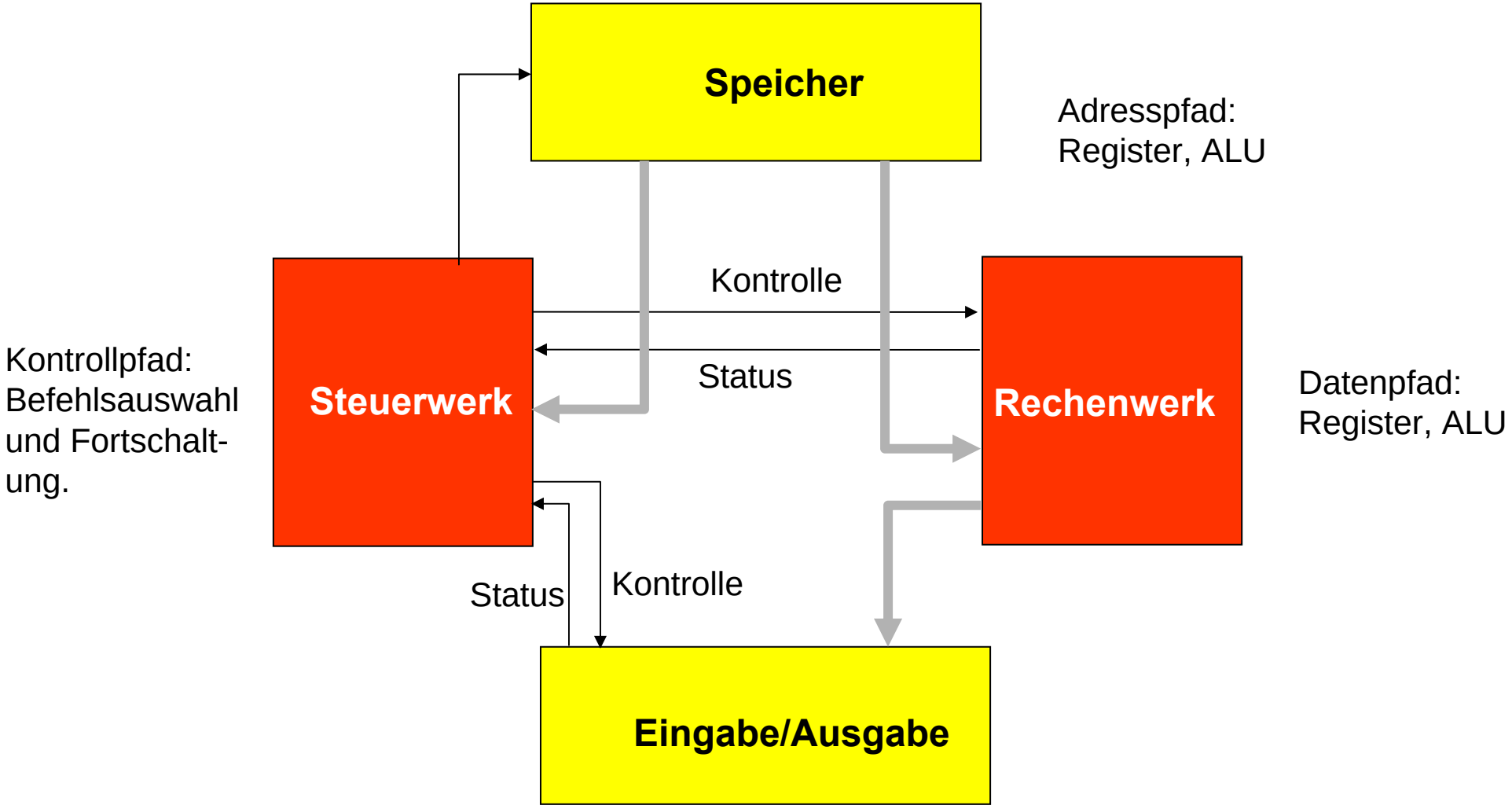
Realisierungsalternativen für eine Zentraleinheit

Otto-von-Guericke-Universität Magdeburg

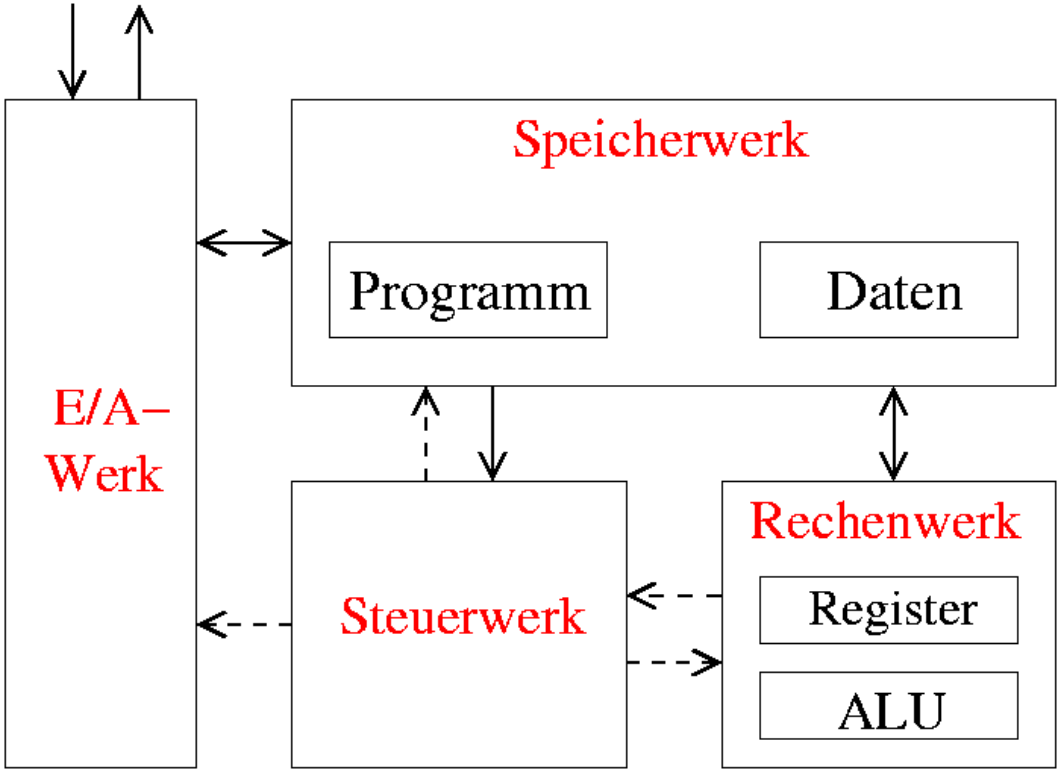
Komponenten des klassischen Rechners

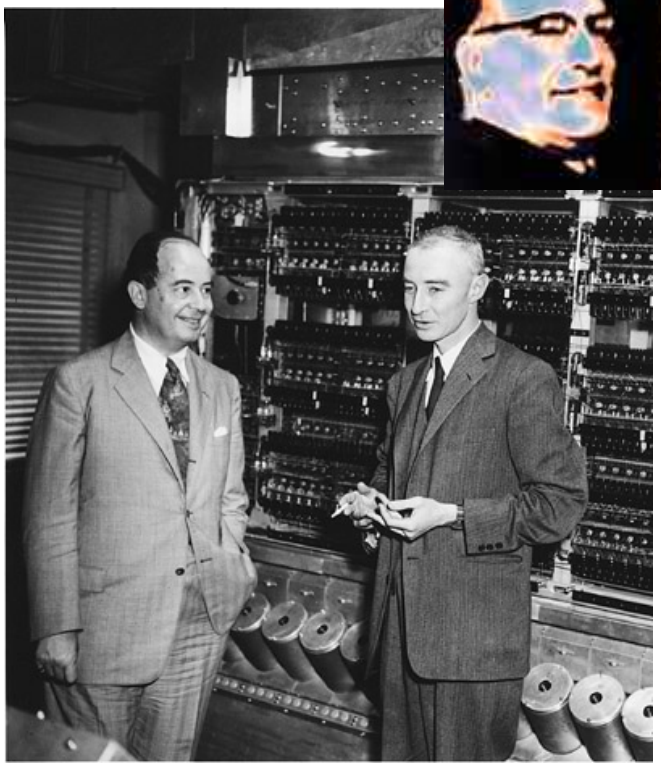


Historische Entwicklung

John Mauchly (1907-1980) &
J. Presper Eckert (1919-1995)

- 1945: Von-Neumann Architektur (Eckert/Mauchly, John von Neumann)

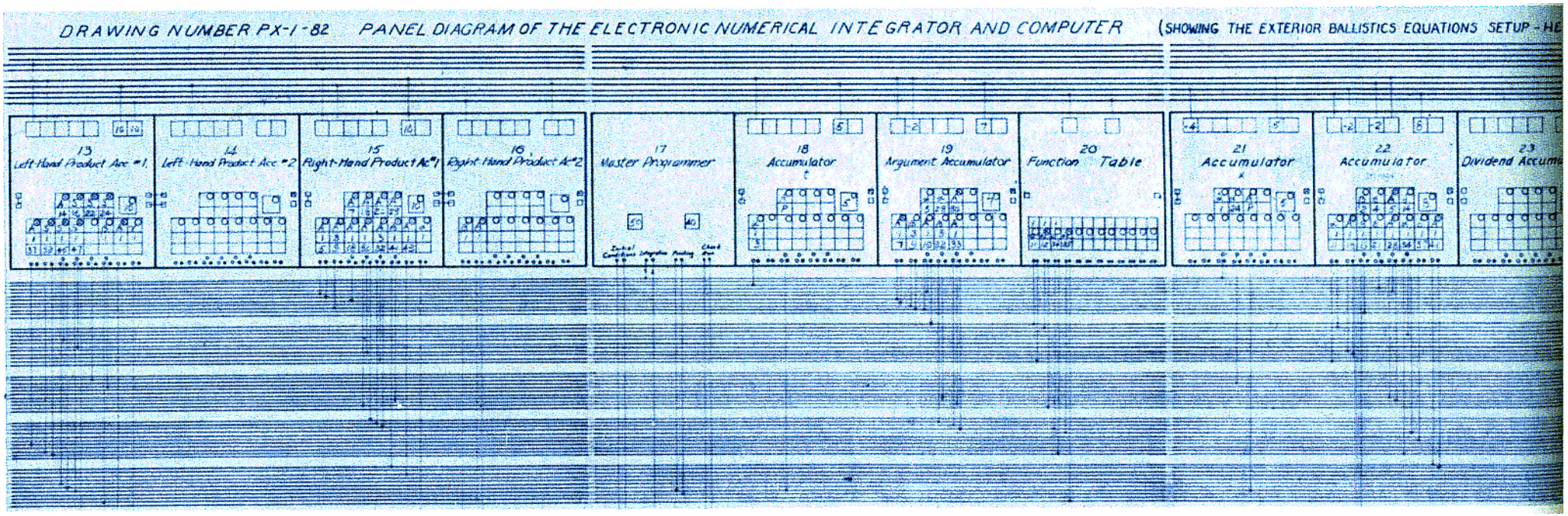




John von Neumann, left, with Robert Oppenheimer,
Director of the Institute for Advanced Study from 1947-66

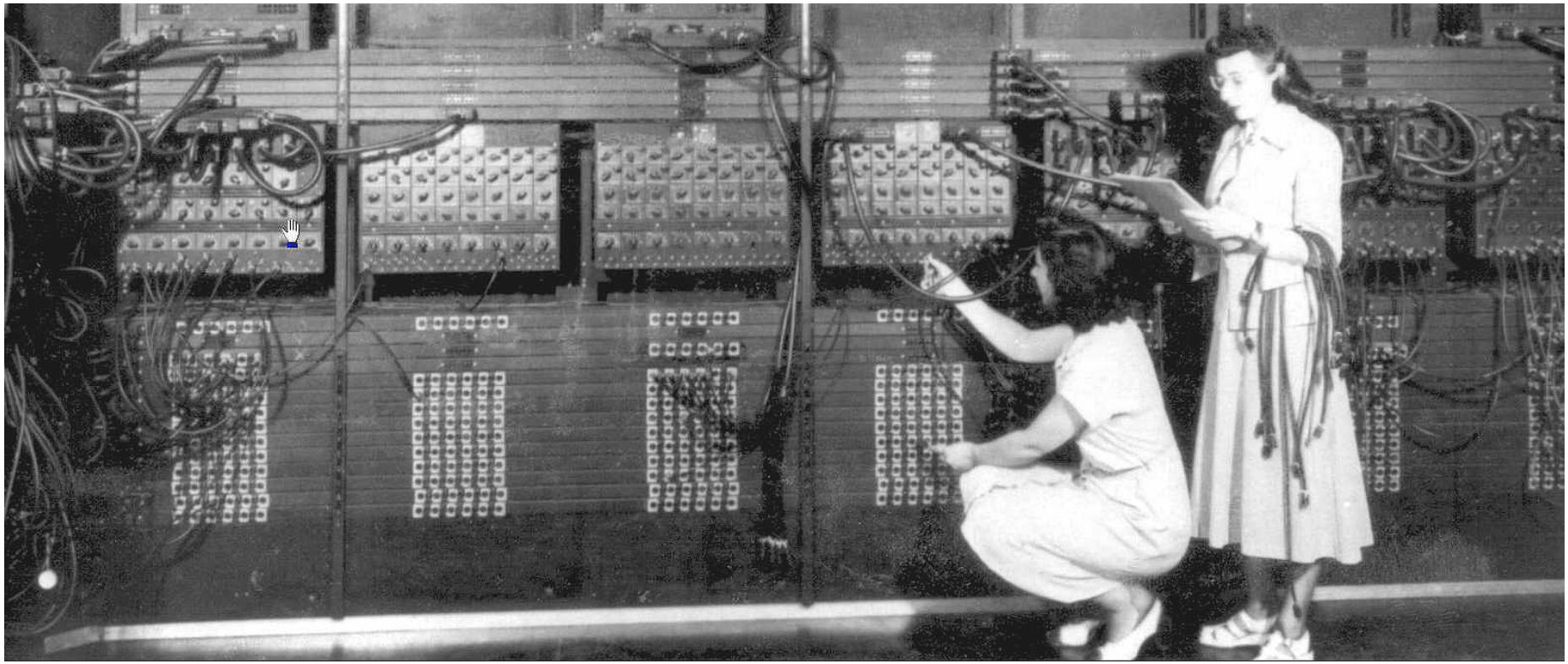
http://www.ias.edu/the-institute-letter/archive/03Winter/winter03.php

Programmierung des ENIAC



Quelle: http://www.columbia.edu/acis/history/eniac.html

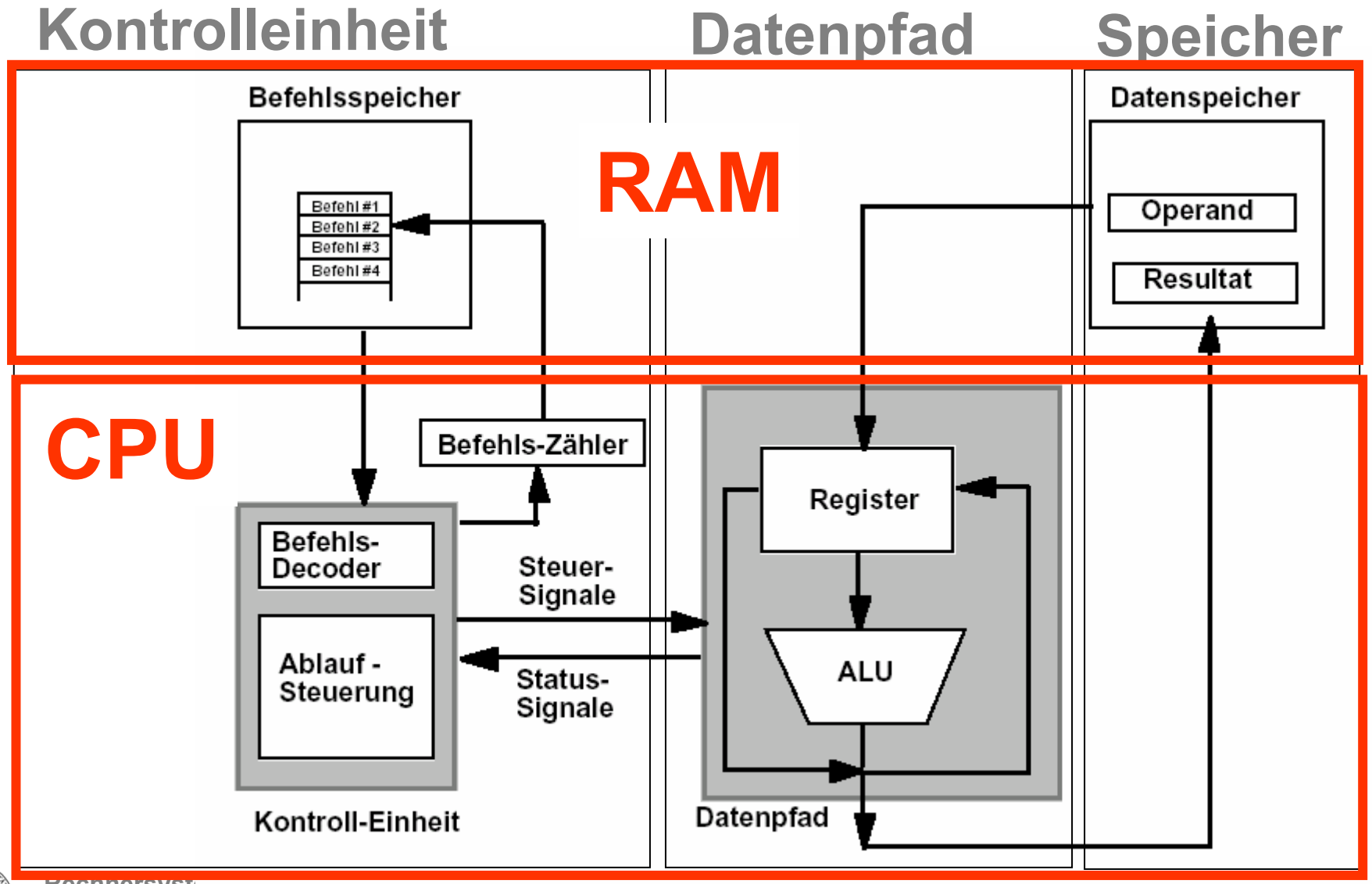
Programmierung des ENIAC (1945)



Gloria Gorden and Ester Gerston at work on the ENIAC

Quelle: http://www.columbia.edu/acis/history/eniac.html

Prinzip des sequentiellen programmgesteuerten Prozessors



Der Entwurf einer CPU

?

Zielkonflikte

Where shall I begin?

Befehlssatz?
Wortbreite?
Speichergröße und Organisation?
Registersatz?
Arithmetisch/logische Einheiten?
Leistung?
Geldmittel?

Befehlsklassen:

Datenpfad-bezogene Befehle:

Arithmetische Befehle:	**ADD, SUB, MULT, DIV, SQRT, SIN, COS, ARC,.....**
Logische Befehle:	**AND, OR, XOR, NOT**
Shift- Befehle:	**ASL, ASR, LSL, LSR, ROR, ROL,**
Vergleichsbefehle:	**<, ≤, >, ≥, =**
Transferbefehle:	**LOAD, STORE, MOVE**

Kontrollfluß-bezogene Befehle: Programmverzweigungen (Sprungbefehle)

Unbedingter Sprung	**BRA, JMP**
Bedingter Sprung	**BRx (Bedingung)**
Unterprogrammsprung	**BSR, JSR**
Softwareinterrupt, Trap	**SWI, Trap, SVC, ...**

Ein/Ausgabe, Betriebssystem, Spezielle Funktionseinheiten (z.B. Grafik), ...

