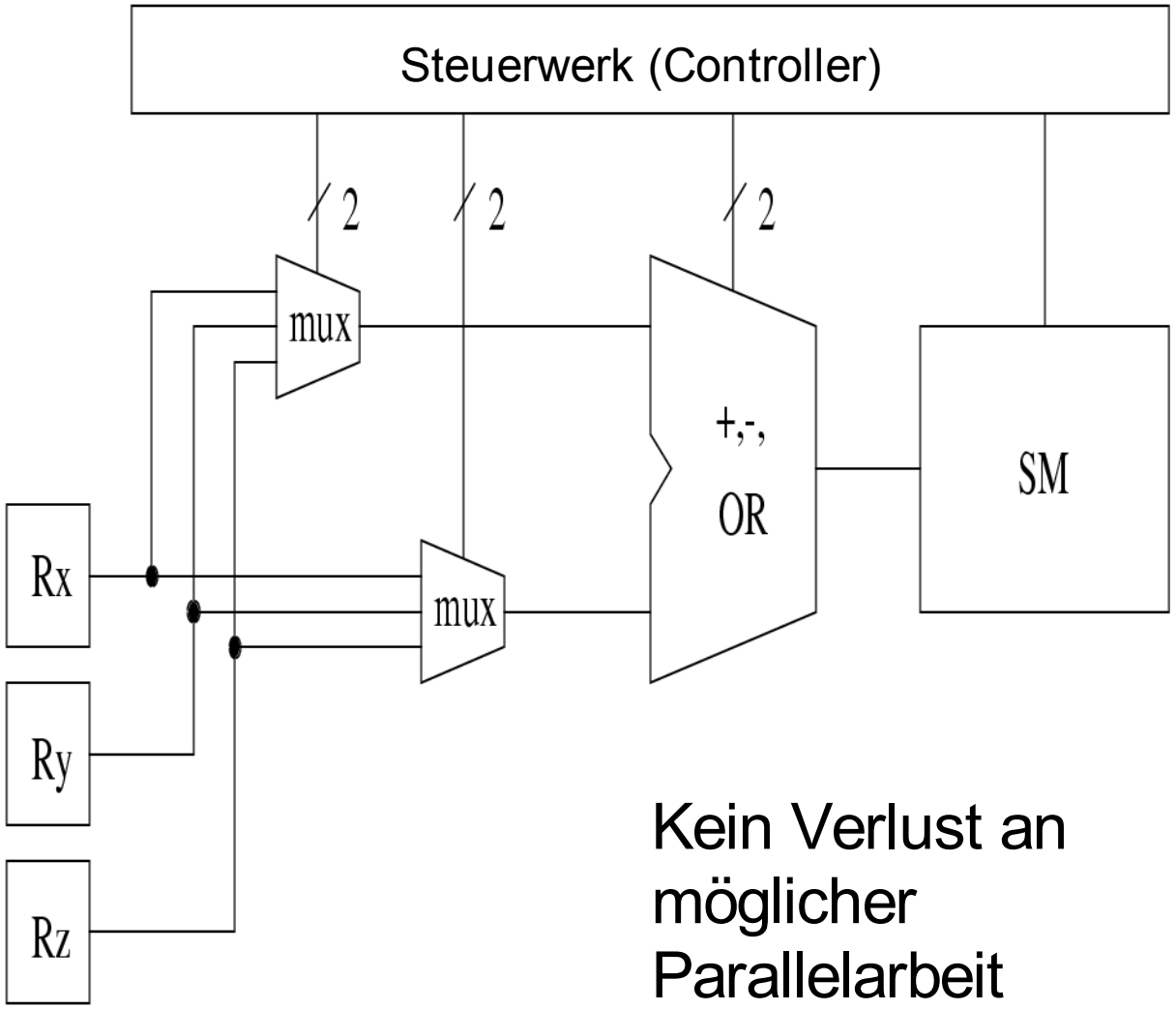


Anwendung auf die MIPS-Architektur gemäß RS



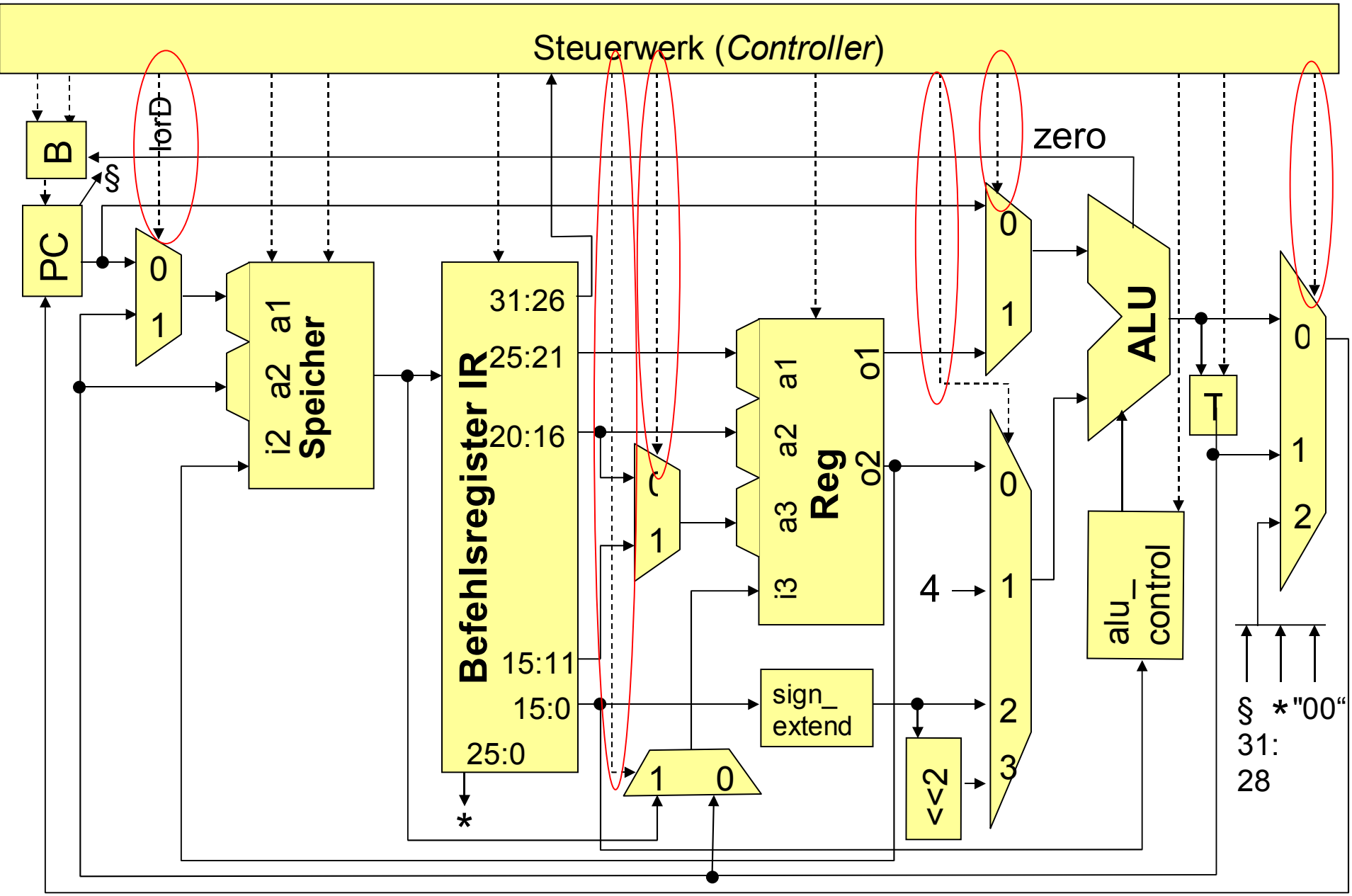
Single level encoding, direct encoding, minimal encoding