Wortformate und Speichergröße

Wie viele Befehle müssen codiert werden?

Wie groß sind meine Operanden?

Wie viele Register sollen verfügbar sein?

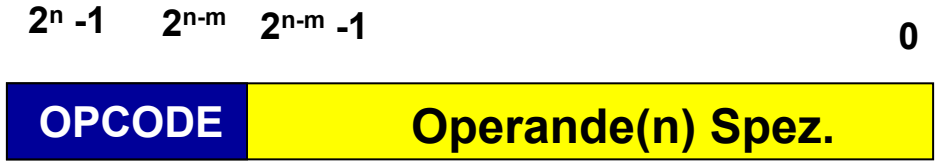
Welcher Adreßraum soll unterstützt werden?

Wortformate und Adreßraumgröße

Datenformat

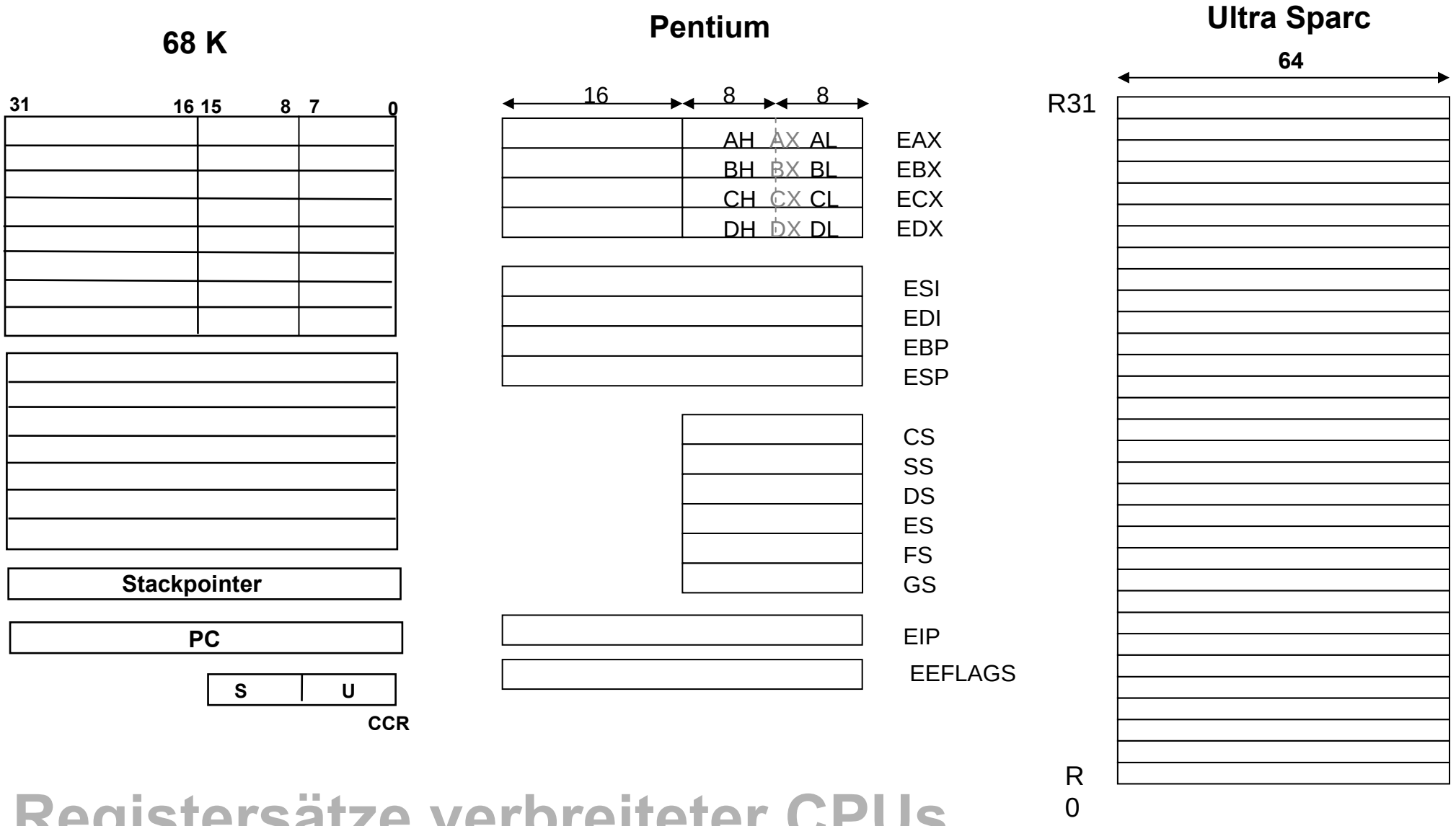
- Festkomma?
- Fließkomma?

Befehlsformat



m Bit

Operandenspezifikation:
- wie viele Operanden?
- Speicheradressen ?
- Operandenregister ?
- direkte Operanden ?



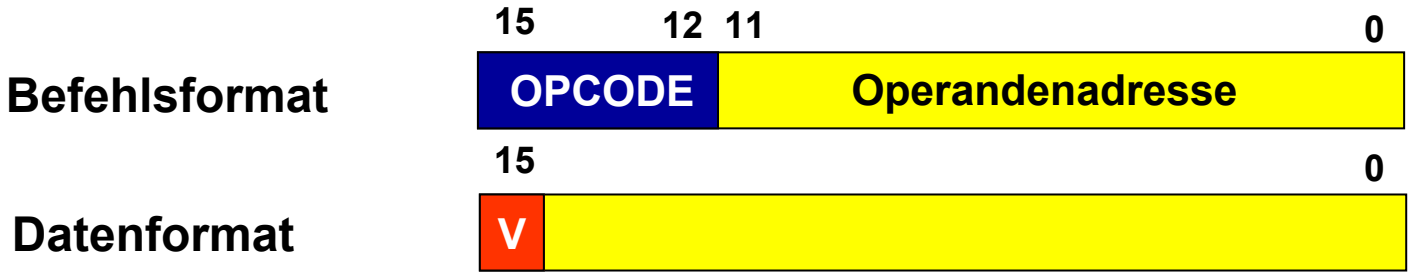
Registersätze verbreiteter CPUs

Ein sehr einfacher Rechner

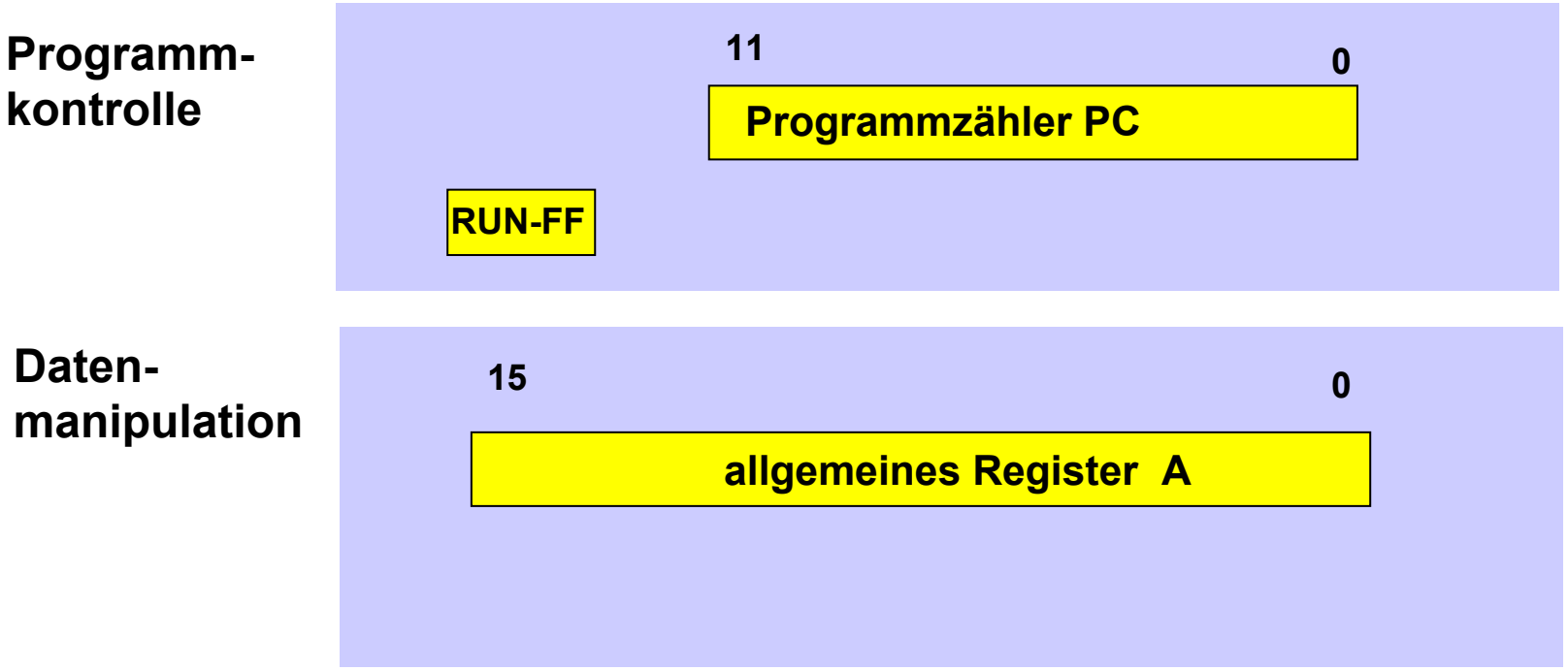
Caxton C. Foster:
Computer Architecture
Computer Science Series,
v. Nostrand Reinhold Company, 1970

- Wortorientiert,
- 1 Datenregister
- 1 Adreßregister
- 16-Bit Worte
- 16 Befehle

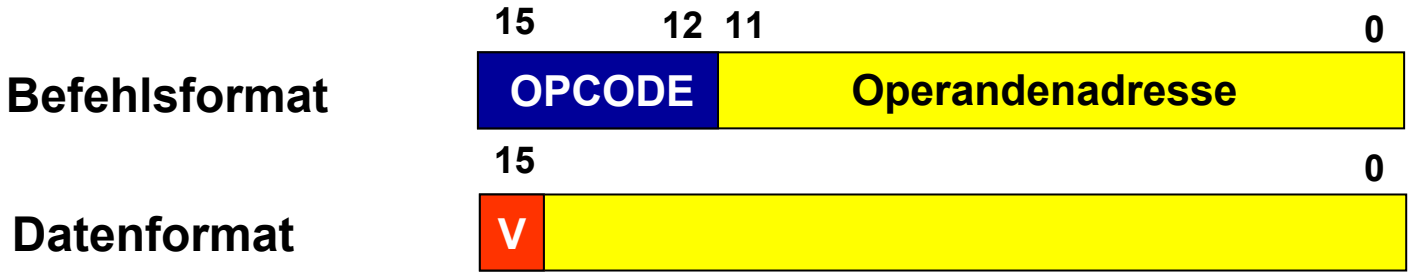
Wortformate



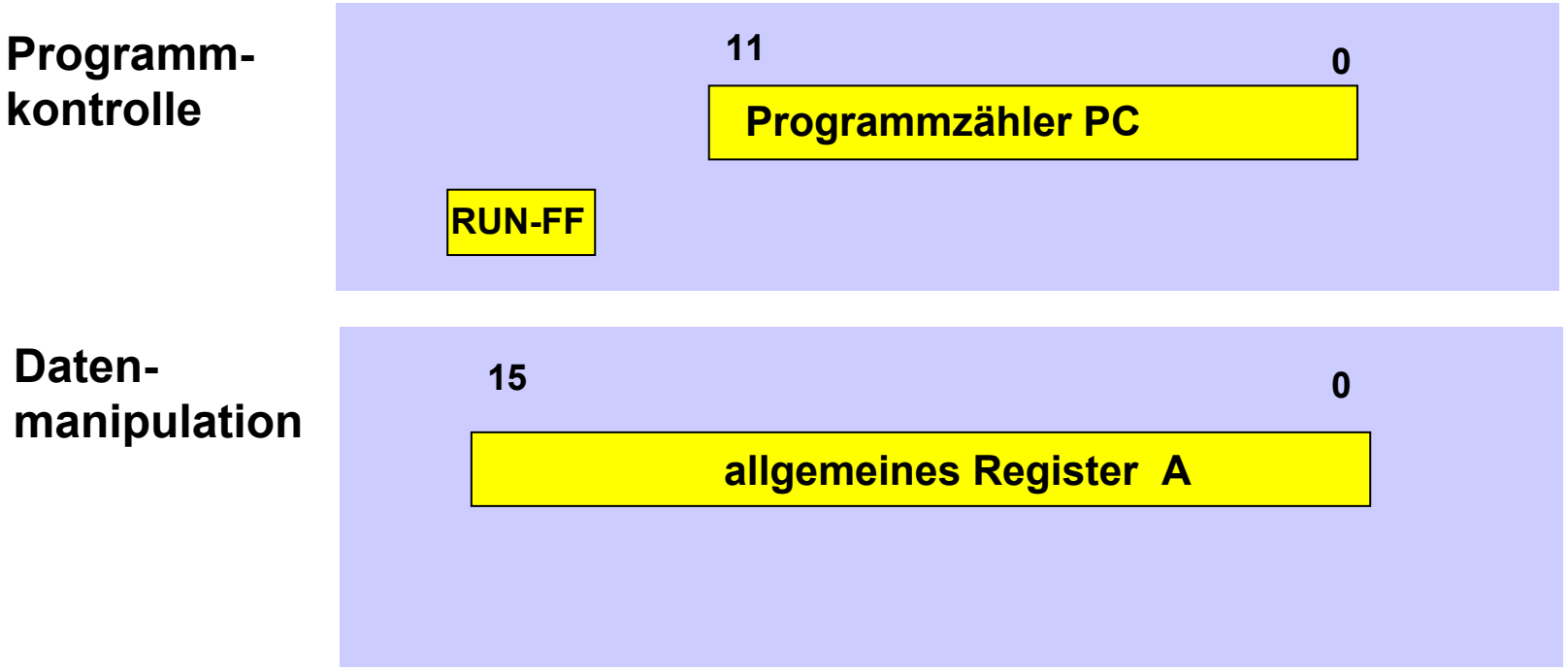
.. und Programmiermodell



Wortformate



.. und Programmiermodell



Befehlsklassen:

Datenpfad-bezogene Befehle:

Arithmetische Befehle:	**Addieren (ADD)**
Logische Befehle:	**AND, OR, XOR, Komplement (AND, IOR,XOR,NOT)**
Shift- Befehle:	**Links-Shift (RAL)**
Vergleichsbefehle:	**<, ≤, >, ≥, =**
Transferbefehle:	**LOAD, STORE, MOVE**

Kontrollfluß-bezogene Befehle: Programmverzweigungen (Sprungbefehle)

Unbedingter Sprung	**JMP**
Bedingter Sprung	**JMP (Bedingung)**
Unterprogrammsprung	**JSR**

Ein/Ausgabe - Befehle

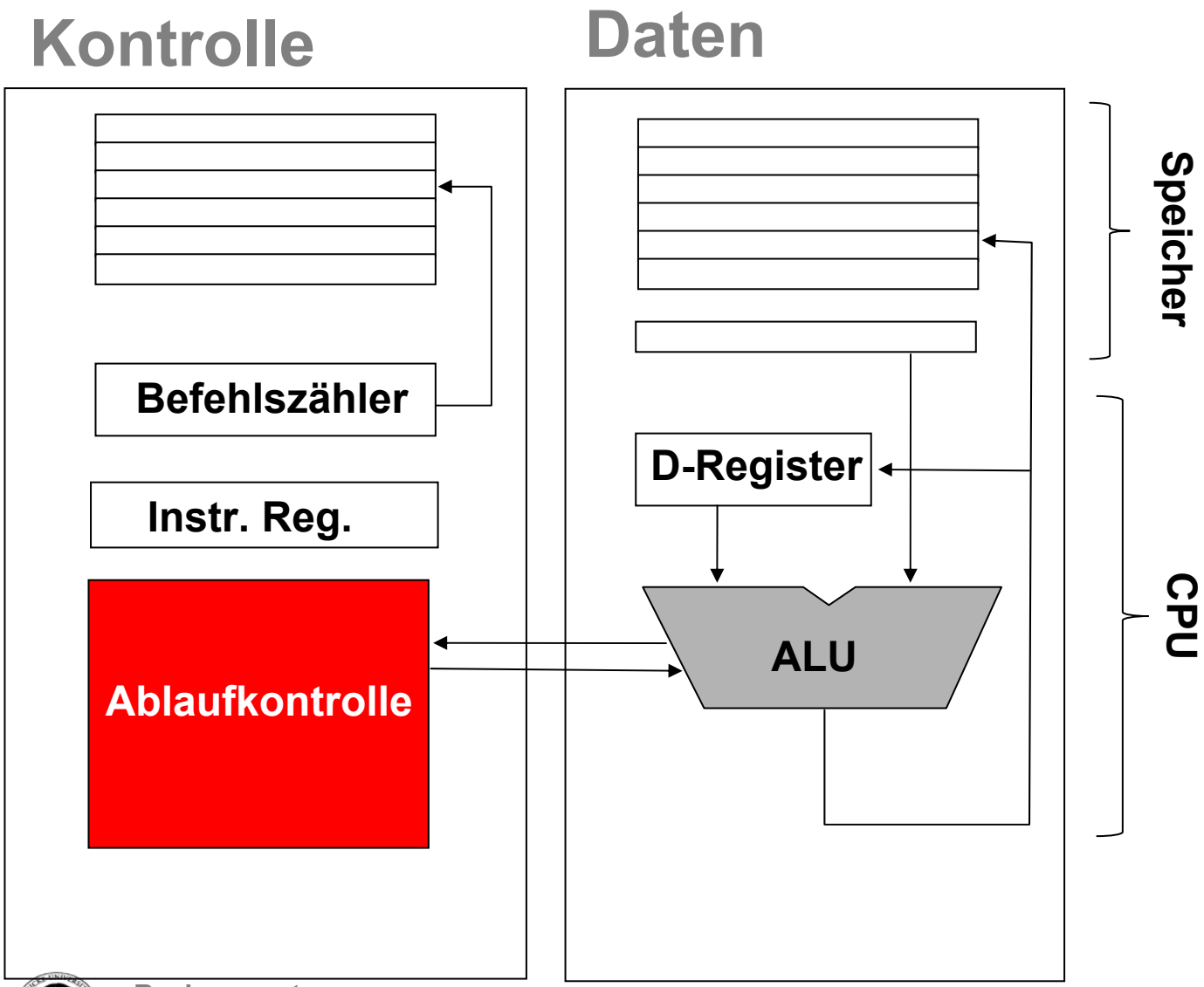
Ein (sehr) einfacher Befehlssatz

Caxton C. Foster:
Computer Architecture
Computer Science Series,
v. Nostrand Reinhold Company, 1970

0000	HLT		Halt
0001	JMA	Adresse	Jump on Minus
0010	JMP	Adresse	Jump to Address
0011	JSR	Adresse	Jump Subroutine
0100	------		nicht belegt (For Future Use)
0101	RAL		Rotate A Left
0110	INP		Input Instruktion
0111	OUT		Output Instruktion
1000	NOT		Komplementbildung
1001	LDA	Adresse	Load A from Memory Address
1010	STA	Adresse	Store A to Memory Address
1011	ADD	Adresse	Add Operand at Memory Address to A
1100	XOR	Adresse	Exor Operand at Memory Address with A
1101	AND	Adresse	AND Operand at memory Address with A
1110	IOR	Adresse	OR Operand at Memory Address with A
1111	NOP		No Operation

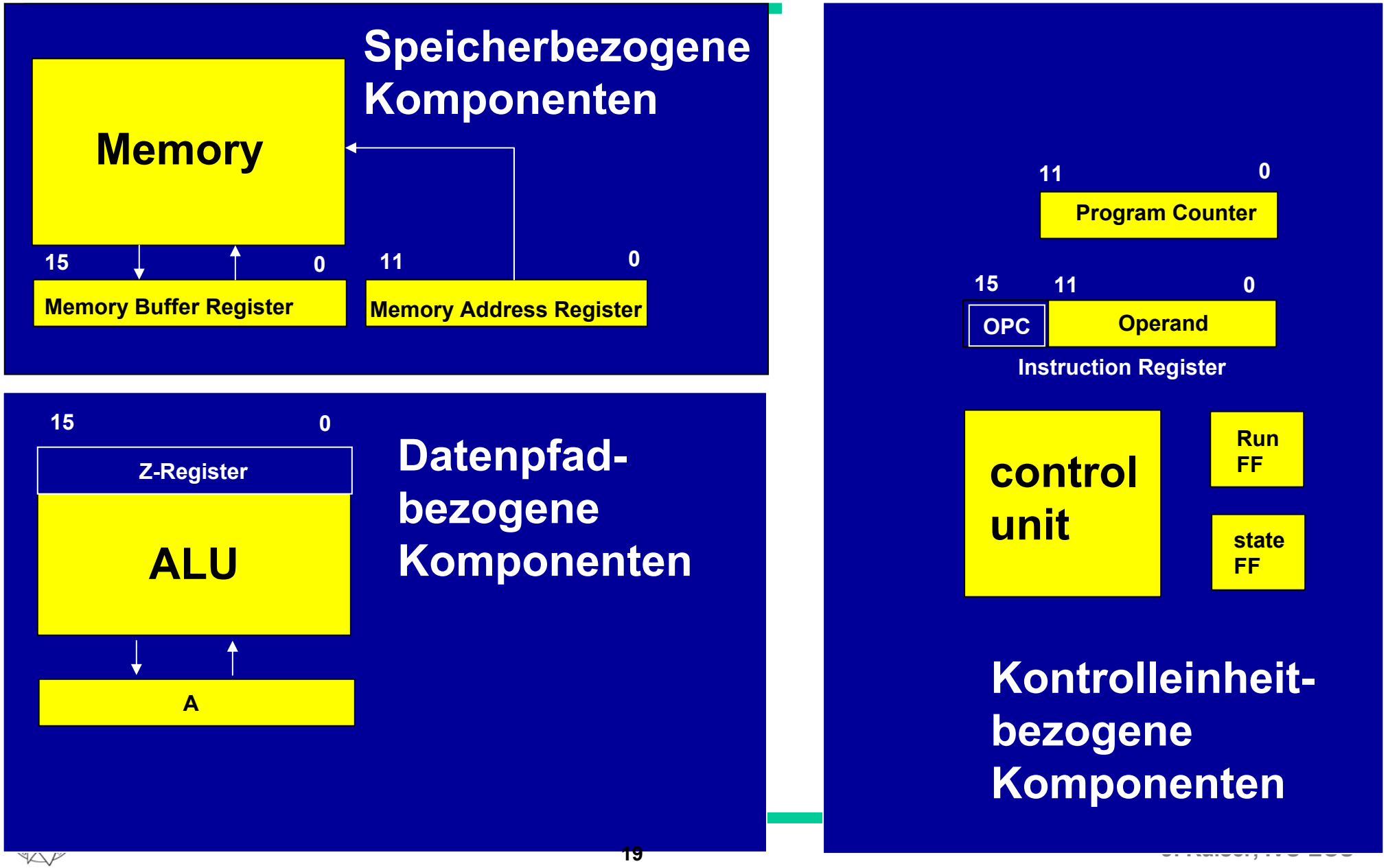
OPC	Mnem.	Operand	Beschreibung
0000	HLT		HLT hält den Computer an. Kann auch manuell eingegeben werden. Nach HLT kann der Rechner nur manuell gestartet werden. Fortsetzung beim nächsten Befehl.
0001	JMA	addr	(Jump on Minus) Bedingter Sprung. Wenn das Ergebnis einer Berechnung negativ ist. Die Adresse "addr" wird in den Befehlszähler geladen. Der nächste Befehl wird von "addr" genommen.
0010	JMP	addr	Unbedingter Sprung. Die Adresse "addr" wird in den Befehlszähler geladen. Der nächste Befehl wird von "addr" genommen.
0011	JSR	addr	Unterprogrammsprung. Die Adresse, die im Befehlszähler enthalten ist, wird ins Register A geladen. Die Adresse "addr" wird in den Befehlszähler geladen. Der nächste Befehl wird von "addr" geladen.
0100	---		
0101	RAL		(Rotate A Left) Zyklischer Links-Shift. Der Inhalt von Register A wird um 1 Stelle nach links rotiert. Ringshift : Bit $A_0 \leftarrow$ Bit A_{15}
0110	INP		INPUT
0111	OUT		OUTPUT
1000	NOT		Komplementbildung des Inhalts von Register A.
1001	LDA	addr	Laden des Registers A von Speicheradresse addr.
1010	STA	addr	Speichern des Registerinhalts auf Speicheradresse addr .
1011	ADD	addr	Inhalt des Speicherwortes mit Adresse adr wird auf den Inhalt des Registers A addiert. Der ursprünglich Inhalt von A wird mit dem Ergebnis überschrieben.
1100	XOR	addr	Inhalt des Speicherwortes mit Adresse adr wird mit dem Inhalt des Registers A durch excl. ODER verknüpft. Der ursprüngliche Inhalt von A wird mit dem Ergebnis überschrieben.
1101	AND	addr	Inhalt des Speicherwortes mit Adresse adr wird mit dem Inhalt des Registers a durch log. UND verknüpft. Der ursprünglich Inhalt von A wird mit dem Ergebnis überschrieben.
1110	IOR	addr	Inhalt des Speicherwortes mit Adresse adr wird mit dem Inhalt des Registers A durch log. ODER verknüpft. Der ursprünglich Inhalt von A wird mit dem Ergebnis überschrieben.
1111	NOP		(No Operation) Der Befehlszähler wird um 1 erhöht.

Phasen der Befehlsausführung



1. Befehl holen
2. Befehl dekodieren
3. Operanden bereitstellen
4. Befehl ausführen
5. Ergebnis speichern

Vom Programmiermodell ..... zum funktionsfähigen Rechner



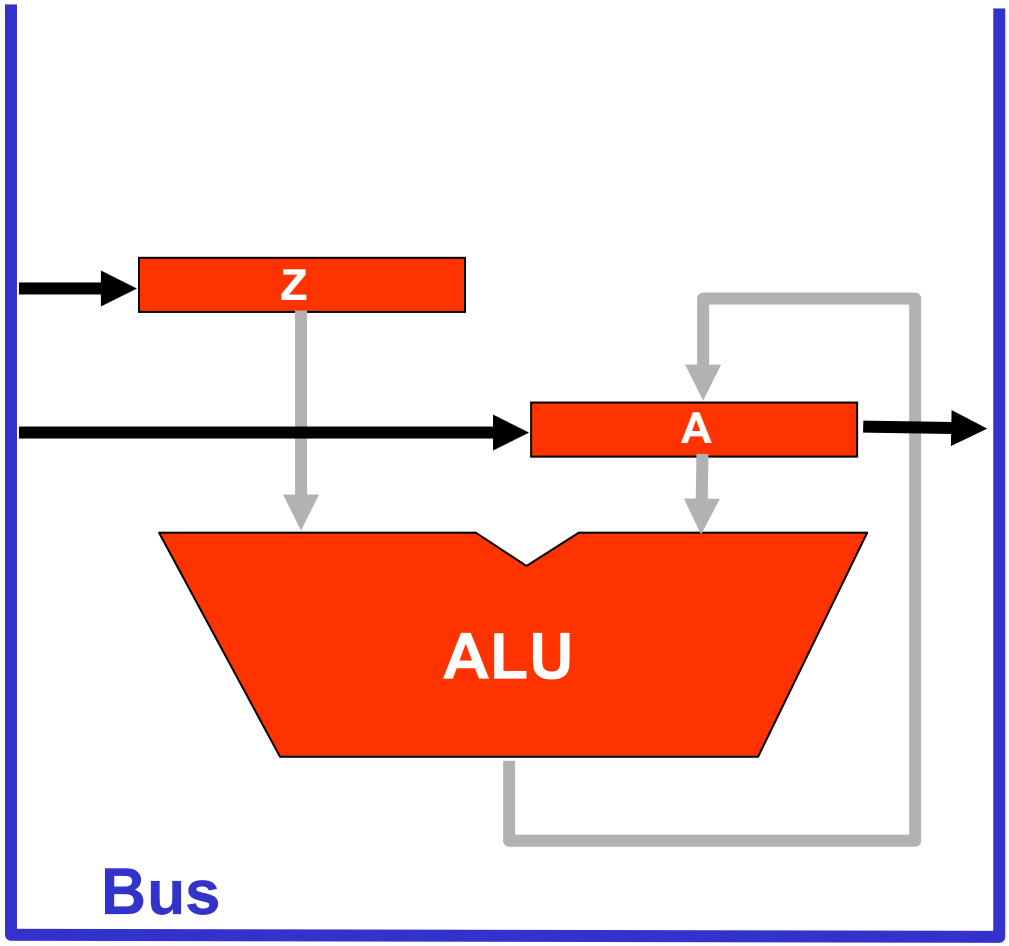
Datenpfad

Durchführen einer arithm. oder logischen Operation:

Ausgangssituation:
1. Operand steht in Register A
2. Operand steht im Speicher

Ziel:
Resultat steht in Register A

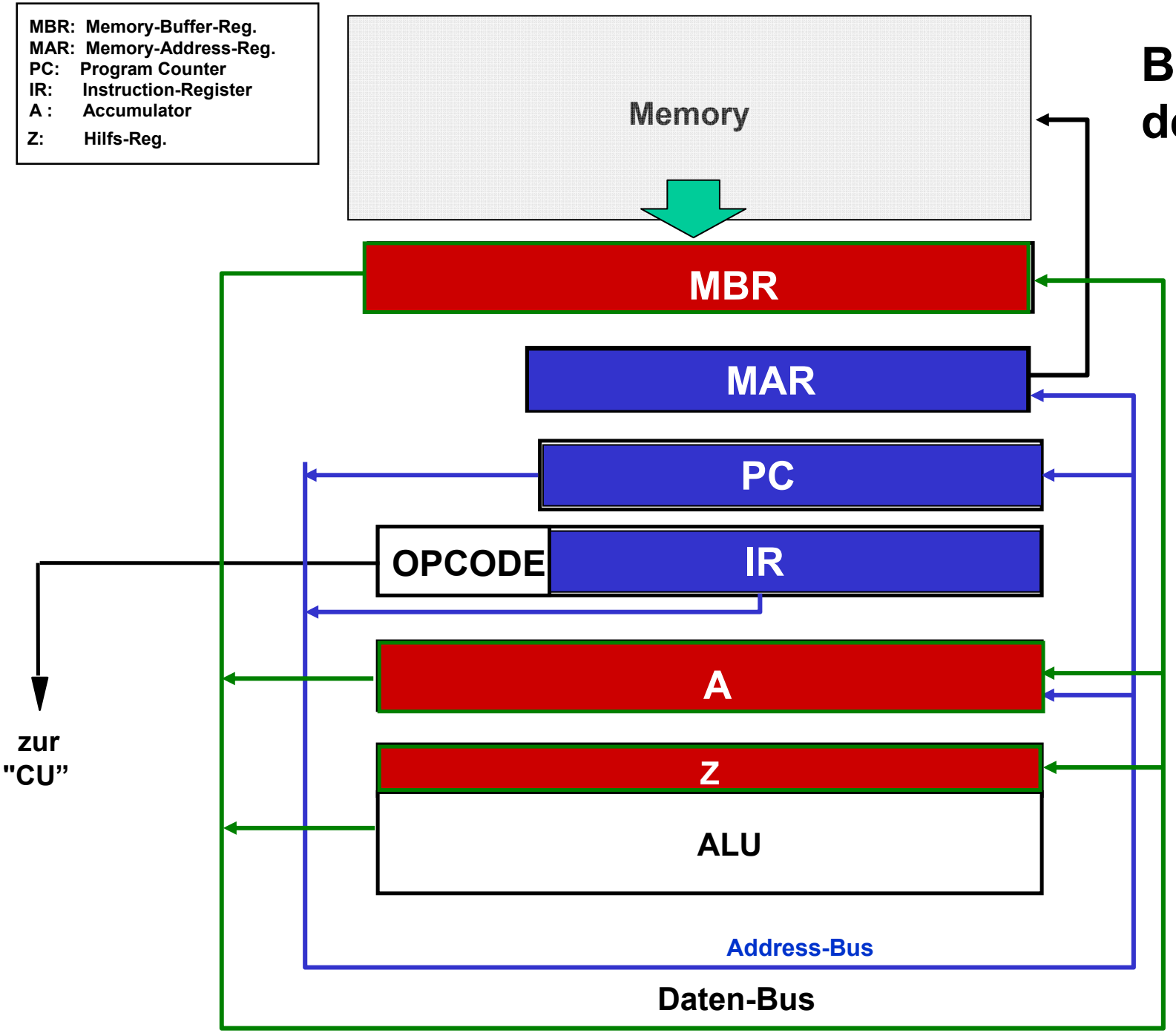
Ablauf:
- **Transferiere 1. Operanden von A nach Z**
- **Lade 2. Operanden nach A**
- **Führe ALU-Operation aus**
- **Transferiere Ergebnis nach A**



A: sichtbares Register
Z: Hilfsregister

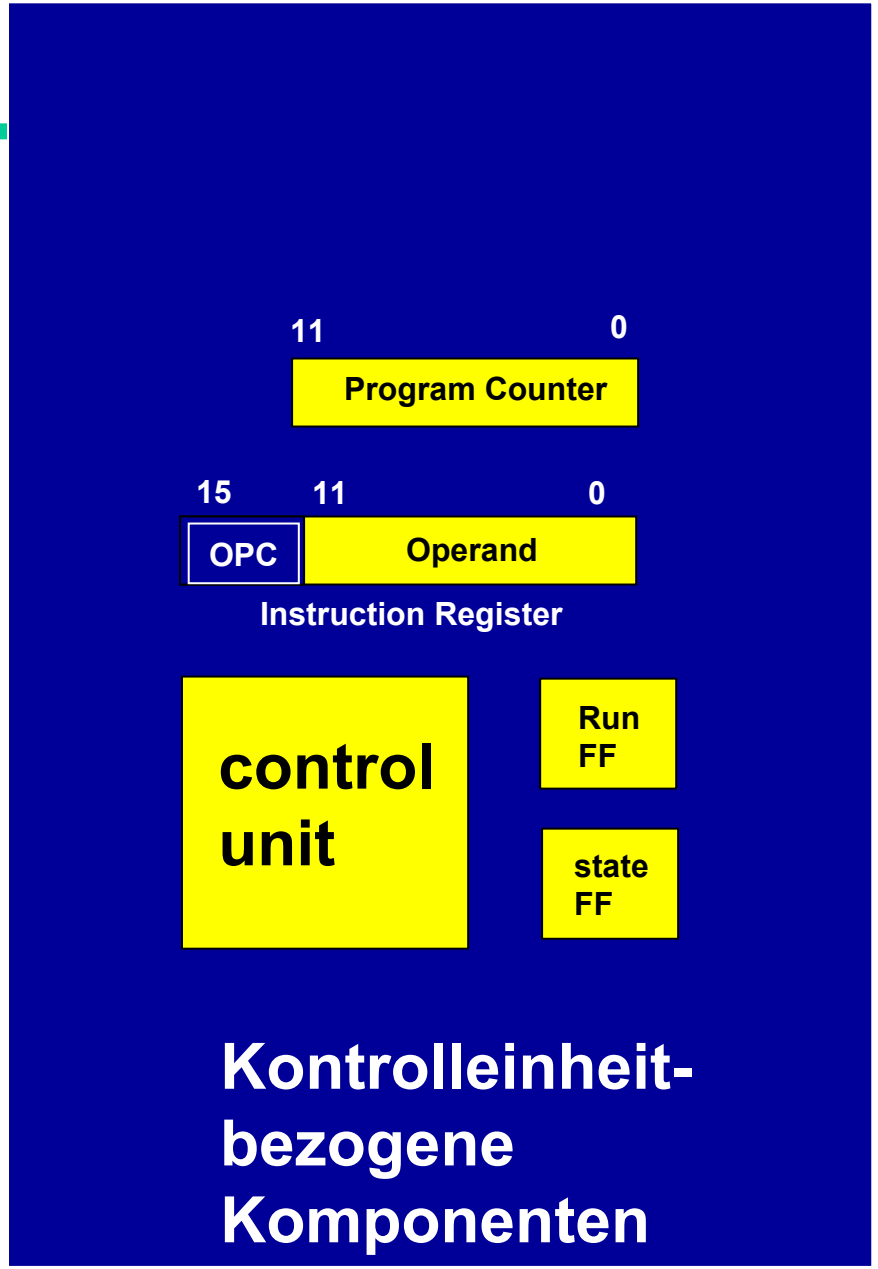
Bus-Struktur des Datenpfads

MBR: Memory-Buffer-Reg.
MAR: Memory-Address-Reg.
PC: Program Counter
IR: Instruction-Register
A : Accumulator
Z: Hilfs-Reg.