Operationen, die sich gegenseitig ausschließen, werden in einem Befehlsfeld gemeinsam kodiert.
Anwendung: Multifunktions-einheiten.
Orthogonalität.



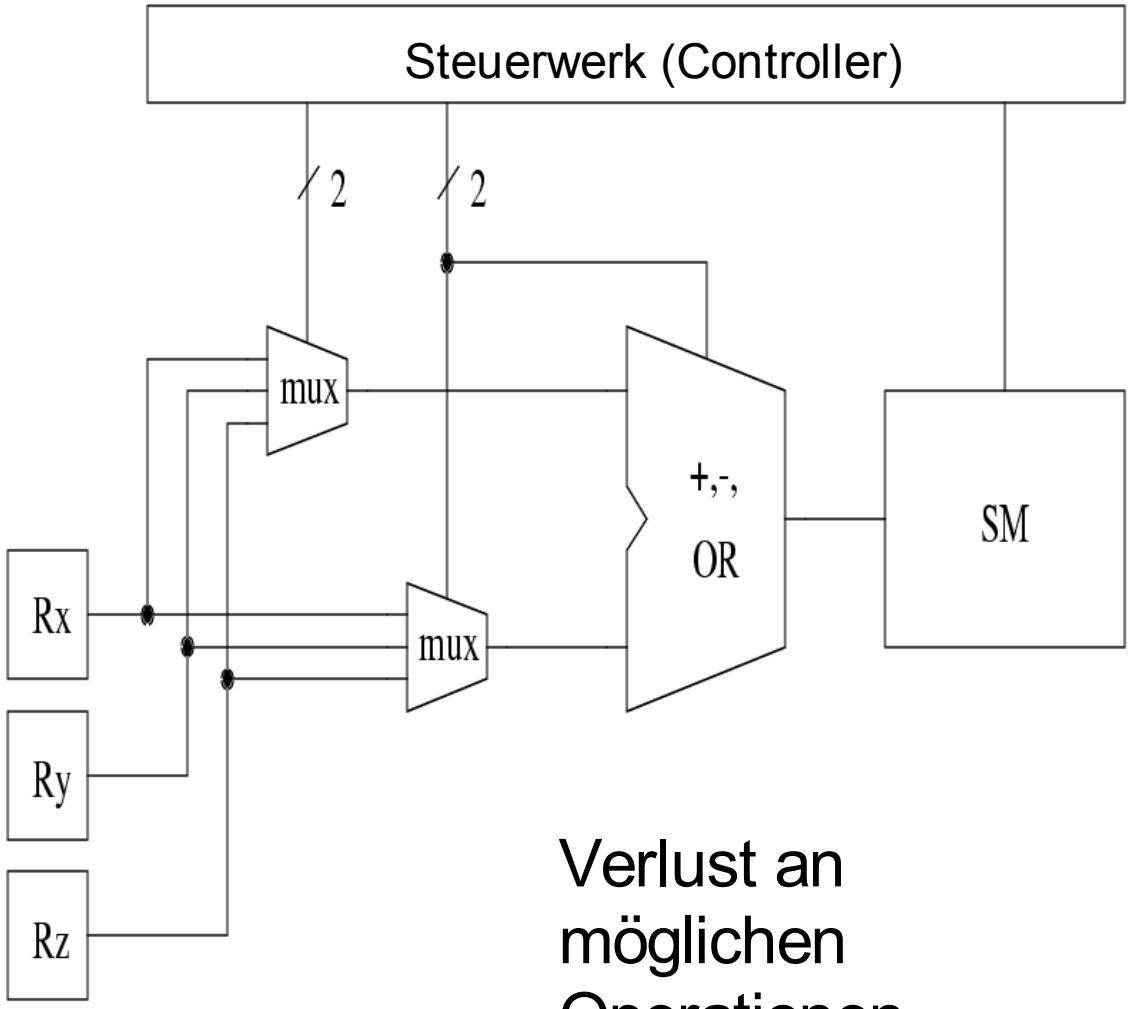
Kein Verlust an möglicher Parallelarbeit

Direct encoding für die MIPS-Architektur



Direct encoding mit *sharing*

Werden an zwei Steuereingängen entweder nie gleichzeitig Steuercodes benötigt oder werden stets nur gleiche Codes benötigt, so kommt man mit einem einzigen Steuerfeld im Befehl aus.