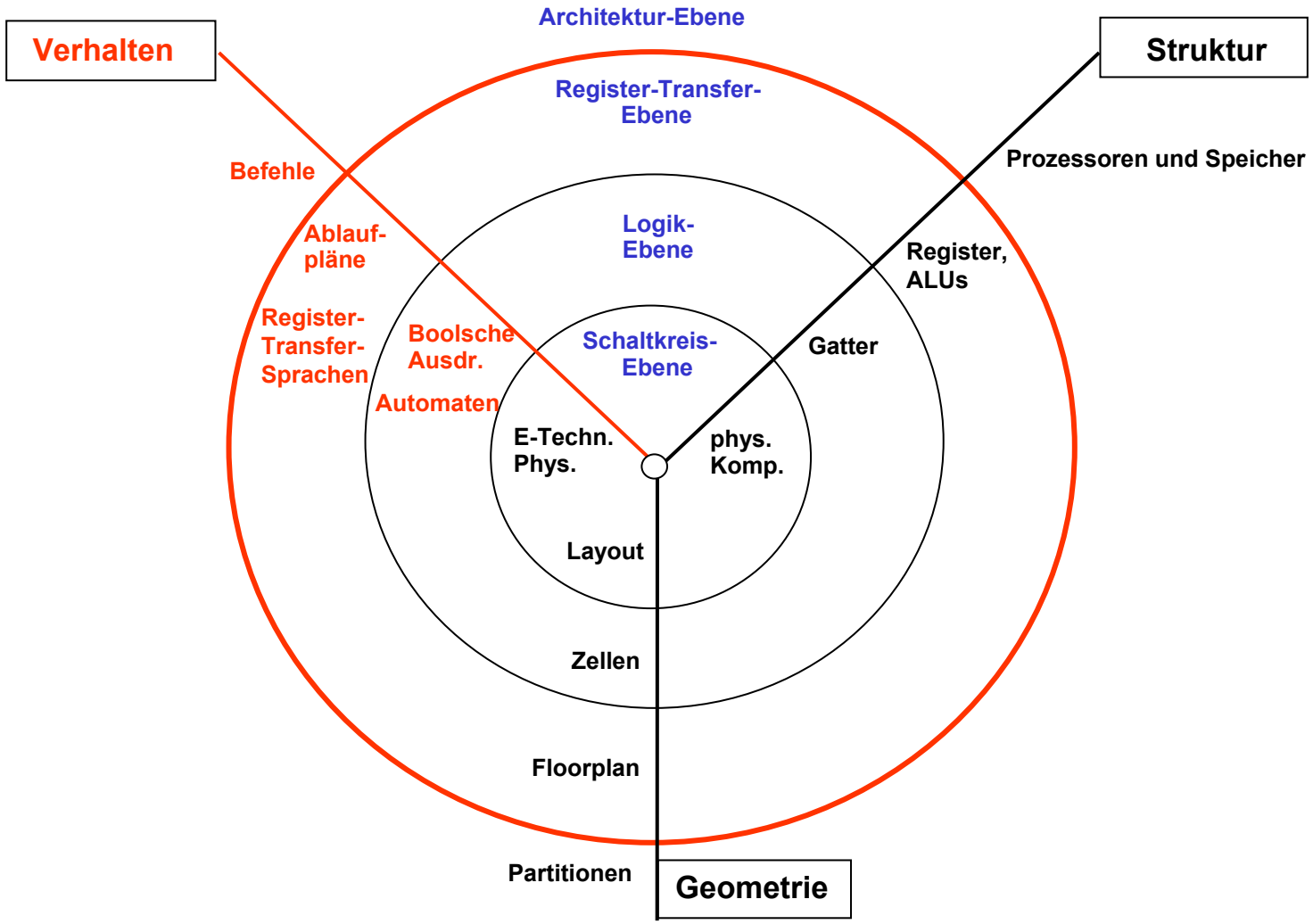


Wie setzt man die Ablaufpläne in eine sequentielle Kontrolle um

?



Wie beschreibt man das Verhalten auf der RT-Ebene?



Die RTL Notation (Register Transfer Language)

Grundelemente von RTL

Register	$\mathbf{R_{n-0}}$	bezeichnet die Bitstellen n,n-1,. . . , 0 des Registers R
Transfer:	←	A ← B bezeichnet den Transfer des Inhalts von Register B nach A
Speicher:	**M[addr]**	bezeichnet den Inhalt der Speicherzelle mit der Adresse "addr"
Bedingungen:	**B:**	z.B. R = 0: A ← B, CP7 • RAL: A ← Z

Die RTL Notation : Beispiele

$a \leftarrow b$	Inhalt von Register b wir nach Register a transferiert.
$a_{3-0} \leftarrow b_{7-4}$	Inhalt der Stellen 4-7 von Register b werden auf die Stellen 0-3 in Register a übertragen.
$a \leftarrow SUM(a,b)$	Die Summe aus den Registerinhalten von a und b wird nach a übertragen
$a \leftarrow SUM(a,1)$	Zum Registerinhalt von a wird 1 addiert. Das Ergebnis wird nach a übertragen.
$a \leftarrow a+b$	Auf die Registerinhalte von a und b wird der Operator "+" angewandt. Das Ergenbis wird nach a übertragen.
$a \leftarrow M[567]$	Speicherwort an der Adresse 567 wird in Register a transferiert.
$R = 0$: $A \leftarrow B$	Wenn die Bedingung R=0 gilt, wird der Inhalt von Register B nach A übertragen.
$C_n \bullet$ CLR: $a \leftarrow 0, b \leftarrow 0, c \leftarrow 0$	Wenn der Takt C_n anliegt und der Befehl "CLR", werden die Register a, b, c auf 0 gesetzt .
$C_n \bullet$ HLT: Run-FF $\leftarrow 0$	Wenn der Takt C_n anliegt und der Befehl "HLT" wird das Run-FF auf 0 gesetzt.
$C_n \bullet$ JMP: $PC \leftarrow IR_{11-0}$	Wenn der Takt C_n anliegt und der Befehl "JMP" wird der Inhalt des Operandenfelds des IR in den PC transferiert.

Ausführungsphasen

Befehlsholphase (Speicherzugriff)

Befehlsdekodierung und Operandenholphase

Ausführung benötigt keine Operanden

Beisp. Shift-Op.s
Sprünge, NOP.

Ausführungsphase

Ausführung benötigt zusätzliche Operanden

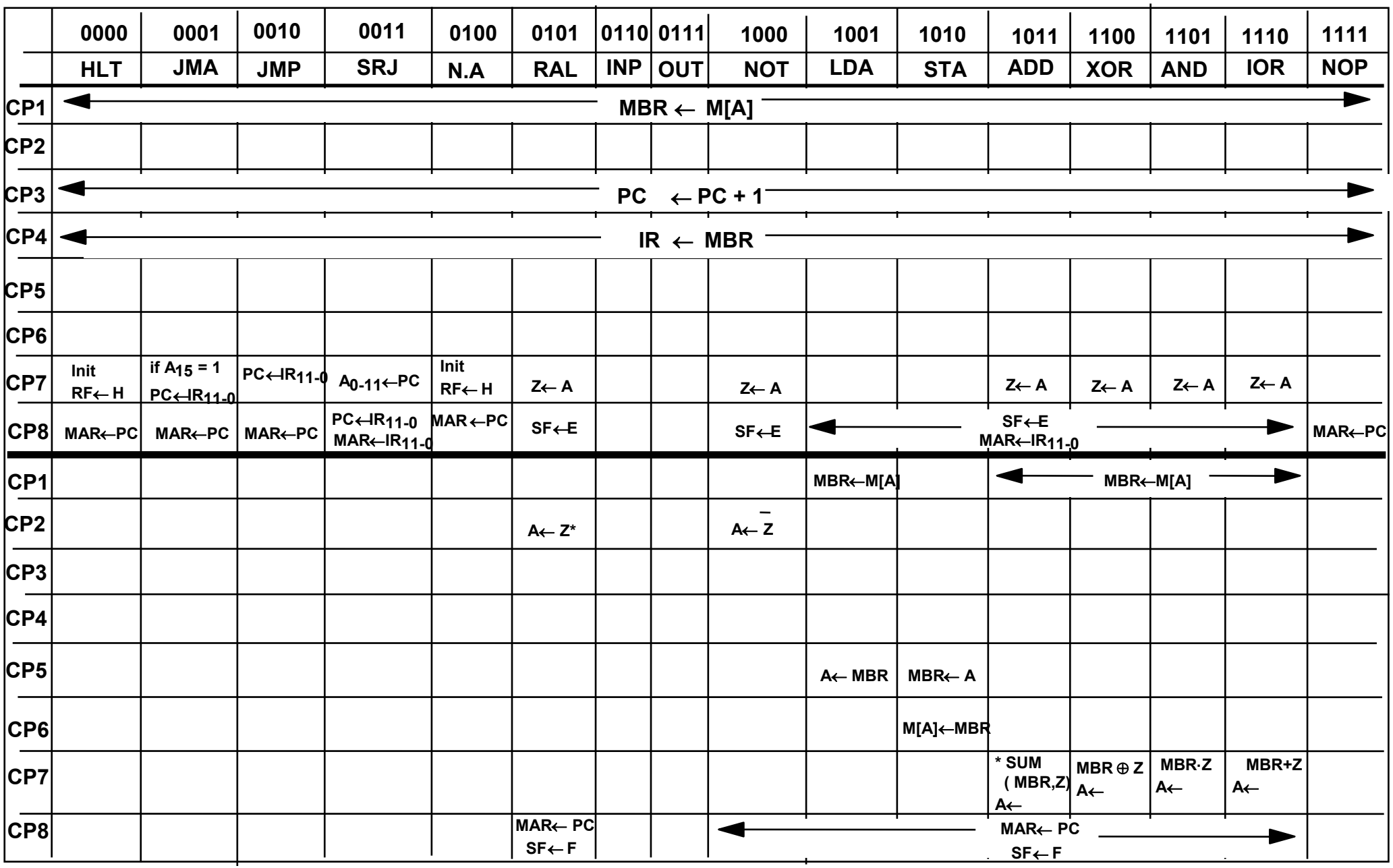
Beisp.: zweistellige arithmetisch/logische Operationen

Operandenholphase
(Speicherzugriff)

Ausführungsphase

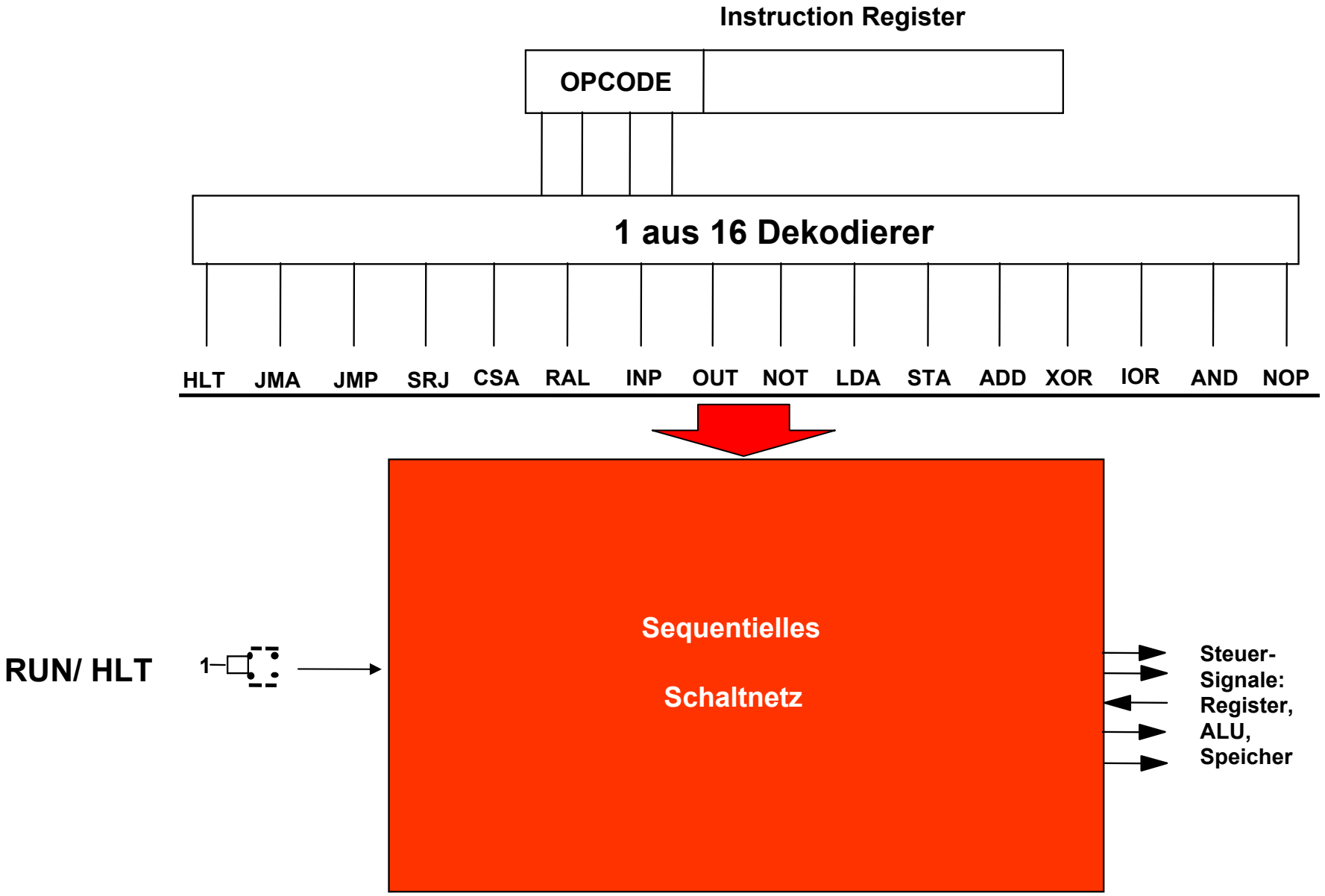
Detaillierter Ablauf und Timing der Maschinenbefehle

SF = F : Fetch Phase
SF = E : Execute Phase

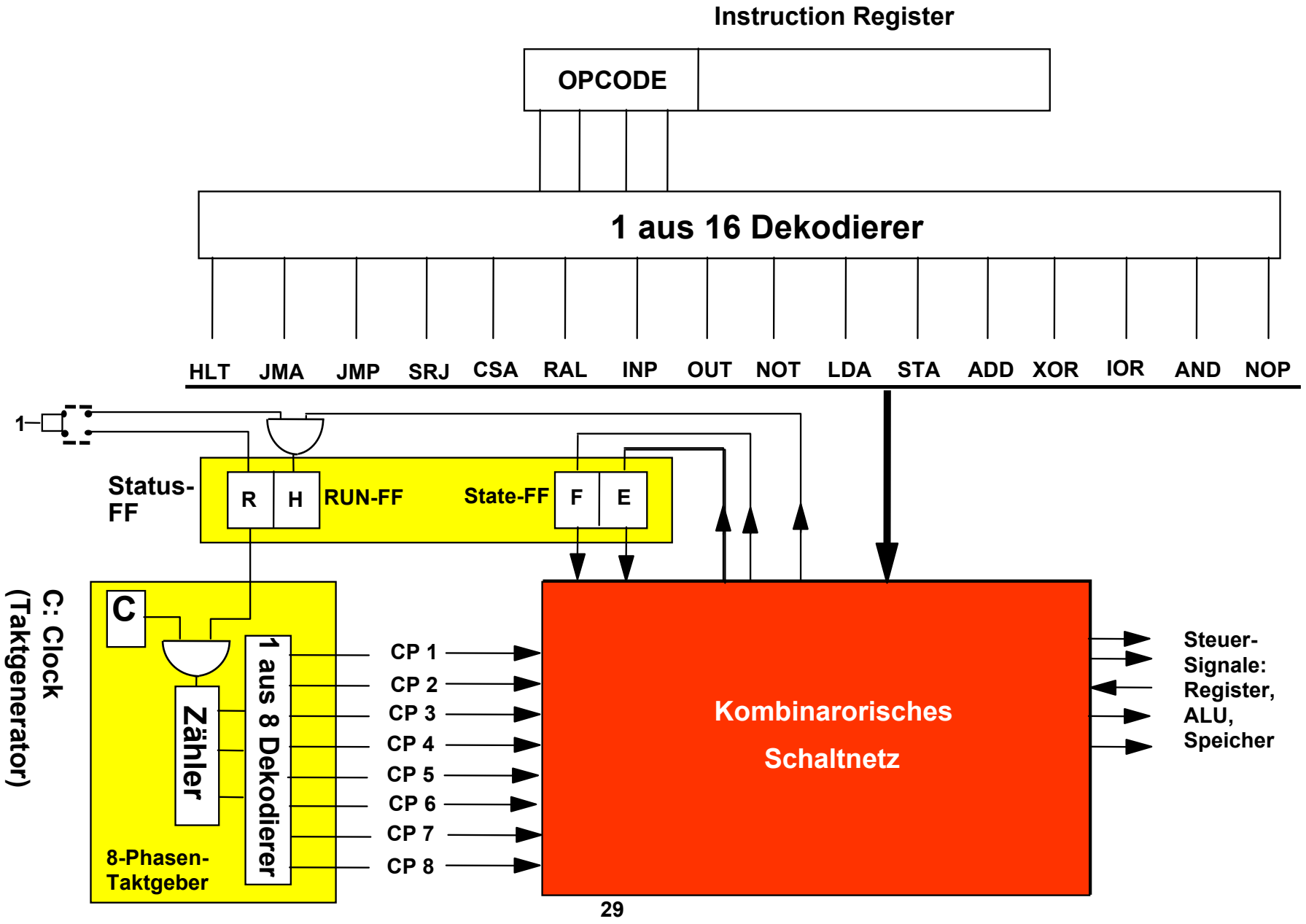


* if C_{15} =1 then RF ← HLT (Overflow)

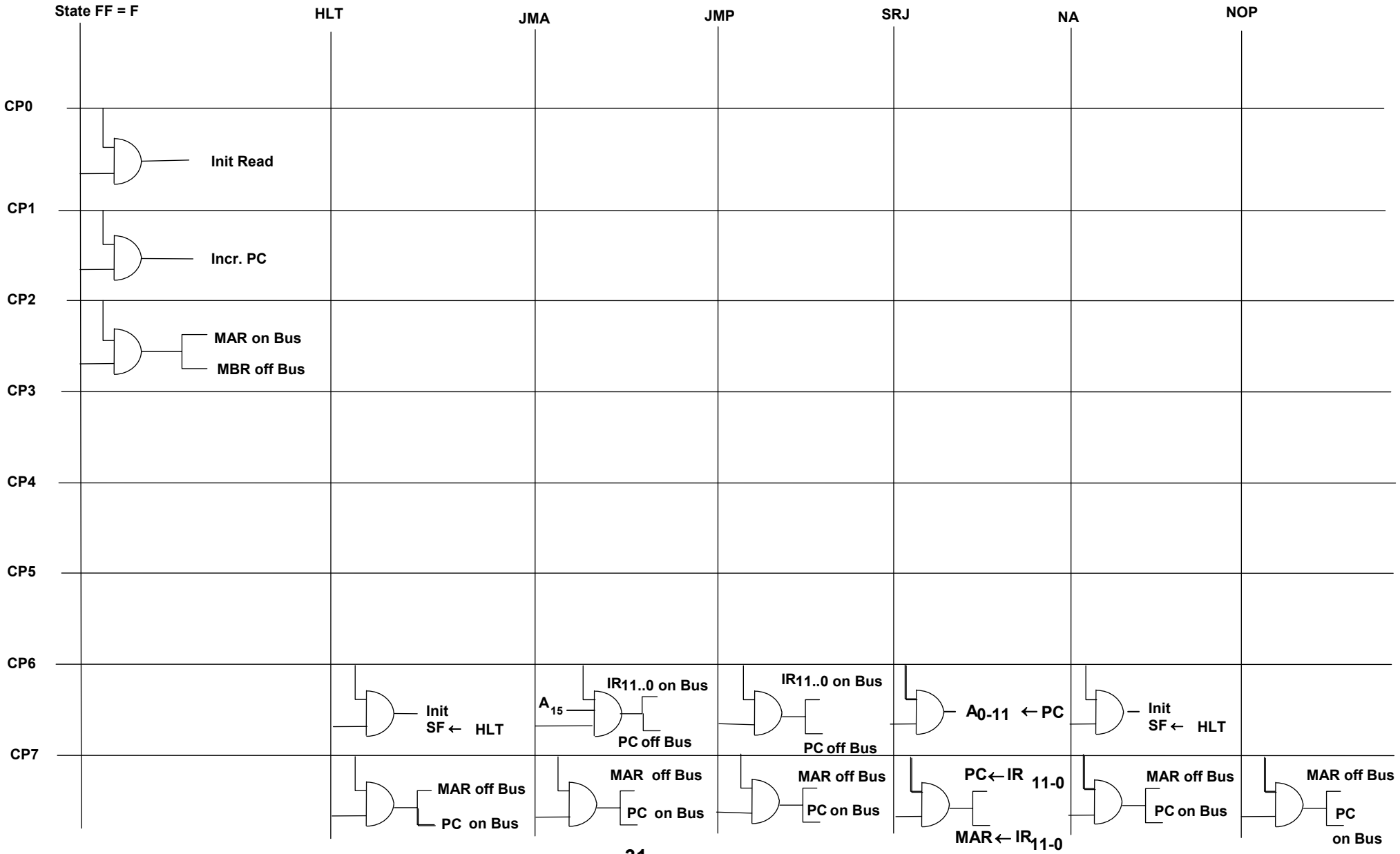
Die Kontrolleinheit des Modellrechners



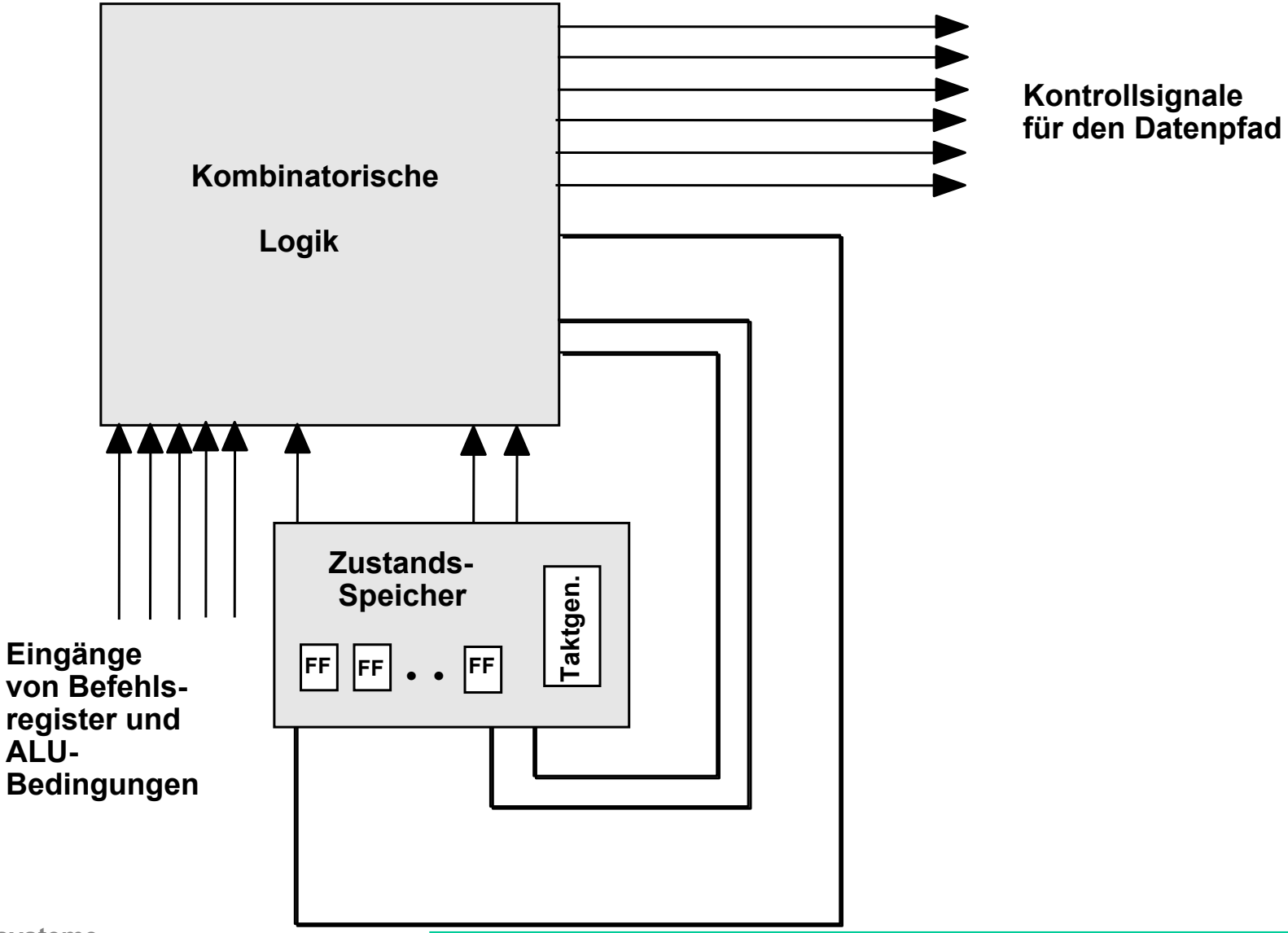
Die Kontrolleinheit des Modellrechners



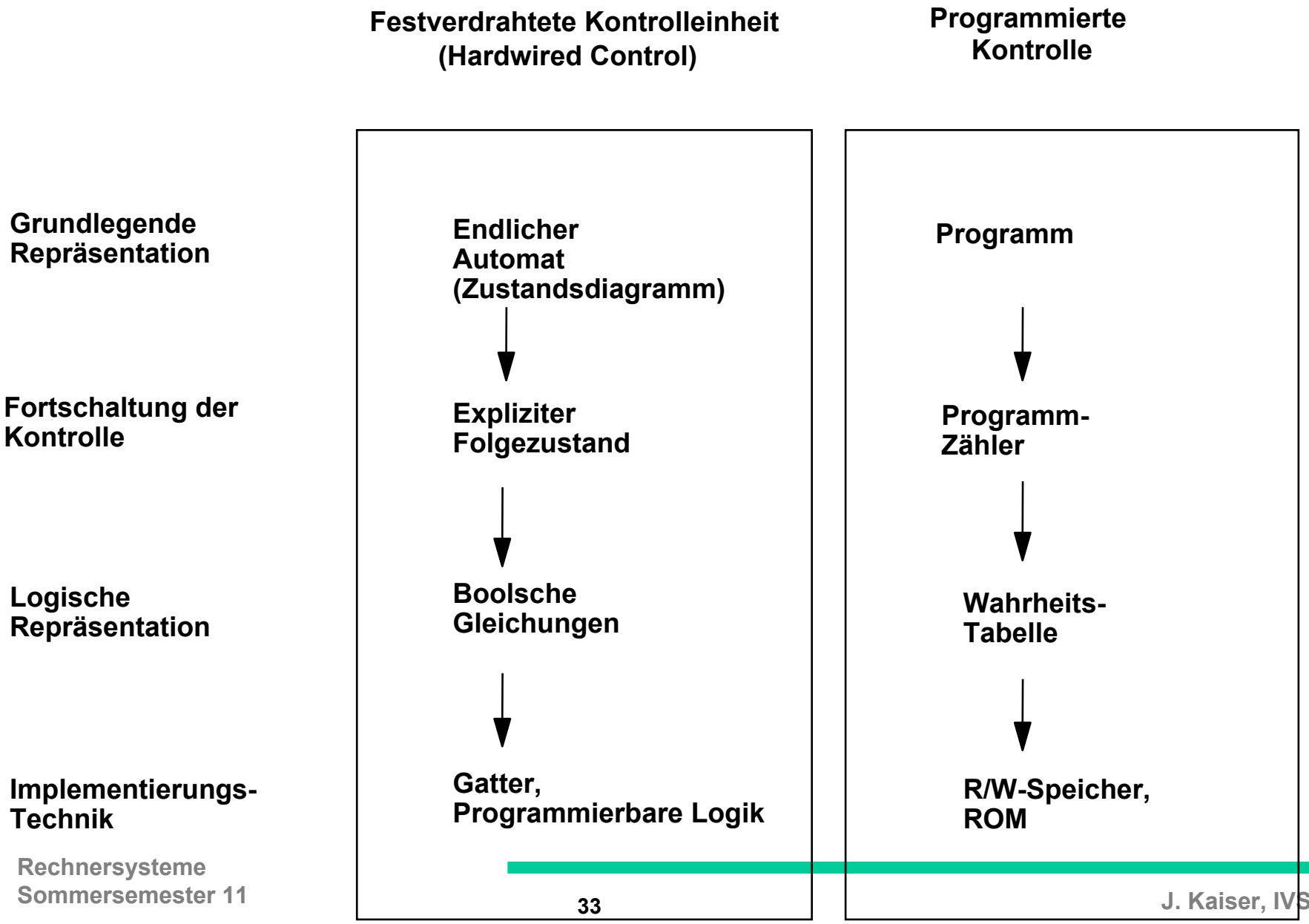
1-Zyklus Befehle



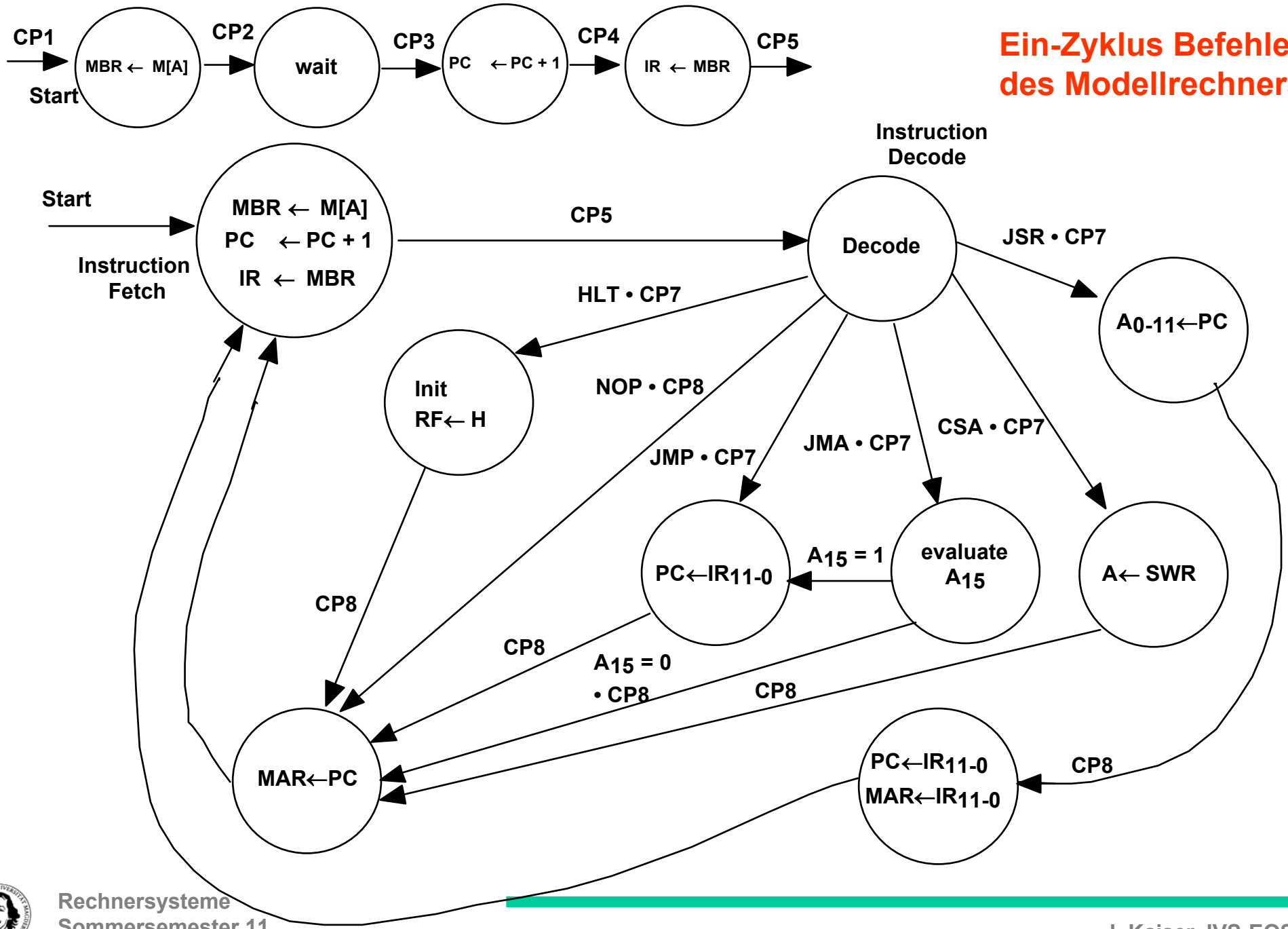
Festverdrahtete Zustandsmaschine zur Realisierung der Kontrolleinheit



Alternativen zur Spezifikation und Realisierung der Kontrolleinheit



Ein-Zyklus Befehle des Modellrechners

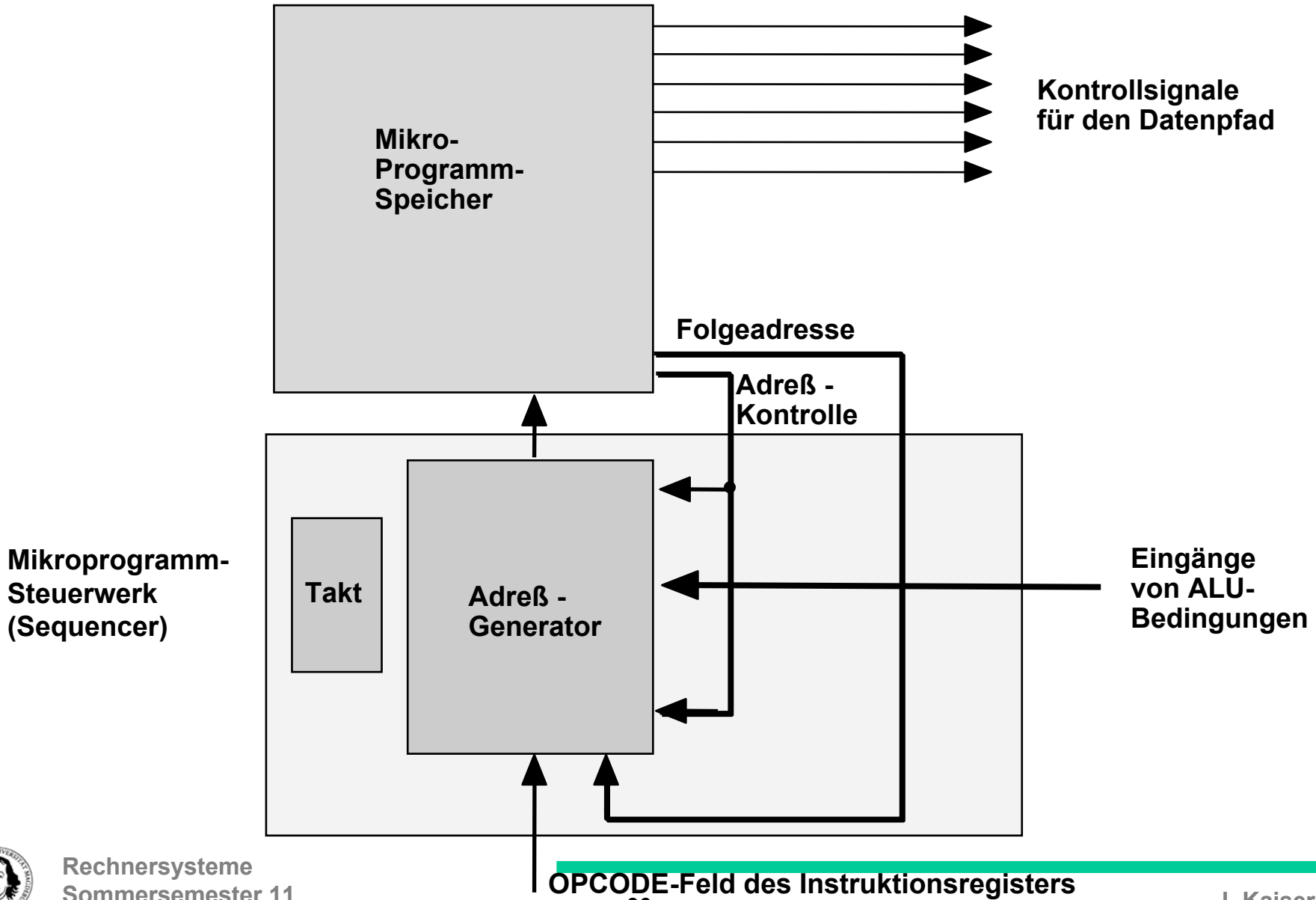


Idee der Mikroprogrammierung

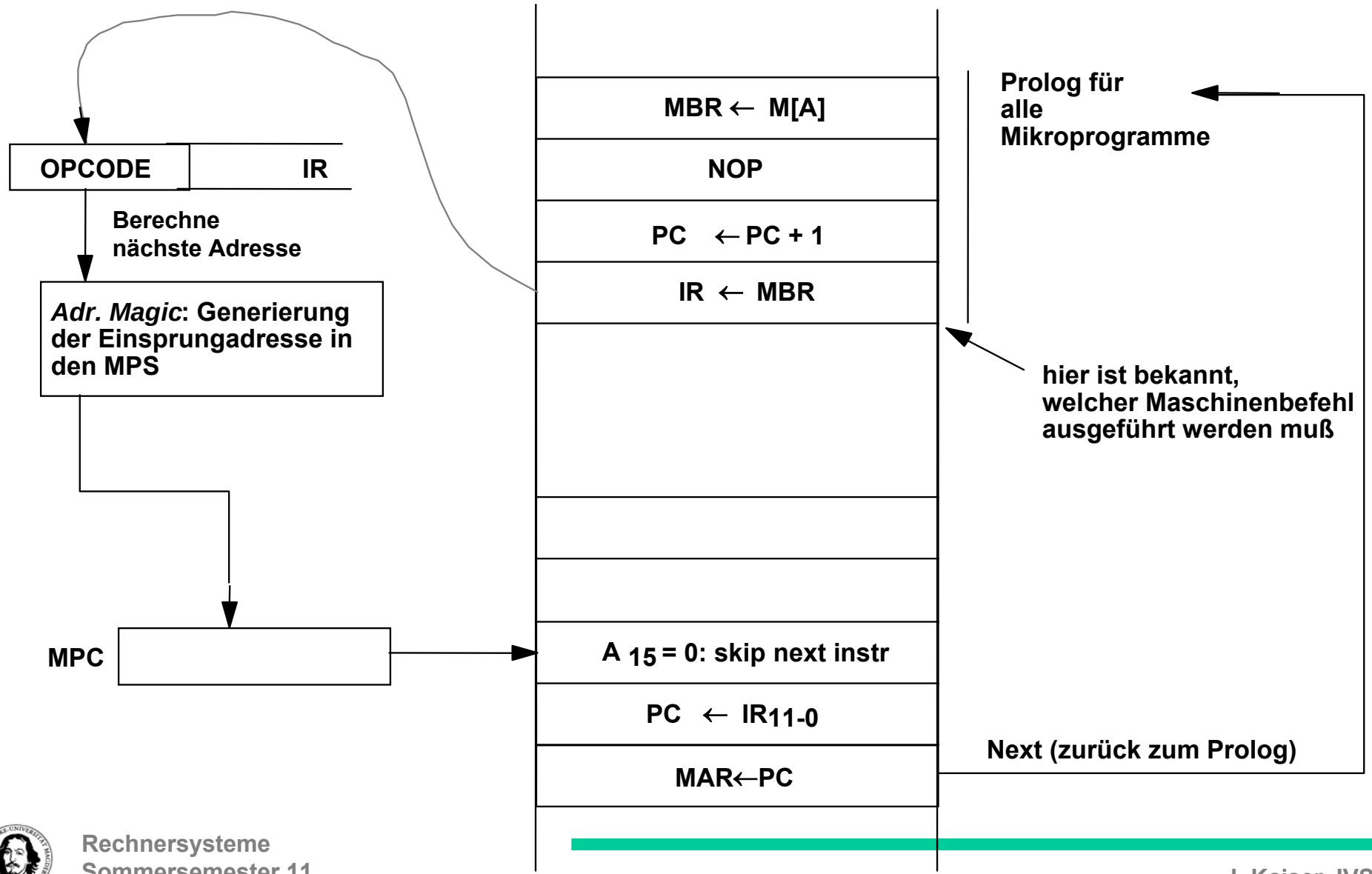
... I realized that the solution was to turn the control unit into a computer in miniature by adding a second matrix to determine the flow of control at the microlevel .....