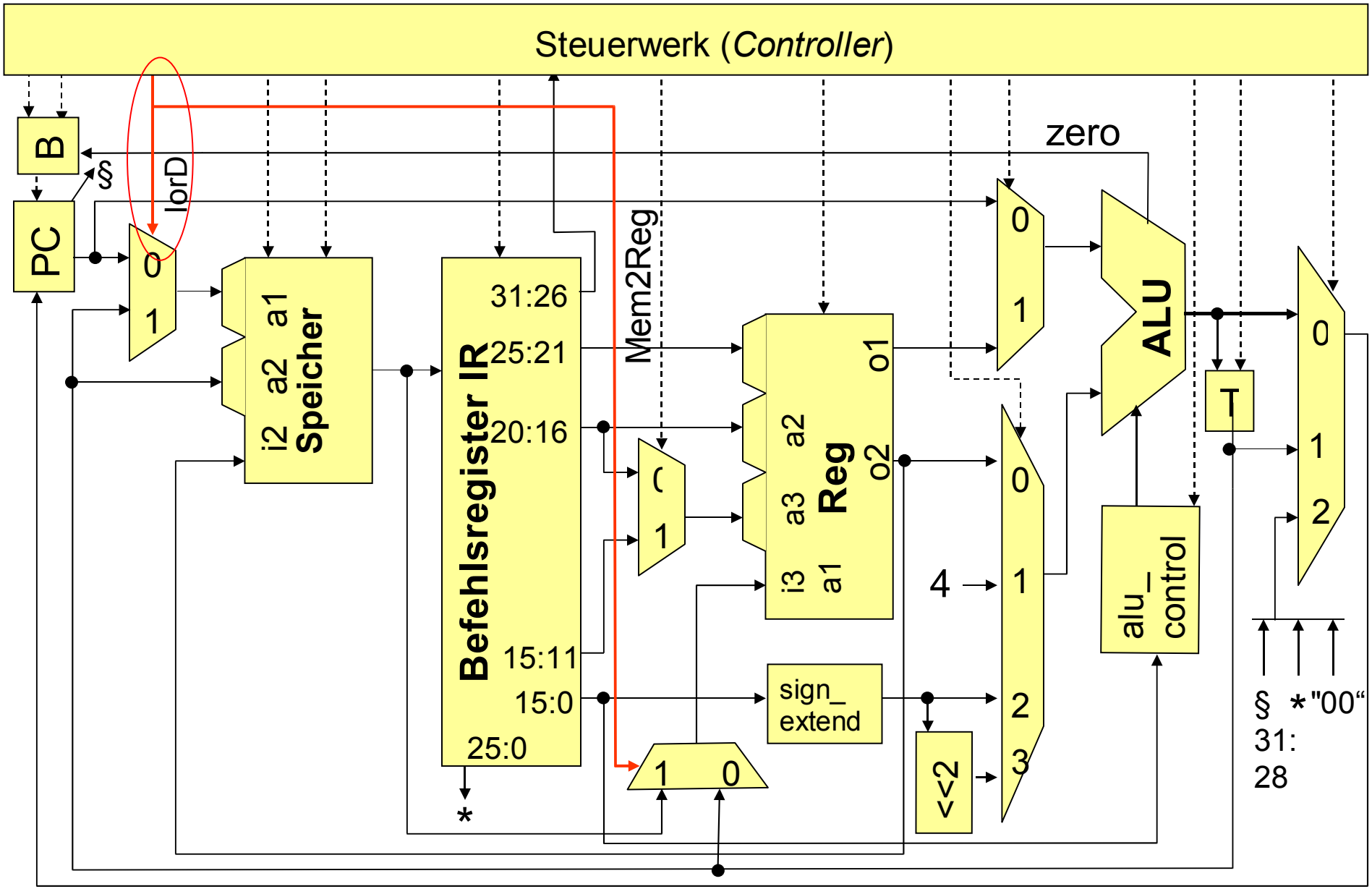


Verlust an möglichen Operationen

Direct encoding mit *sharing* bei MIPS-Architektur

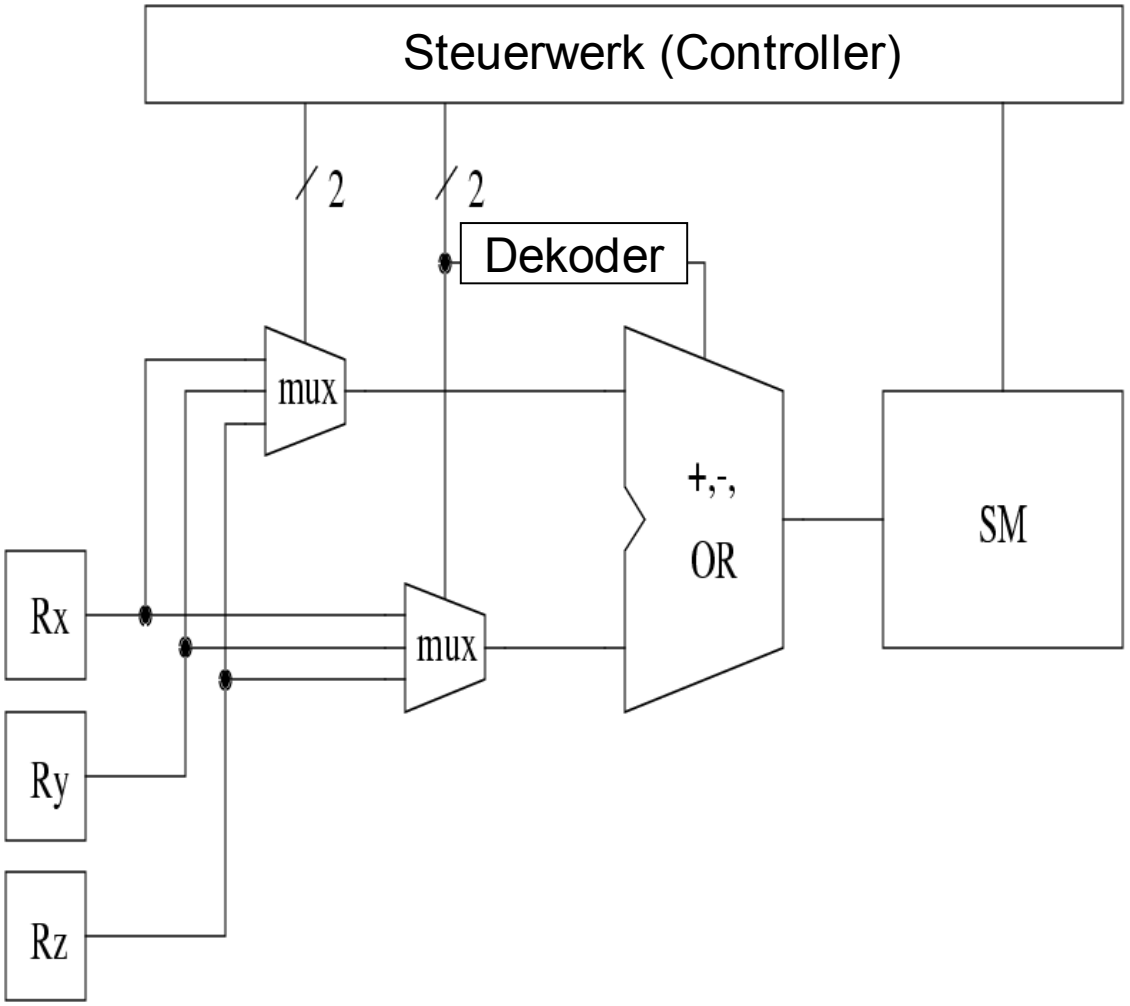
Zustand	Folge-Zustand bzw. – Zustände	PCWrite	PCWriteC	IorD	MemWrite	MemRead	IRWrite	Mem2Reg	RegDest	RegWrite	ALUSelB	ALUSelA	ALUOp	TargetWri	PCSource
fetch	decode	1	0	0	0	1	1	X	X	0	01	0	+	0	00
decode	f(Opcode)	0	0	X	0	0	0	X	X	0	XX	X	X	0	XX
mar, mar‘	load, store	0	0	X	0	0	0	X	X	0	10	1	+	1	XX
load	fetch	0	0	1	0	1	0	1	0	1	XX	X	X	0	XX
store	fetch	0	0	X	1	0	0	X	X	0	XX	X	X	0	XX
rr1	rr2	0	0	X	0	0	0	X	X	0	00	1	IR	1	XX
rr2	fetch	0	0	X0	0	0	0	0	1	1	XX	X	X	0	XX
branch	branch2	0	0	X	0	0	0	X	X	0	11	0	+	1	XX
branch2	fetch	0	1	X	0	0	0	X	X	0	00	1	=/-	0	01
jump	fetch	1	0	X	0	0	0	X	X	0	XX	X	X	0	10