Maurice Wilkes,
Memories of a Computer Pioneer

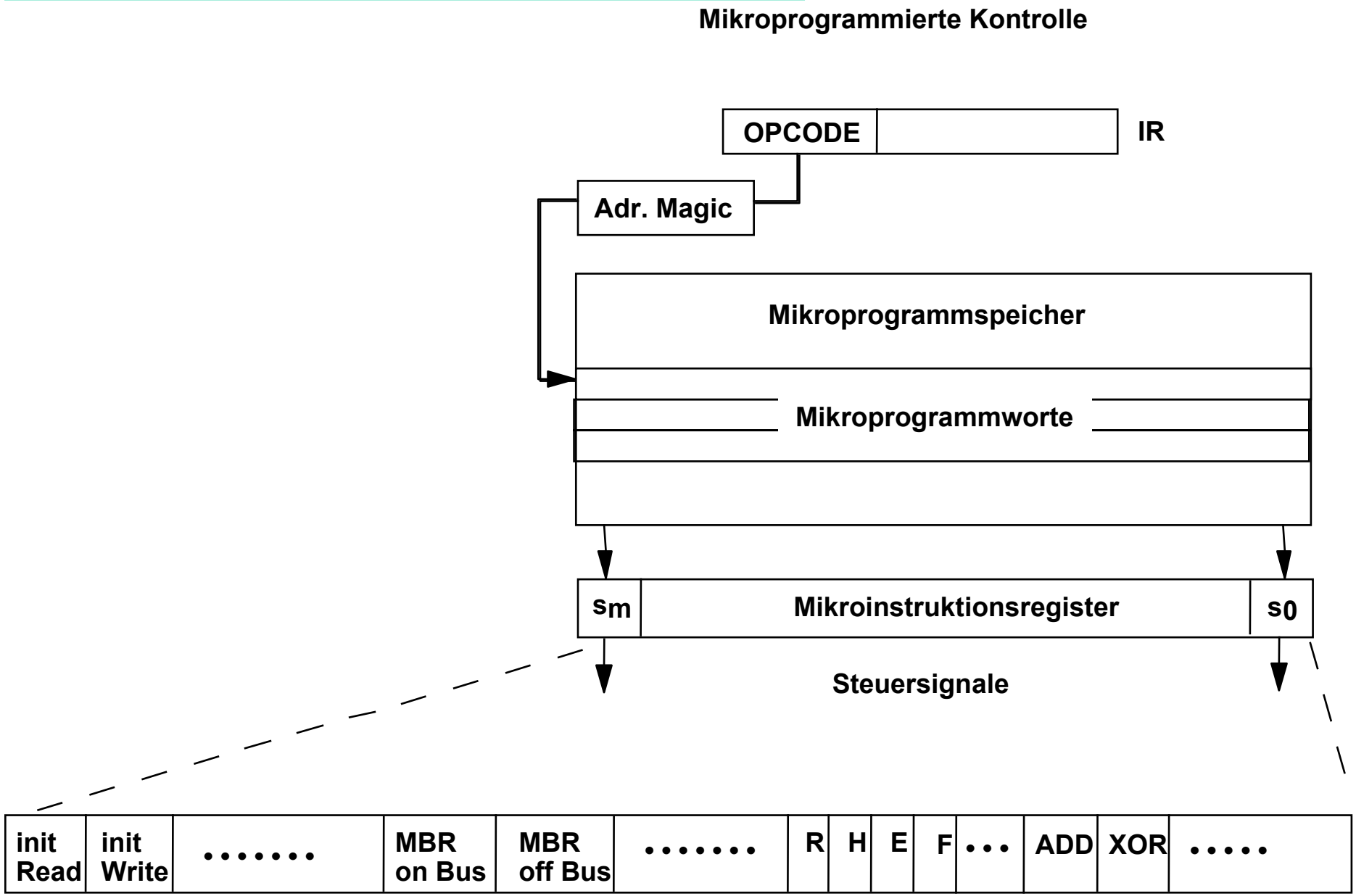
Mikroprogrammierte Kontrolleinheit



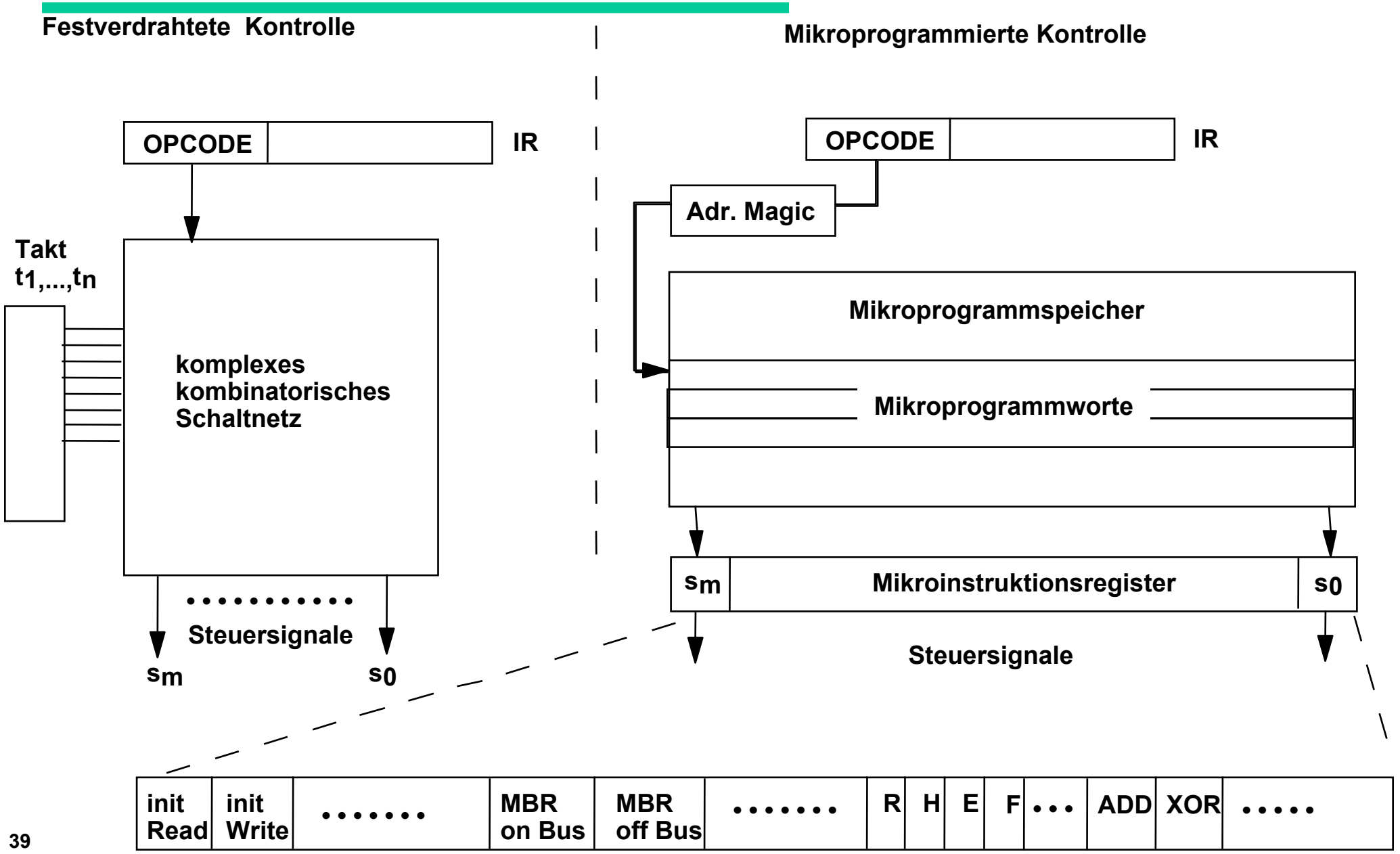
Beispiel für die mikroprogrammierte Implementierung des JMA-Befehls



Alternativen beim Entwurf der Kontrolleinheit



Alternativen beim Entwurf der Kontrolleinheit



Aufgaben des Mikroprogramm-Steuerwerks (Sequencer)

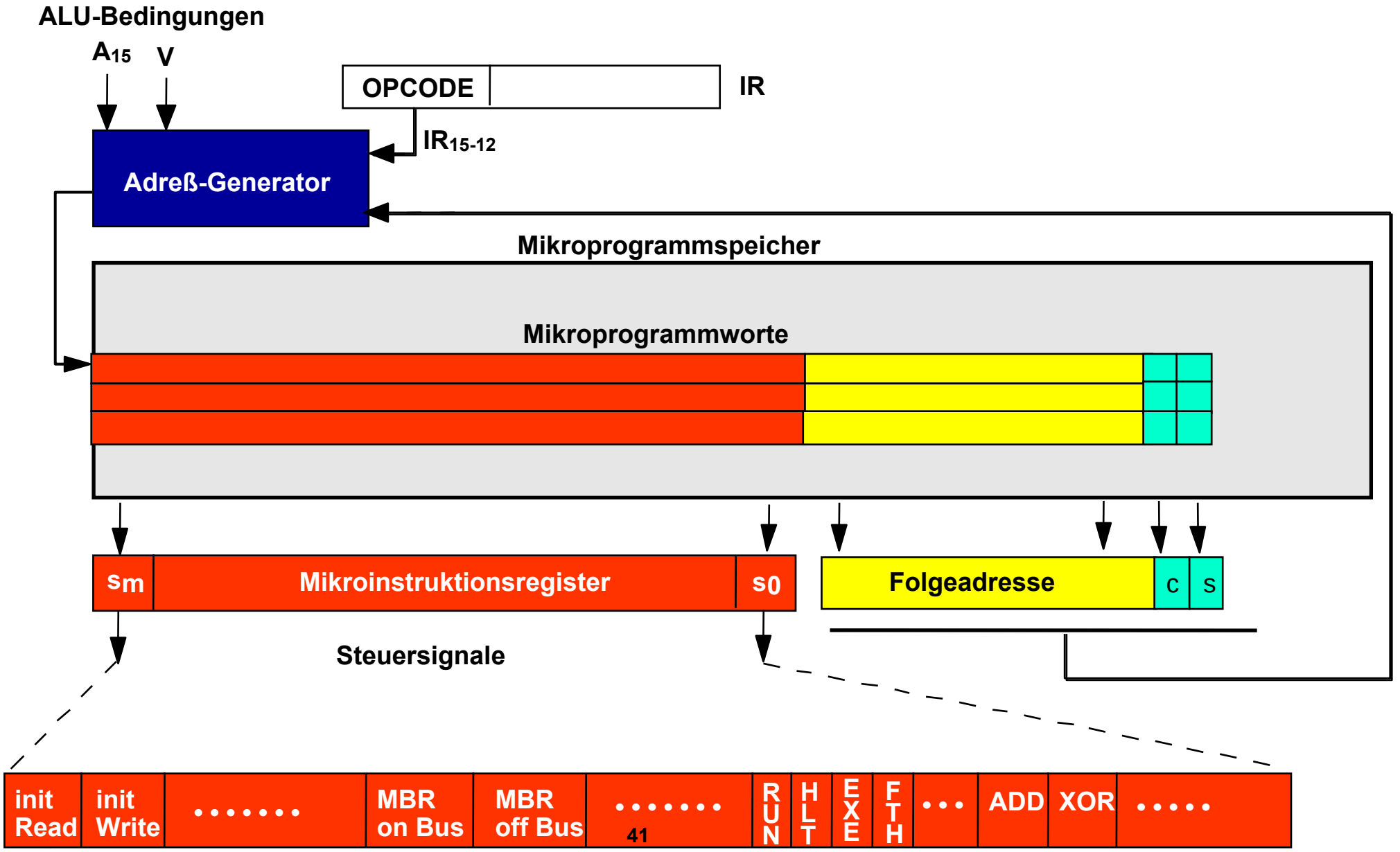
Generierung der "Adresse der nächsten Mikroinstruktion"

"Nur" Adreßberechnung !

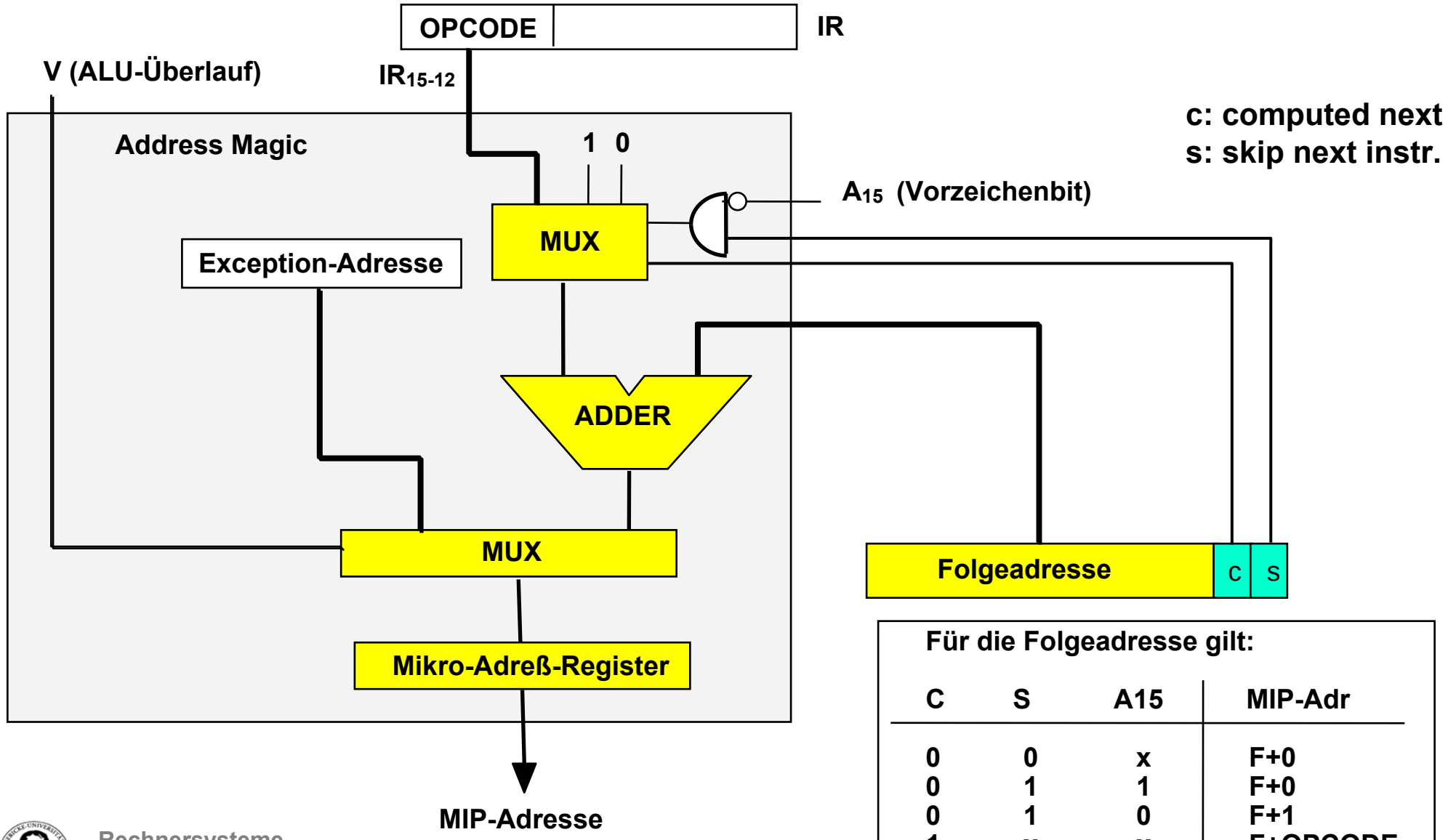
Standardeigenschaften:

- **unbedingte Adreßfortschaltung**
- **bedingte Adreßfortschaltung**

Mikroprogrammierte Kontrolle



Adressierungseinheit

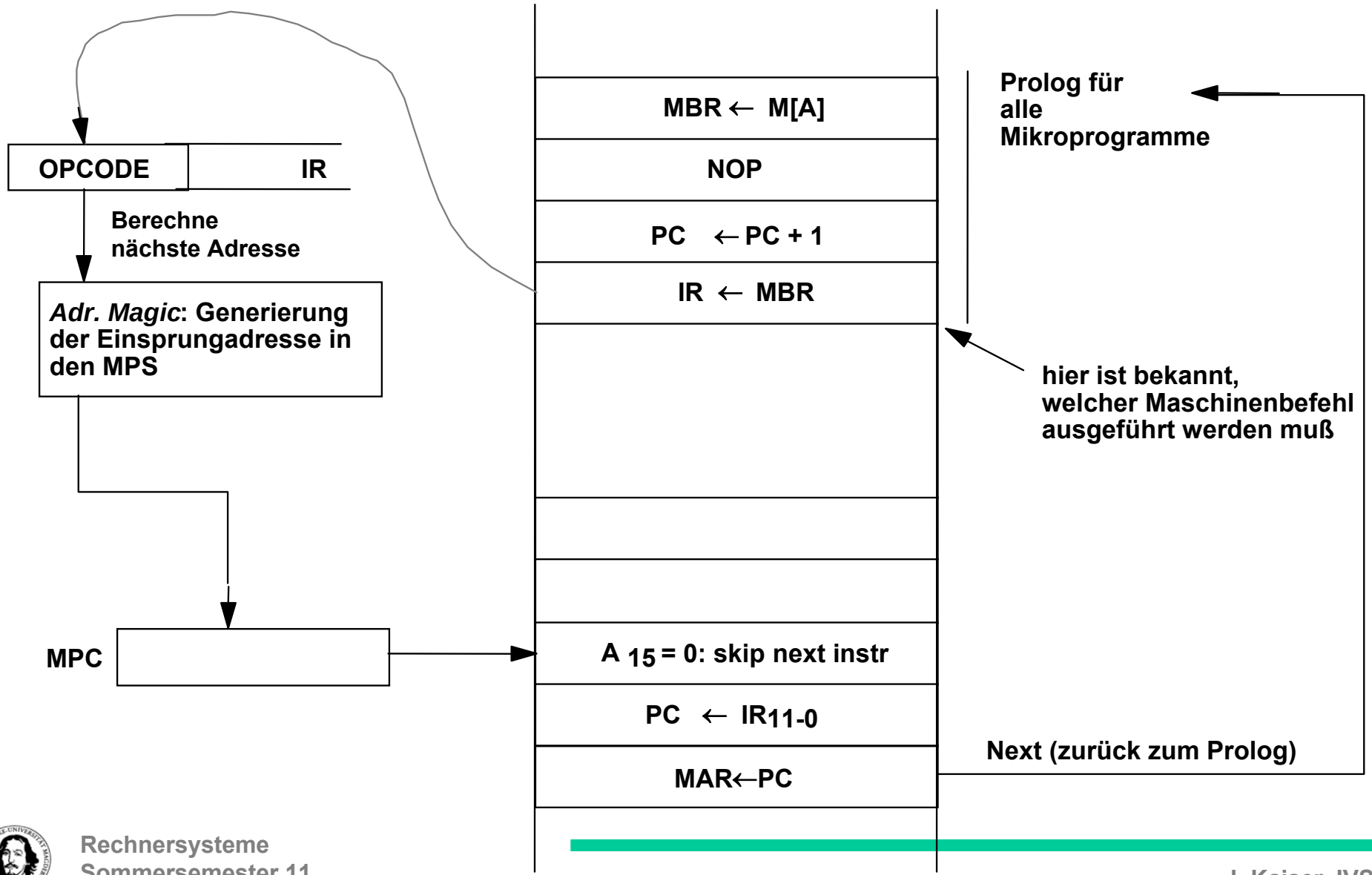


c: computed next
s: skip next instr.

Für die Folgeadresse gilt:

C	S	A15	MIP-Adr
0	0	x	F+0
0	1	1	F+0
0	1	0	F+1
1	x	x	F+OPCODE

Beispiel für die mikroprogrammierte Implementierung des JMA-Befehls

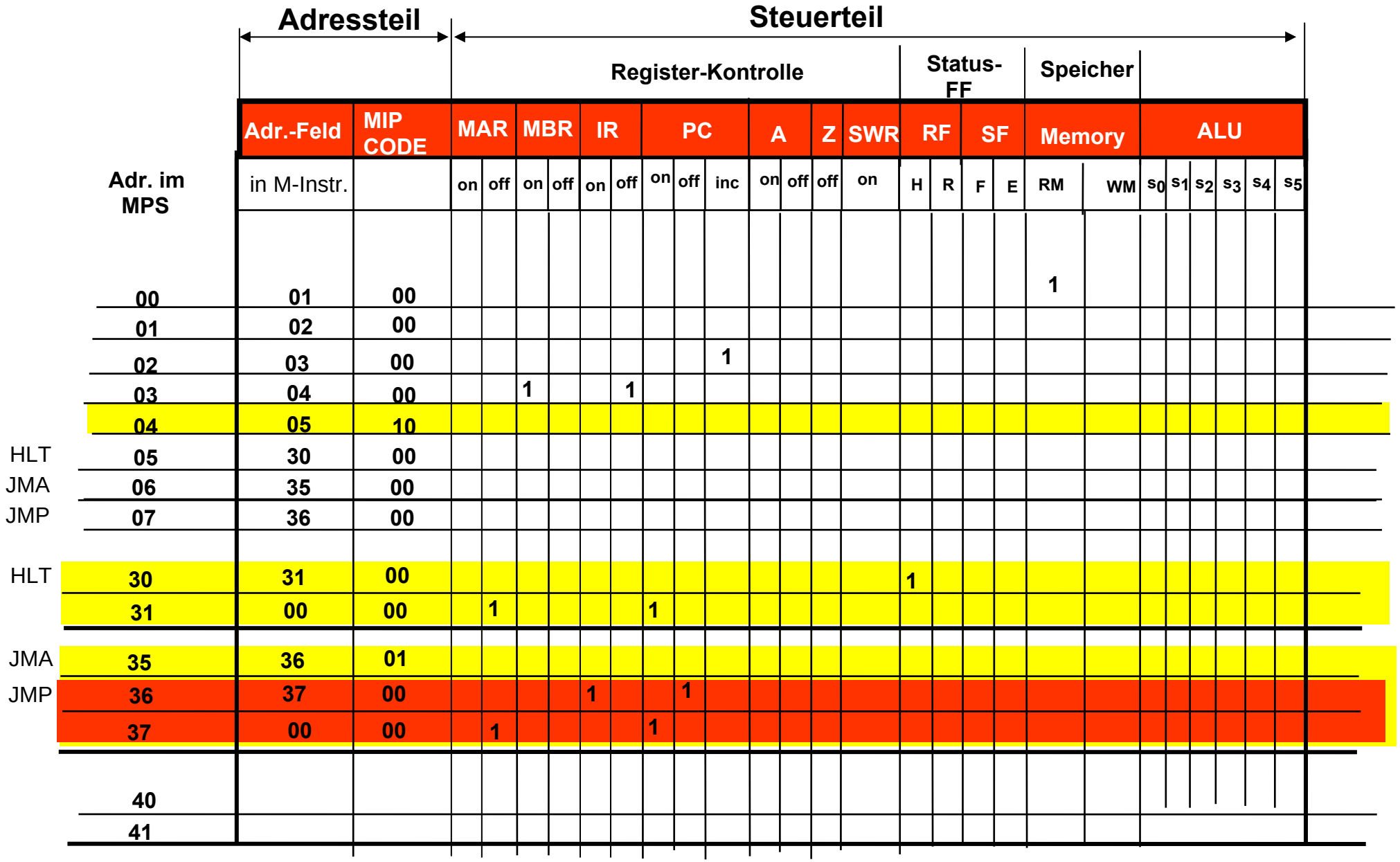


MPS-Adresse	Adreßauswahl c	s	Adreßteil	Steuerteil	Kommentar
00	0	0	01	Init Read	
01	0	0	02		
02	0	0	03	PC← PC+1	increment PC
03	0	0	04	IR ← MBR	IR off Bus, MBR on Bus
04	1	0	05		Folgeadresse = Adreßteil + OP-code
05	0	0	30		Startadresse für HLT
06	0	0	35		Startadresse für JMA
07	0	0	40		Startadresse für JMP
• • •					
30	0	0	31	RF ← H	
31	0	0	00	MAR ← PC	Beendigung von HLT und Rücksprung
32 33 34					
35	0	1	36		if A_{15} = 0 THEN skip
36	0	0	37	PC ← IR_{11-0}	
37	0	0	00	MAR ← PC	Beendigung von JMA und Rücksprung
38					
39					
40	0	0	41	PC ← IR_{11-0}	
41	0	0	00	MAR ← PC	Beendigung von JMP und Rücksprung
• • •					

Layout des Mikroprogrammspeichers

	Adressteil		Steuerteil																									
			Register-Kontrolle													Status-FF				Speicher		ALU						
	Adr.-Feld	MIP CODE	MAR		MBR		IR		PC			A		Z	SWR	RF		SF		Memory		ALU						
Adr. im MPS	in M-Instr.		on	off	on	off	on	off	on	off	inc	on	off	off	on	H	R	F	E	RM	WM	s_0	s_1	s_2	s_3	s_4	s_5	
00	01	00																		1								
01	02	00																										
02	03	00									1																	
03	04	00			1			1																				
04	05	10																										
HLT 05	30	00																										
JMA 06	35	00																										
JMP 07	40	00																										
HLT 30	31	00														1												
31	00	00		1					1																			
JMA 35	36	01																										
36	37	00					1			1																		
37	00	00		1					1																			
JMP 40	41	00					1			1																		
41	00	00		1					1																			

Zusammenfassung von Mikroprogrammen



Horizontale Mikroprogrammierung

Jedes Bit des Mikroinstruktionswortes steuert direkt eine Kontroll-Leitung

Vorteil : **maximale Flexibilität, Geschwindigkeit**

Nachteil : **sehr langes Mikroinstruktionswort, großer Mikroprogrammspeicher**

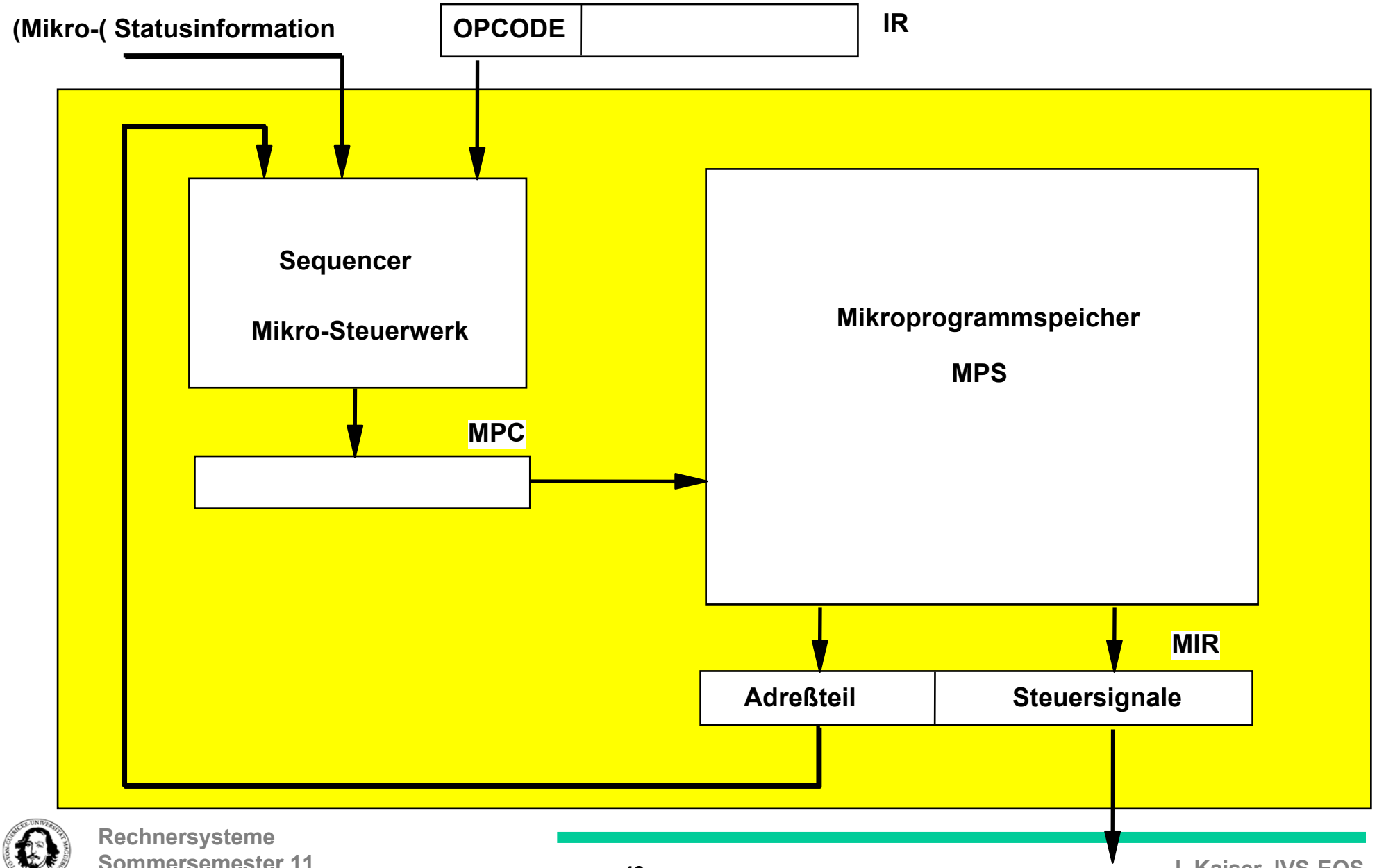
Fragestellungen bei der Mikroprogrammierung

- **Wie kann der Micro-Code optimiert werden?**
  Zielkonflikt: Hardwarekosten gegen Geschwindigkeit

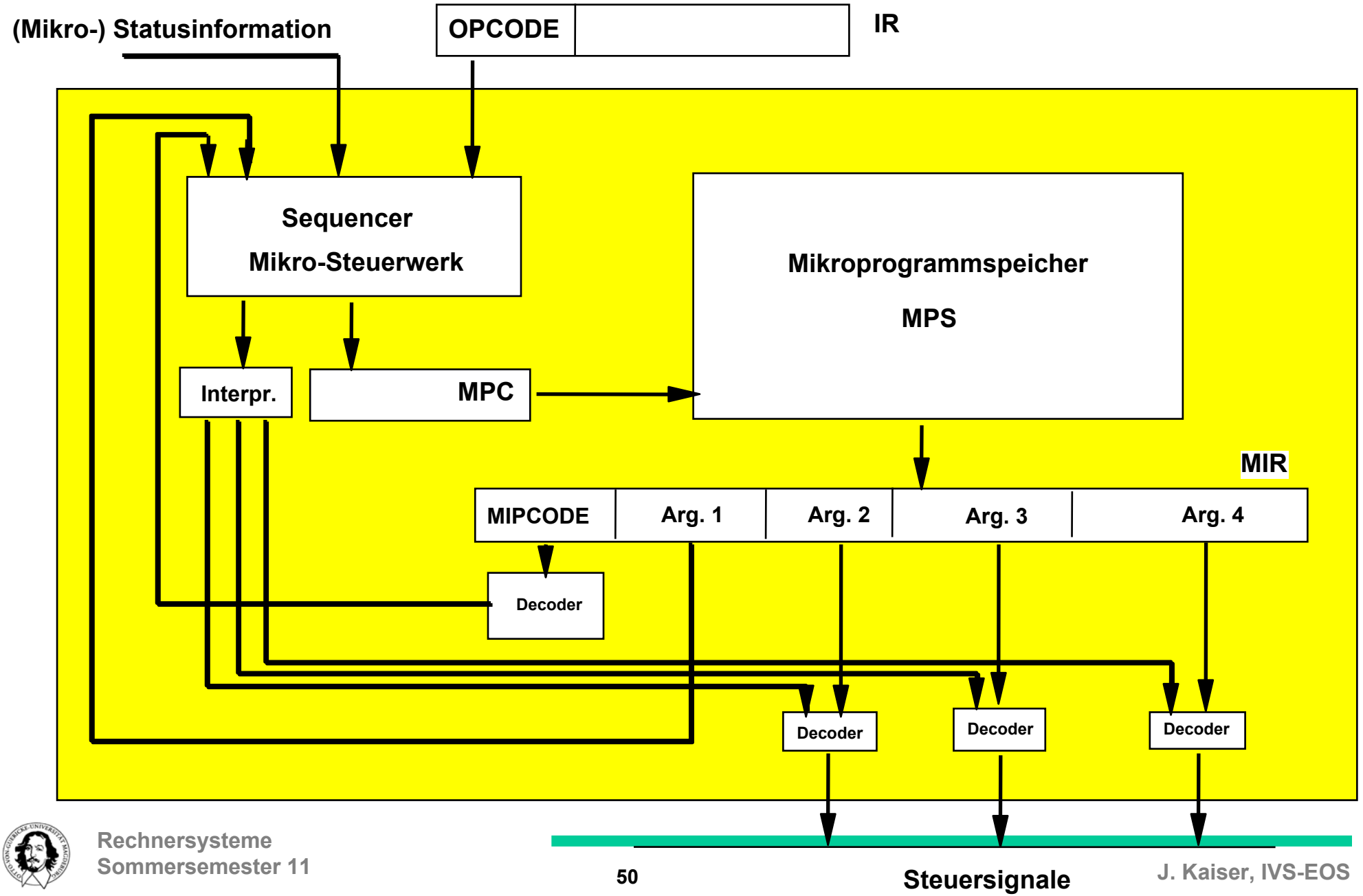
- **Wie kann des Micro-Wort verkürzt und der M-Speicher vekleinert werden?**
  Zielkonflikt: Wortlänge gegen Dekodierungsaufwand und Flexibilität

- **Wie erreicht man größtmöglichen Komfort bei der Programmierung?**
  Zielkonflikt: Komfort gegen Effizienz und Flexibilität

Mikroprogrammierte Kontrolleinheit (horizontale MP)



Mikroprogrammierte Kontrolleinheit mit mehrstufiger Dekodierung



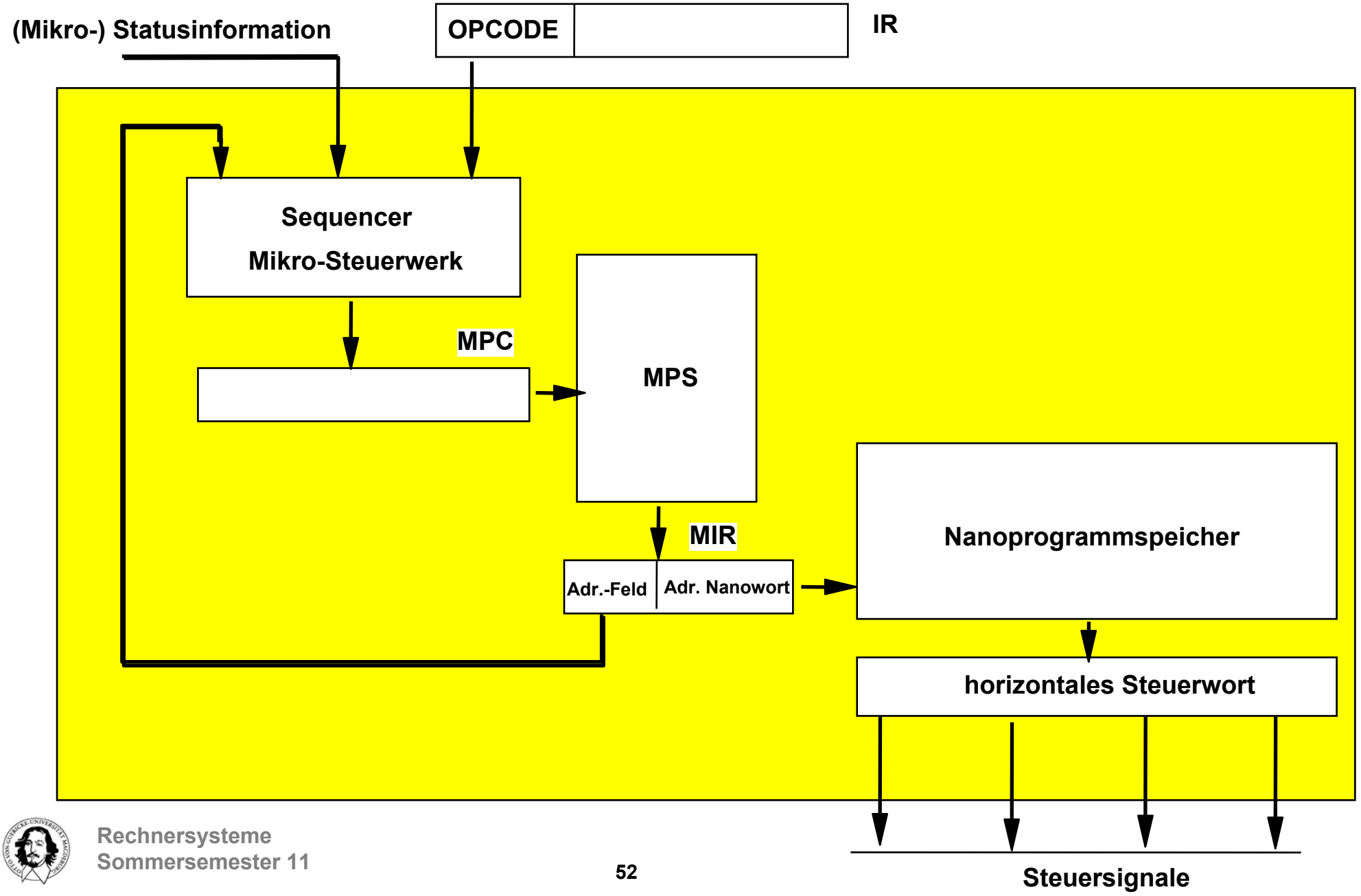
Vertikale Mikroprogrammierung

Mikroinstruktionswort ist in Felder aufgeteilt, die codierte Steuerinformation enthalten