Direct encoding mit *sharing* bei MIPS-Architektur



Direct encoding mit Umkodierung

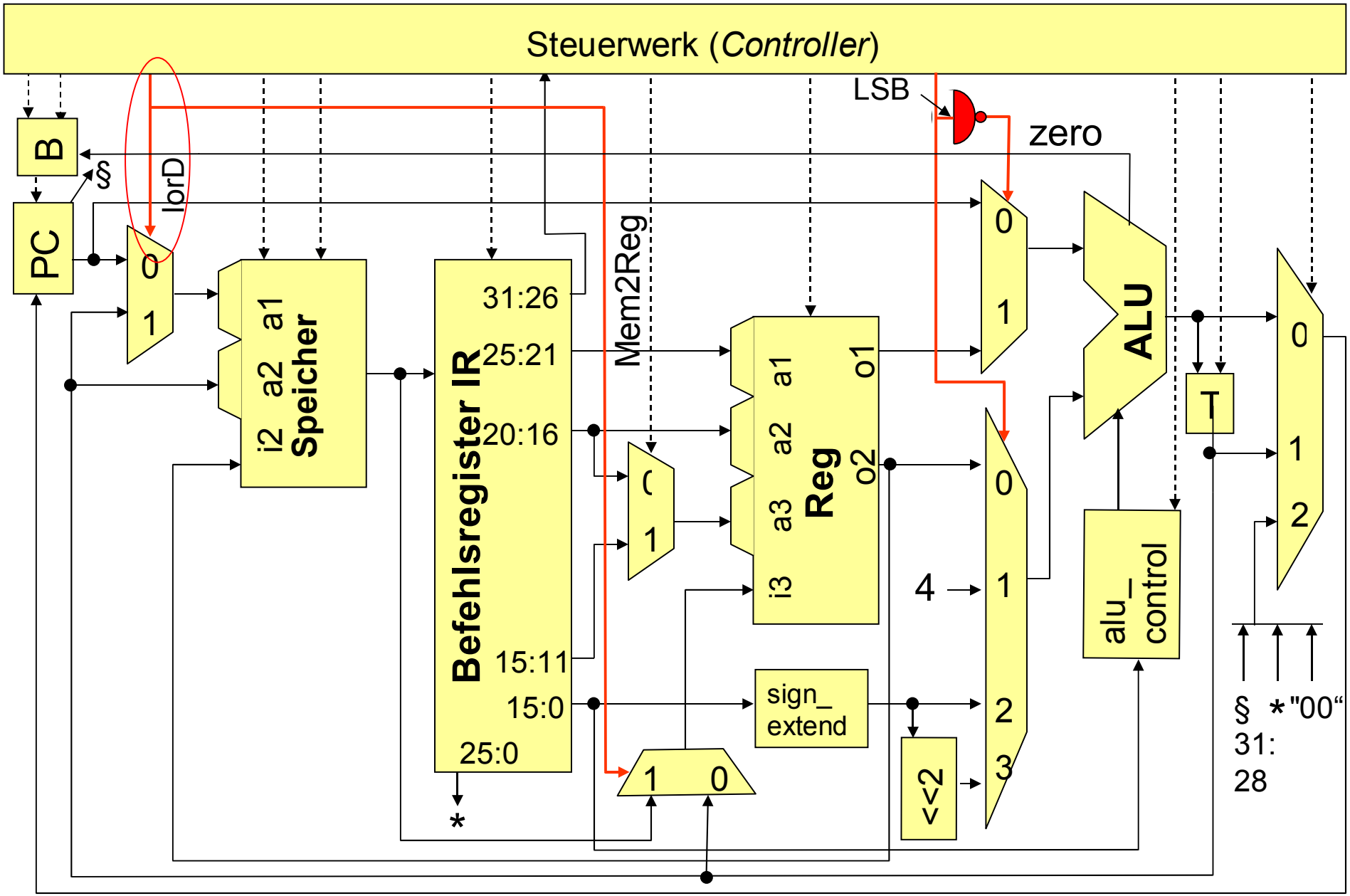
Als Erweiterung des *Sharing* kann man eine Umkodierung der Ausgabe zulassen



Direct encoding mit Umkodierung bei MIPS-Architektur

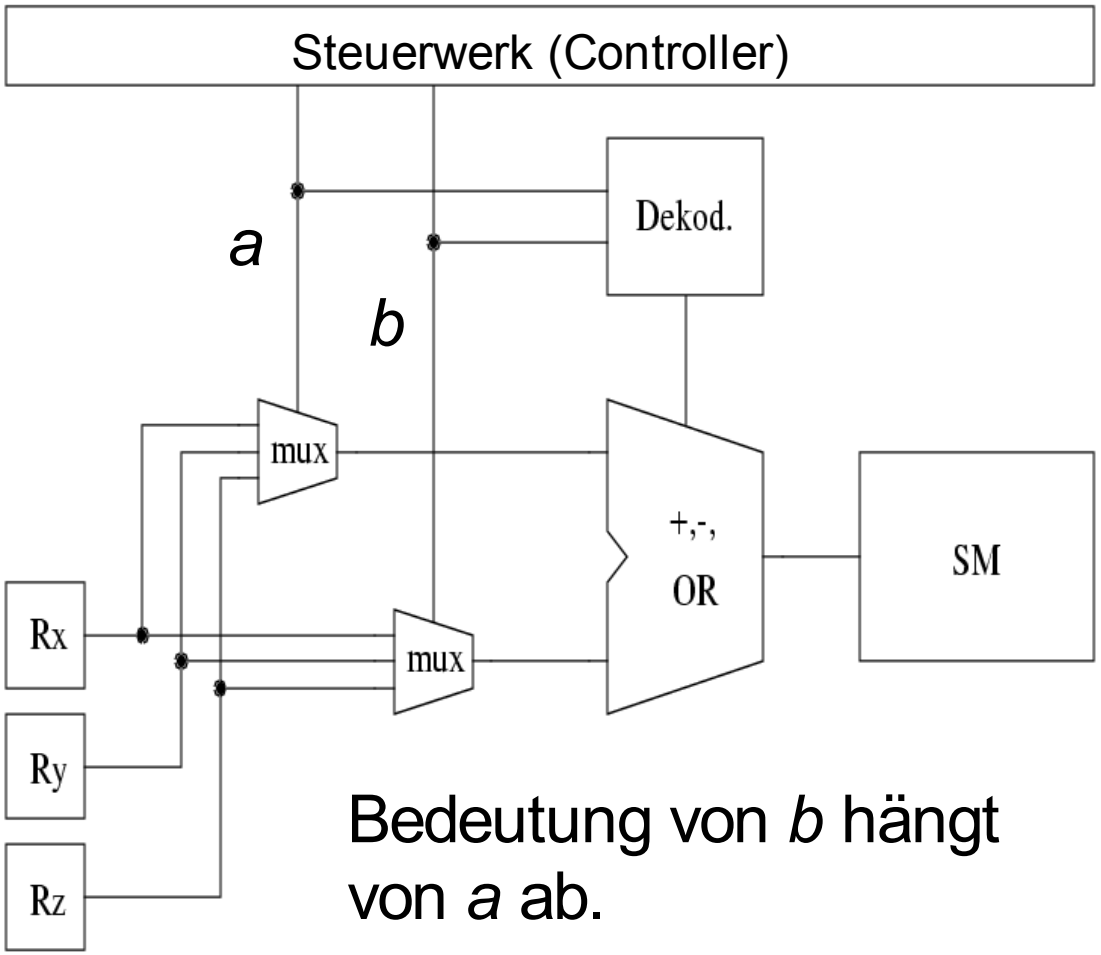
Zustand	Folge-Zustand bzw. – Zustände	PCWrite	PCWriteC	IorD	MemWrite	MemRead	IRWrit e	Mem2Reg	RegDest	RegWrite	ALUSelB	ALUSelA	ALUOp	TargetWri PCSource	
fetch	decode	1	0	0	0	1	1	X	X	0	01	0	+	0	00
decode	f(Opcode)	0	0	X	0	0	0	X	X	0	XX	X	X	0	XX
mar, mar‘	load, store	0	0	X	0	0	0	X	X	0	10	1	+	1	XX
load	fetch	0	0	1	0	1	0	1	0	1	XX	X	X	0	XX
store	fetch	0	0	X	1	0	0	X	X	0	XX	X	X	0	XX
rr1	rr2	0	0	X	0	0	0	X	X	0	00	1	IR	1	XX
rr2	fetch	0	0	~~X~~0	0	0	0	0	1	1	XX	X	X	0	XX
branch	branch2	0	0	X	0	0	0	X	X	0	11	0	+	1	XX
branch2	fetch	0	1	X	0	0	0	X	X	0	00	1	=/-	0	01
jump	fetch	1	0	X	0	0	0	X	X	0	XX	X	X	0	10

Direct encoding mit Umkodierung bei der MIPS-Architektur



Bit Steering

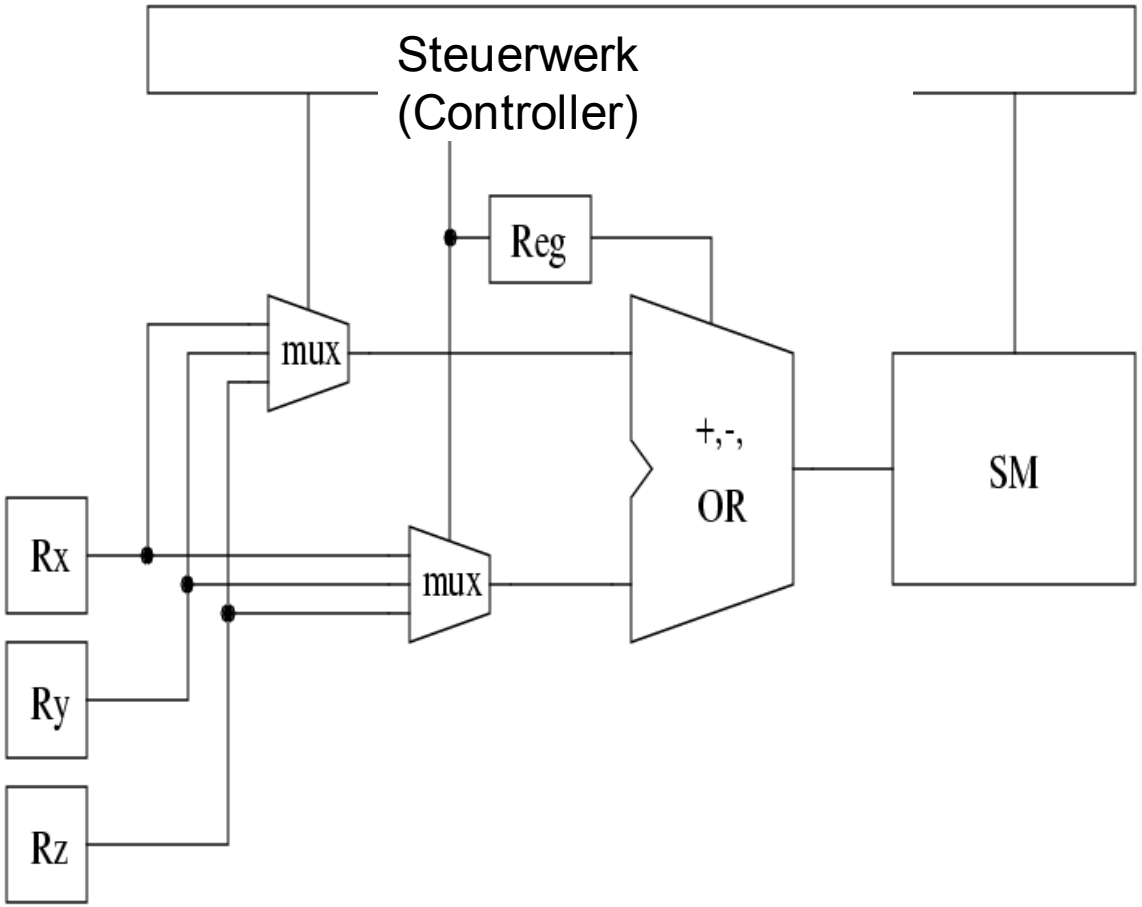
- ***Bit steering*** = Bedeutung eines Befehlsfeldes hängt von einem anderen Befehlsfeld ab.



Bedeutung von *b* hängt von *a* ab.

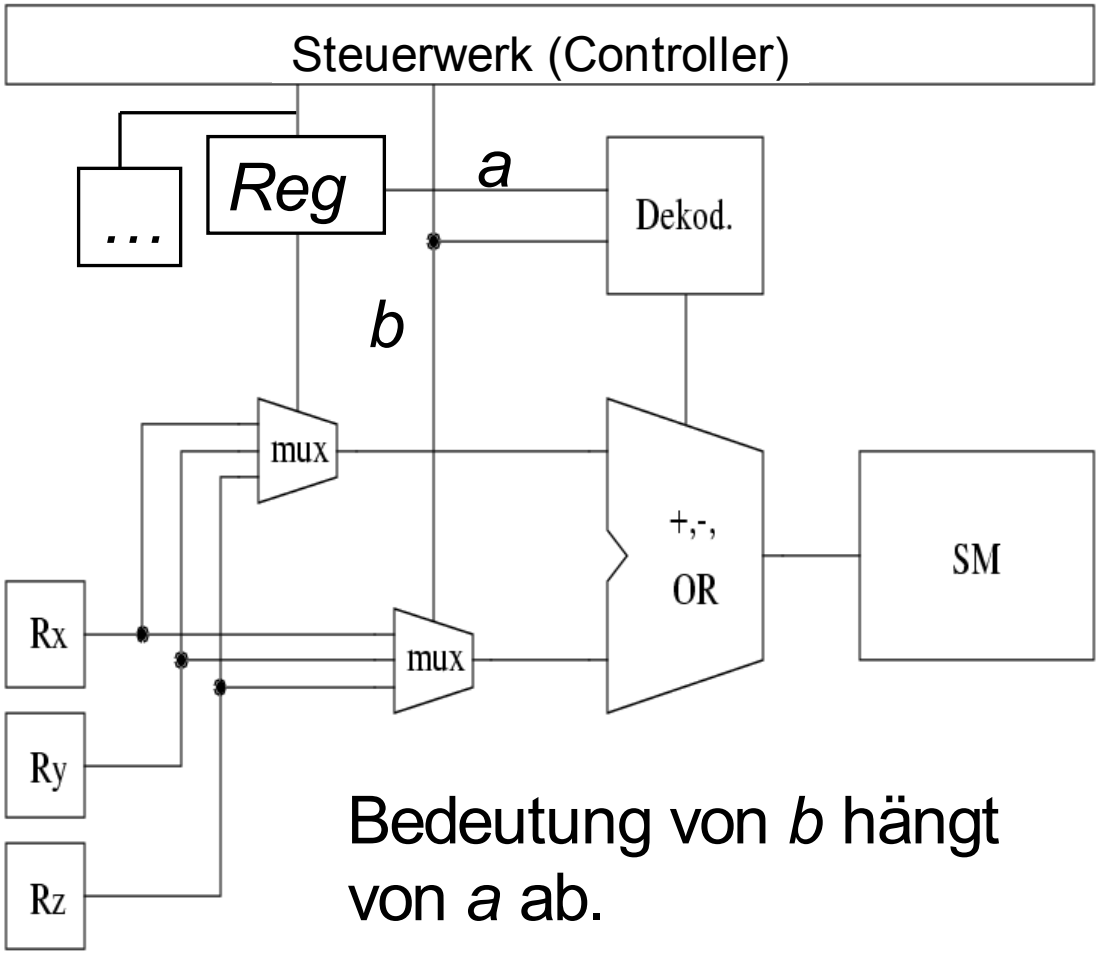
Residual control

Funktion der HW nicht nur vom aktuellen Befehlswort, sondern auch von einem inneren Zustand ab. Steuercodes werden in *residual control*-Registern abgespeichert. Anwendung: seltene Wechsel der Belegung der Kontrolleingänge, Nachbildung von Befehlssätzen.



Format shifting

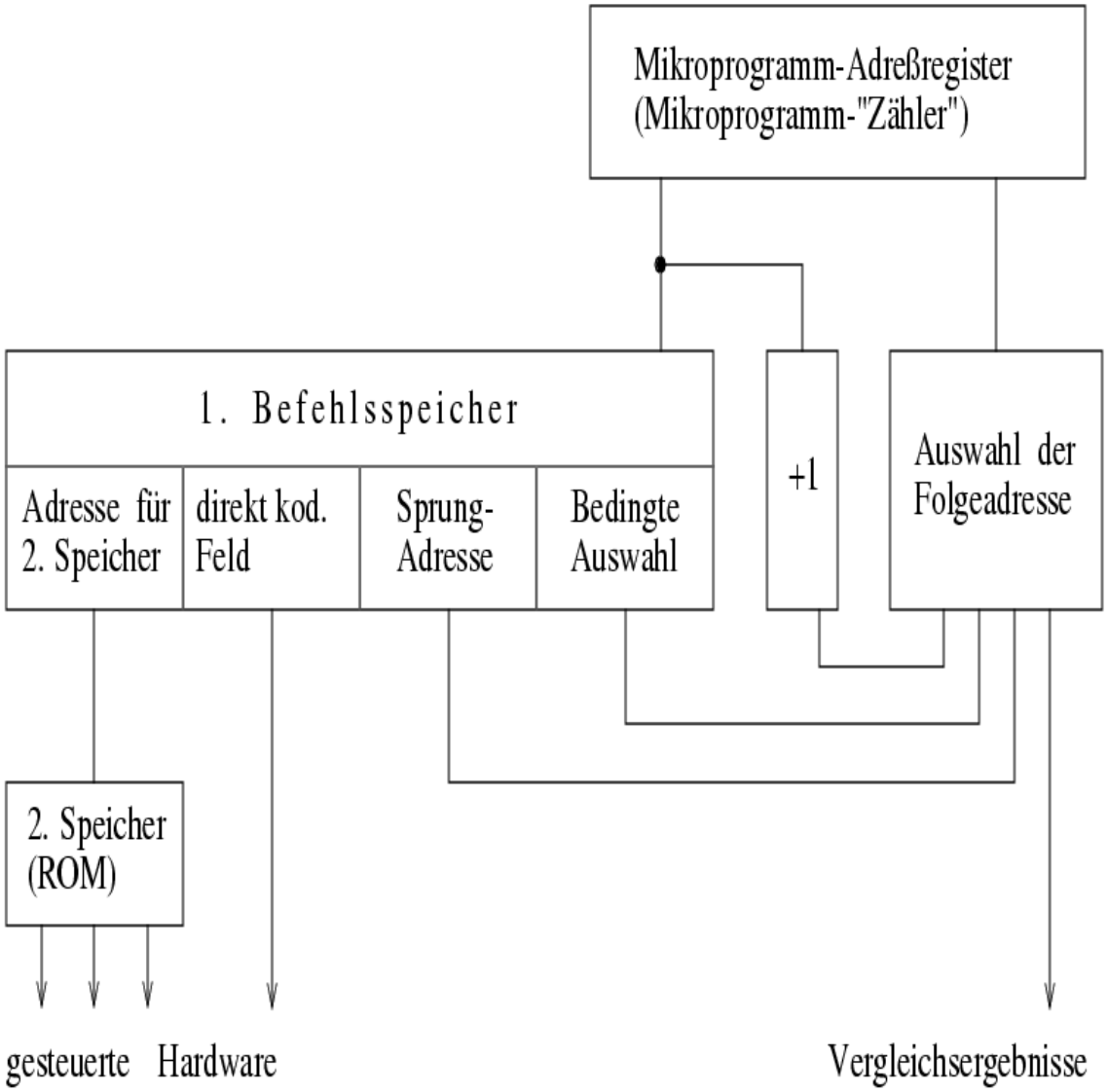
- ***format shifting*** = Bedeutung eines Befehlsfeldes hängt von einem *residual control*-Register ab.



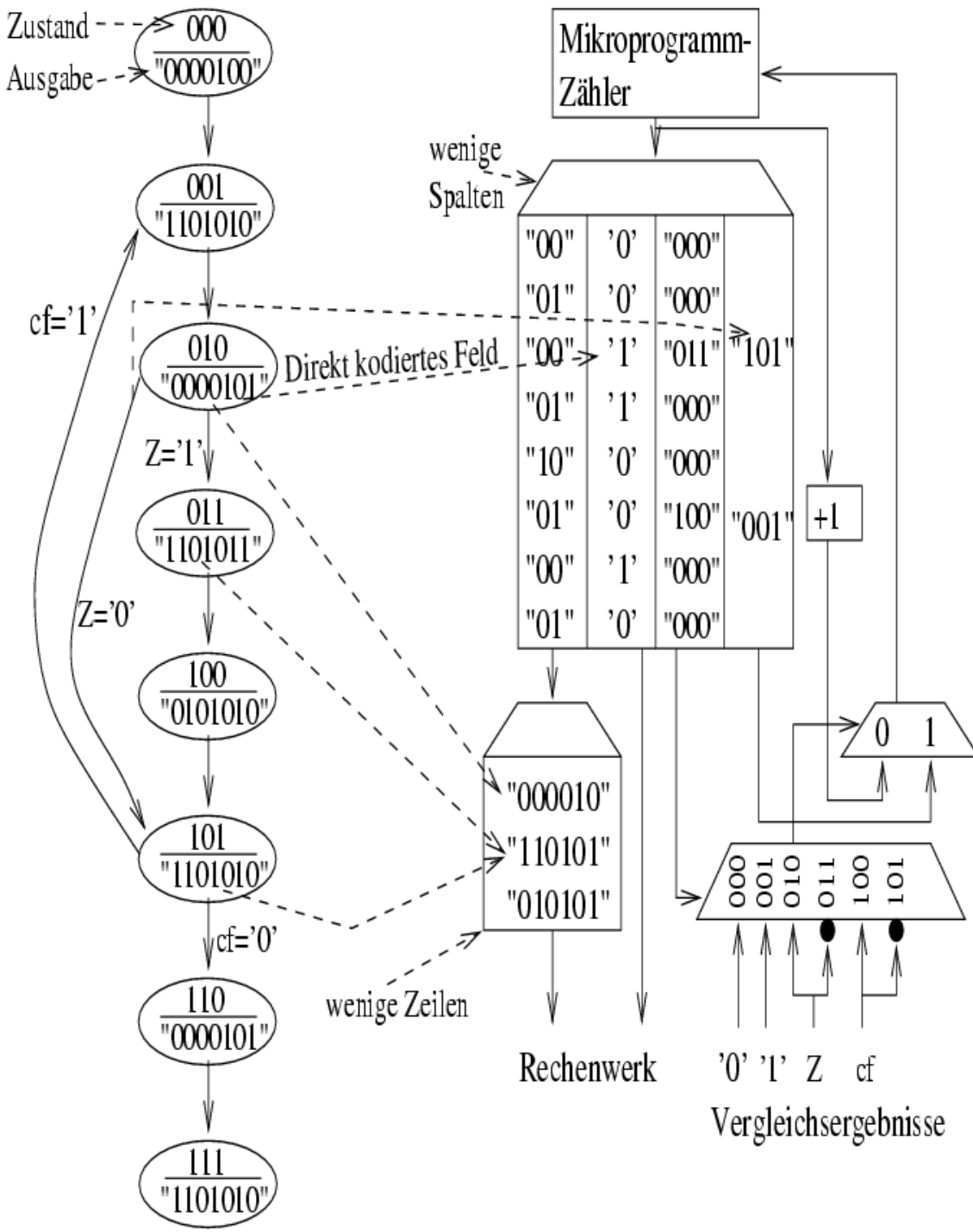
Bedeutung von *b* hängt von *a* ab.

Two-level control store

two-level control store = Großer Teil der Hardware über Dekoder oder ROM gesteuert.
Alle vorkommenden Kontrollworte genau einmal im ROM.
Prominente Anwendung: Mikroprozessor Motorola MC 68.000 („Nanoprogram-mierung“)



Realisierung eines konkreten Flussgraphen



Multiplexer wählt relevante Bedingung aus.

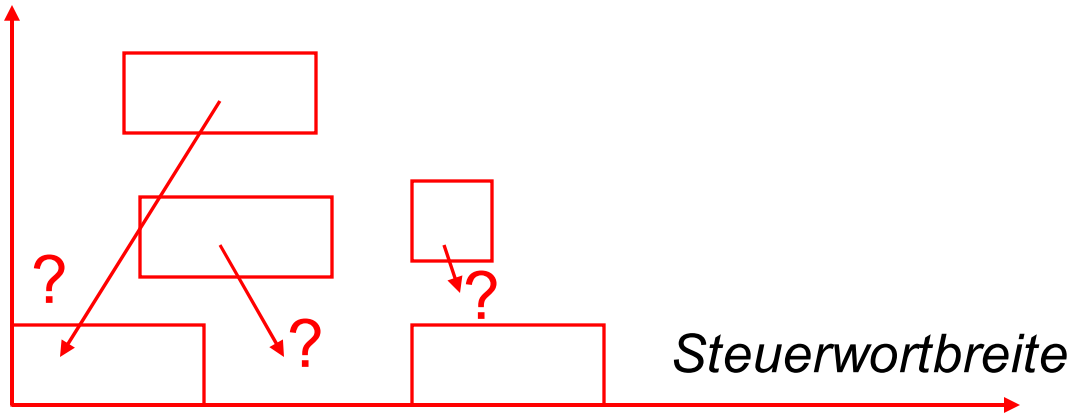
Befehlsfeldüberlagerung in TODOS (1)

TODOS [Mar86]: *direct encoding*-Methode: Jedem Steuereingang wird direkt ein Befehlsfeld zugeordnet. Überlagerung von Befehlsfeldern.

ℓ_i : Länge des Befehlsfeldes *i* in Bits.

$$c_{i,j} = \begin{cases} \text{true, Felder } i \text{ und } j \text{ sind in} \geq 1 \text{ Befehl nicht gleichzeitig redundant.} \\ \text{false, sonst} \end{cases}$$

Anordnung der Felder?

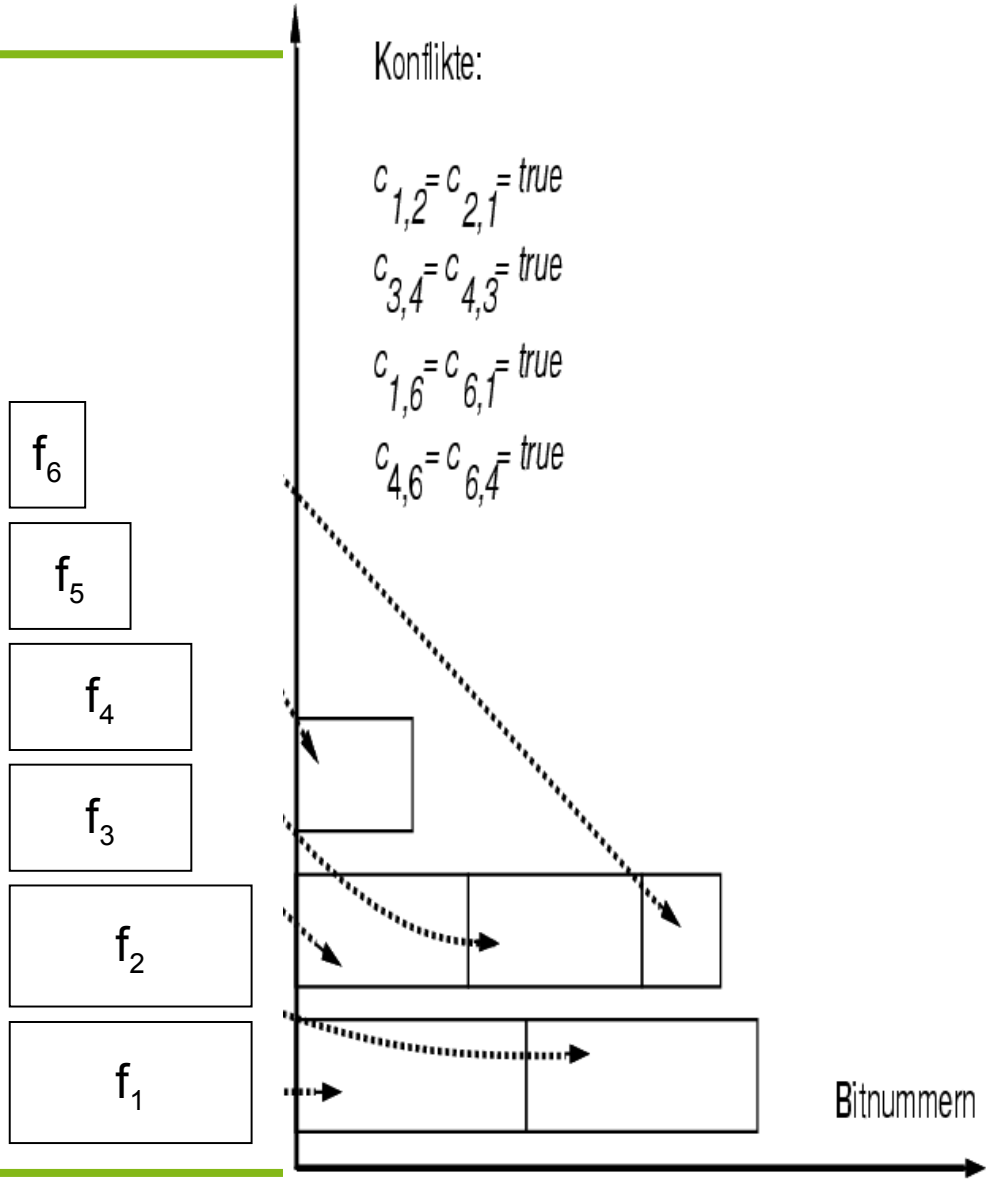


Äquivalent zu Scheduling-Problem von Jobs der Ausführungszeit ℓ_i mit Ressource-Konflikten $c_{i,j}$

Befehlsfeldüberlagerung in TODOS (2)

Lösung:

- Den jeweils längsten Feldern wird zuerst eine Position im Befehlsformat zugeordnet.
- Unter den gleich langen Feldern haben die Felder mit den meisten Konflikten Vorrang.
- Im Übrigen erfolgt die Auswahl nach *best fit*-Methode.



Zusammenfassung

- Zustandsreduktion
  - Ergebnisse aus der Automatentheorie
- Zustandskodierung
  - ASYL als Beispiel
- Realisierung der Ausgabefunktion
  - *Direct control*
  - *Direct encoding* (mit *sharing*)
  - *Residual control*
  - *Bit steering*
  - *Format shifting*
  - *Two-level control store*