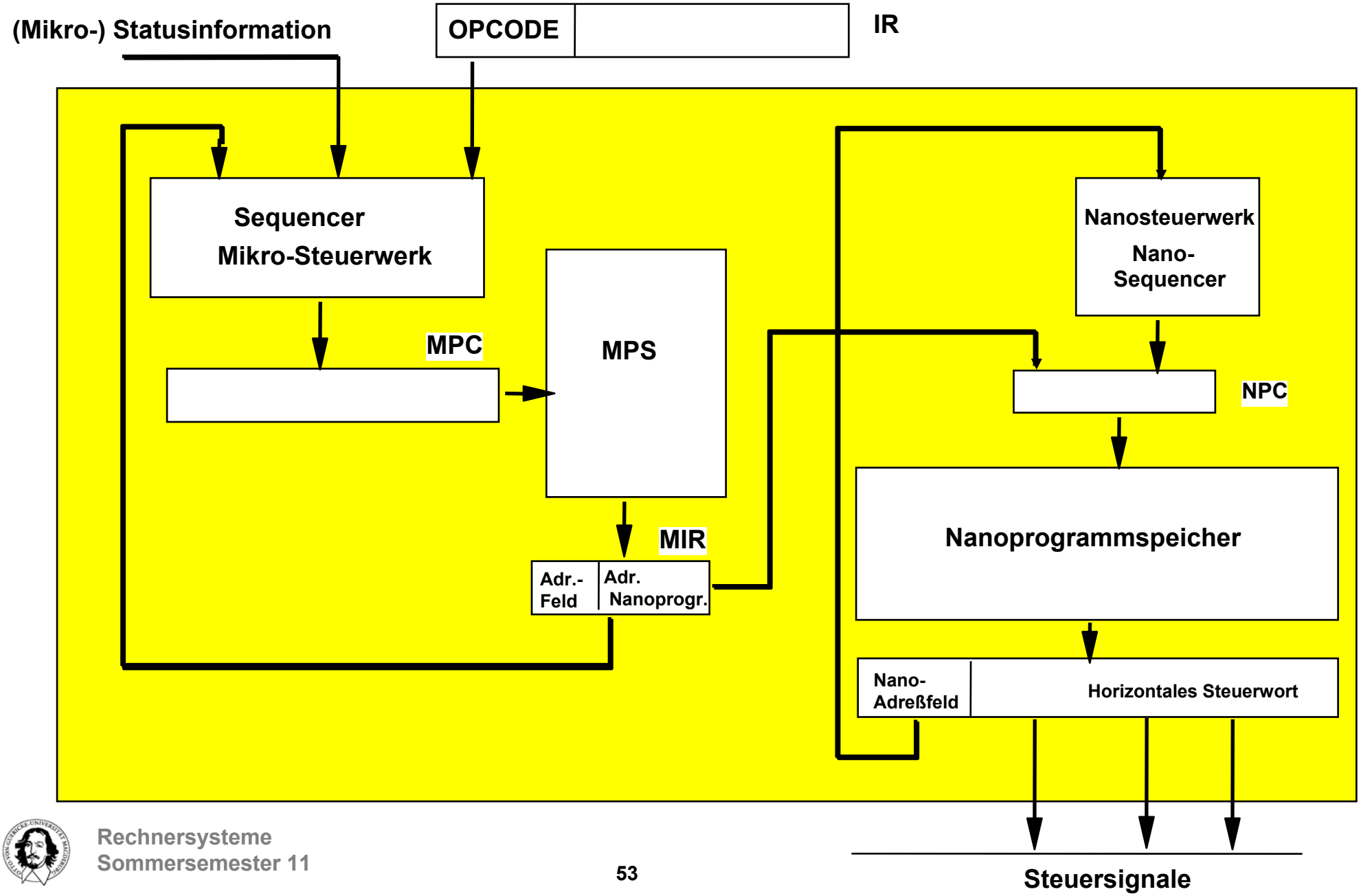
Nachteil : Zusätzliche Dekodierungshardware, Flexibilität, Geschwindigkeit

Vorteil : Kurzes Mikroinstruktionswort, kleiner Mikroprogrammspeicher
Übersichtlicher Mikroprogramme

Nanoprogrammierte Kontrolleinheit Stufe 1 (Quasi-Nanoprogr.)



Nanoprogrammierte Kontrolleinheit Stufe 2 (Echte Nanoprogr.)



Nanoprogrammierung:

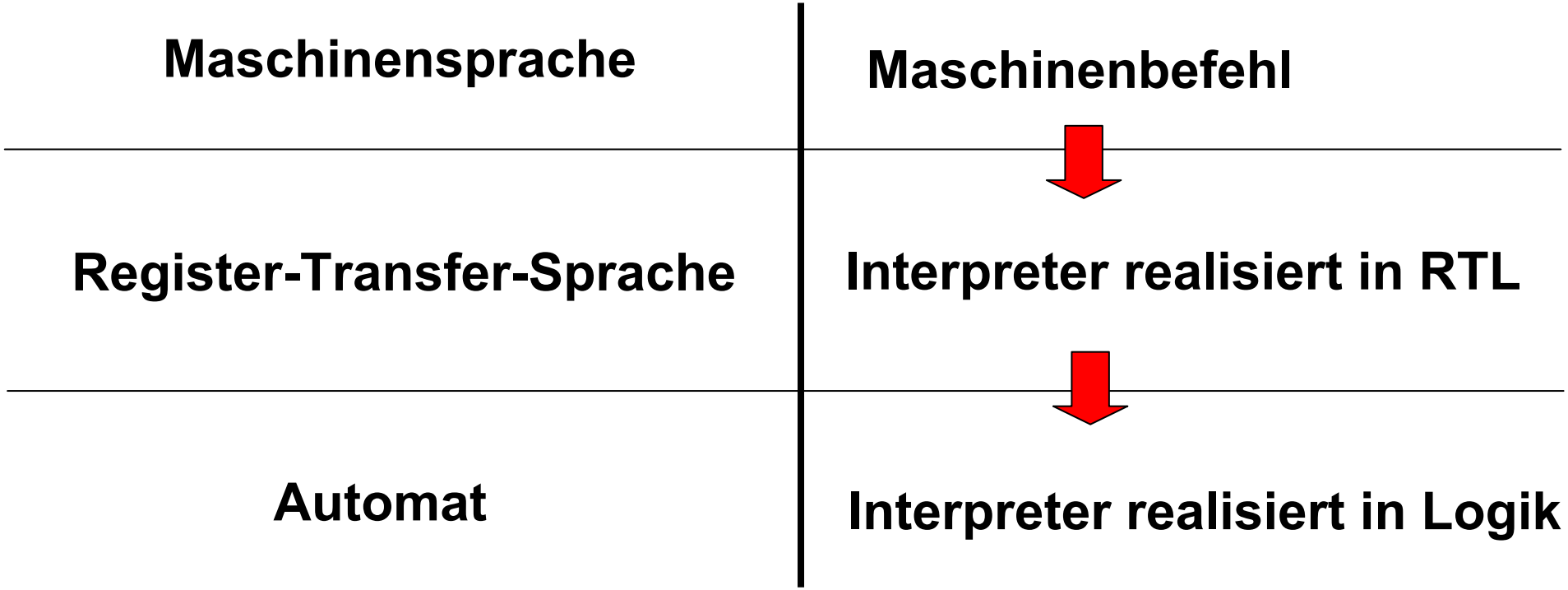
Kombination von vertikaler und horizontaler Mikroprogrammierung

Ersetzen der Dekodierungshardware durch eine weitere Stufe der Mikroprogr.

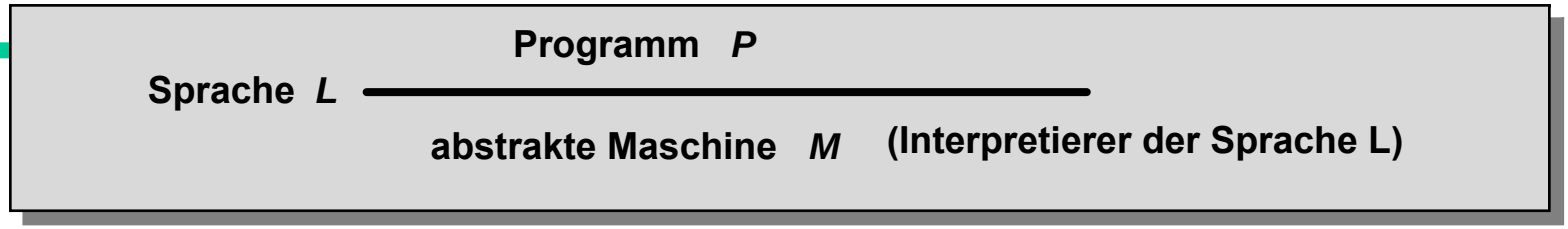
Vorteile:

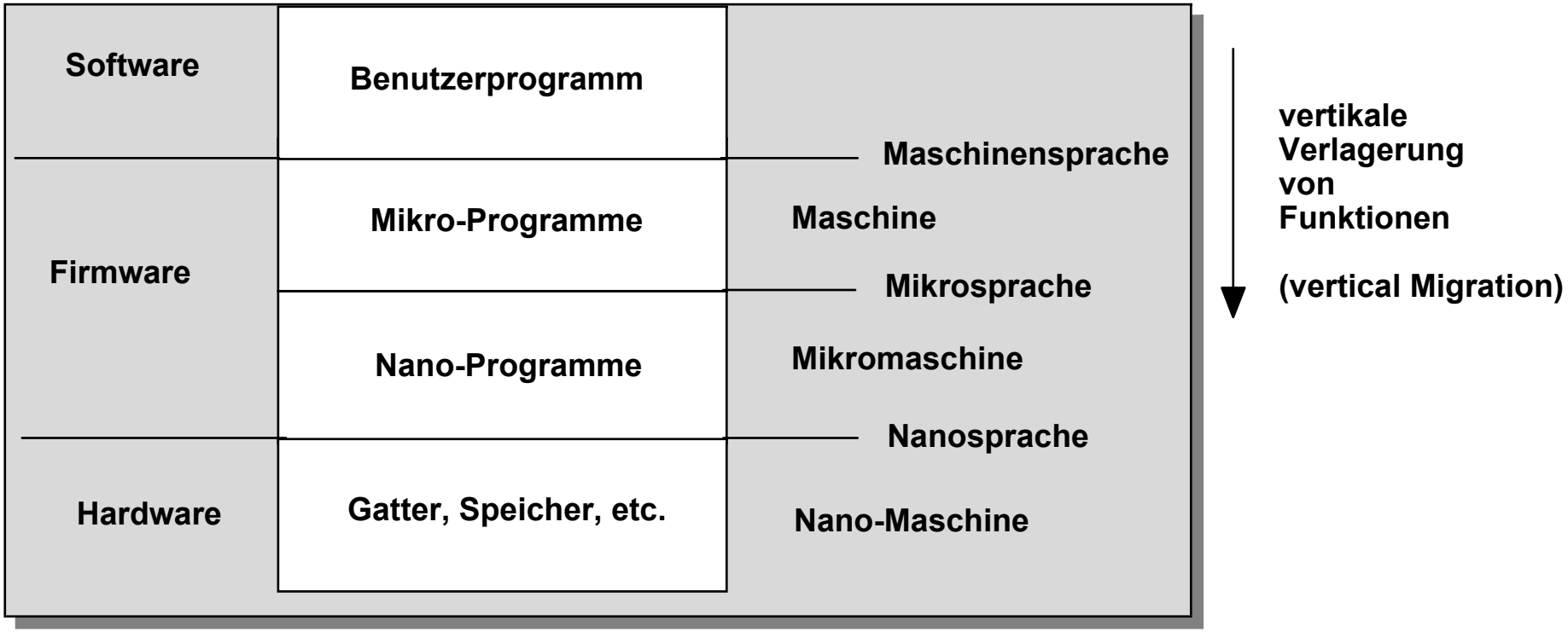
- **Flexibilität wie bei horizontaler MP**
- **Einsparung von Speicherplatz : gleiche Steuerteile können von mehreren Instruktionen genutzt werden**
- **(Festwert-) Speicher ist regulärer als ein Dekoder und leichter zu ändern**
- **Entkopplung von Mikroprogramm-Entwurf und Hardware-Entwurf**

Eine Hierarchie von Interpretern



Hierarchie von Maschinen und Vertikale Verlagerung





Vertikale Verlagerung ist die Realisierung von Funktionen einer höheren Ebene in der Architektur der unterliegenden Maschine, so daß diese Funktionen als Maschinenbefehle von der unterliegenden Maschine ausgeführt werden.

Motivationen, Vorteile und Trugschlüsse in der Mikroprogrammierung

\+ **Entkopplung von Hardware-Design und Design der Maschinensprache (konzeptuelle Maschinen, Rechnerarchitektur)**

\+ **Definition von "Rechnerfamilien" auf unterschiedlicher Hardwarebasis**
Beisp.: IBM 360 und 370 Familien

? **Mikrooperationen sind in schnellem Speicher abgelegt**
(dieser Vorteil ist seit dem Einsatz von Caches deutlich gesunken)

? **Parallelismus der Hardware kann genutzt werden**
(dieser Vorteil ist seit dem Einsatz von Pipelines und superscaleren Architekturen auf der Prozessorebene relativiert)

Trugschluß: Ein Mikroprogramm, das eine komplexe Instruktion realisiert ist, ist nie langsamer als eine Folge einfacher Maschinenoperationen.
Beisp.: Index Instruktion bei VAX, Multiply
Grund: eine mikroprog. Maschinenoperation muß alle Komb. von Operanden berücksichtigen

Versuchung: viele komplexe Operationen(Algorithmen) in der Maschinensprache
Compilerschwierigkeiten, Aufwärts/Abwärtskompatibilität
lange Designzeiten mit Problemen des Softwareengineerings,
Beispiel: MC 68020

Gründe, welche die Relevanz der Mikroprogrammierung in Frage stellen:

Technische Gründe:

- **Die Kontrolleinheit ist in einem Microprocessor auf demselben Chip realisiert. Mikroprogramme können nicht einfach verändert werden.**
- **ROM ist nicht mehr schneller als RAM. Dadurch werden Mikroprogramme im ROM nicht mehr schneller als Maschinenprogramme im RAM.**
- **Programmierbare Logik mit ihrer regulären Struktur ist oft weit effizienter zur Realisierung von Kontrollstrukturen als ein Mikroprogramm.**
- **Programmierbare Logik kann ebenso leicht geändert werden wie ein Mikroprogramm.**
- **Befehlssätze werden einfacher (RISC- Ansatz) . Damit wird auch die Kontrolleinheit einfacher.**

Gründe, die sich aus der Änderung der Anwendung und der Entwicklungsumgebung ergeben:

- **Anwendungs- und Betriebssystem-Software ist auf einen bestimmten Befehlssatz zugeschnitten. Ein neuer Befehlssatz bedeutet, daß diese Software angepaßt werden muß.**
- **Instruktionssätze werden immer ähnlicher**
- **Der Entwurf eines Gate Arrays zur Realisierung einer Kontrolleinheit ist, bedingt durch geeignete Entwurfswerkzeuge, nicht schwieriger oder fehleranfälliger als das Schreiben eines Mikroprogramms.**
- **Programmierbare Logik kann ebenso leicht geändert werden wie ein Mikroprogramm.**

Weitere Probleme der Mikroprogrammierung

- Chip-Flächenbedarf
  - gängige Mikroprozessoren (50 bis 500 Steuerleitungen)
  - großer Mikroprogrammspeicher erforderlich
    - oft mit vielen Nullen gefüllt
    - Reduktion mit vertikaler Mikroprogrammierung und Nanoprogrammierung möglich
- Langsame Ausführung
  - Maschinenbefehl benötigt mehrere Takte des Mikroprogramms
  - Taktgeschwindigkeit begrenzt durch
    - Speicherzugriff Adresstabelle, Speicherzugriff Mikroprogrammspeicher
    - Dekodierung der vertikalen Mikroprogrammierung

Lernziele

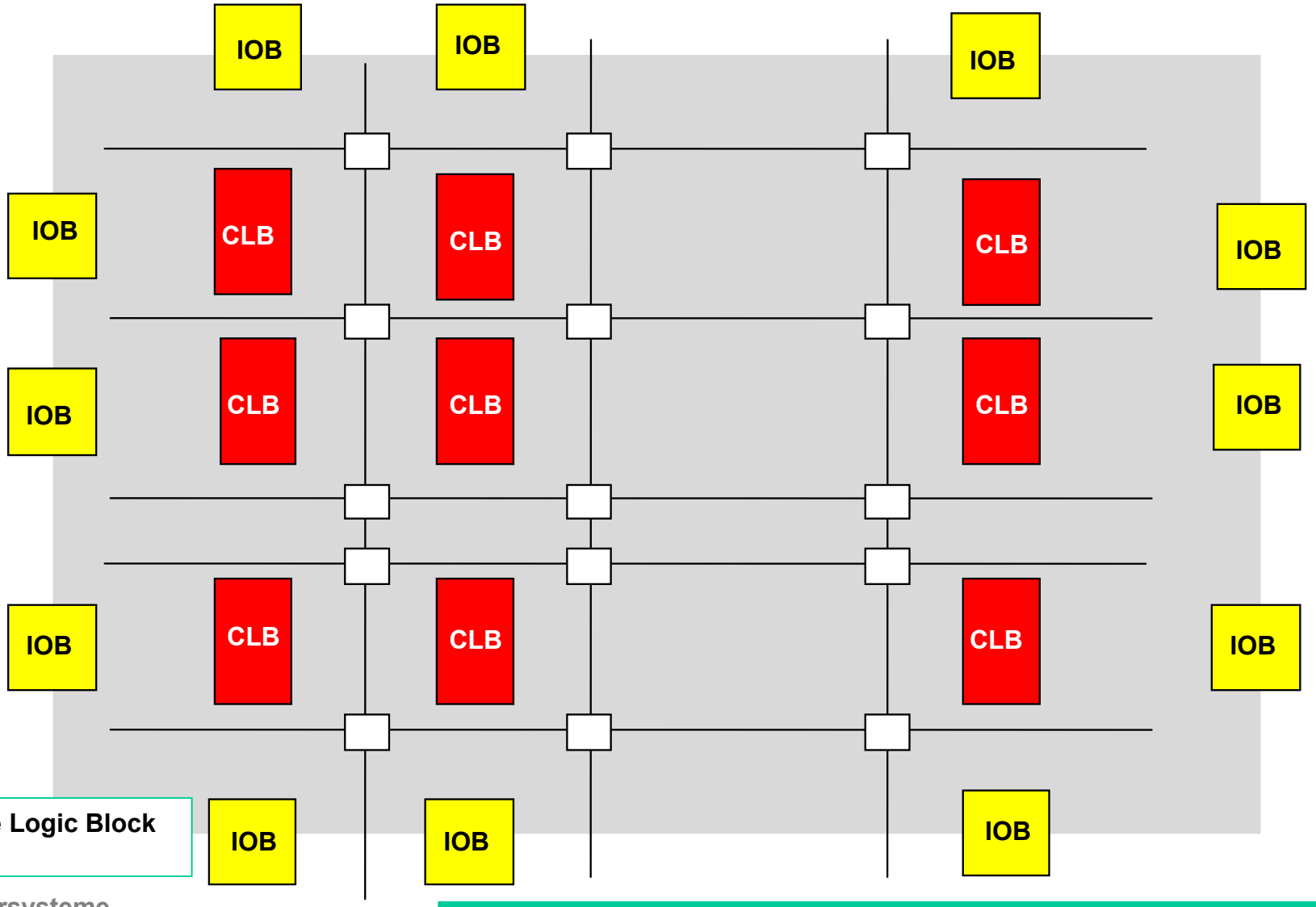
- Verständnis der Abläufe in der CPU
- Beschreibung durch eine Register Transfer Sprache
- Entwurf von Befehlen und deren Ablaufsteuerung
- Verständnis der Hierarchie von Interpretern
- Mikroprogrammierung als Implementierungskonzept
- Äquivalenz von Logik, Speicher und Programm zur Realisierung von Ablaufsteuerungen
- Zielkonflikte bei der Mikroprogrammierung

komplexer, komplexer, komplexer, komplexer, komplexer, komplex

vom PAL zum Gate Array

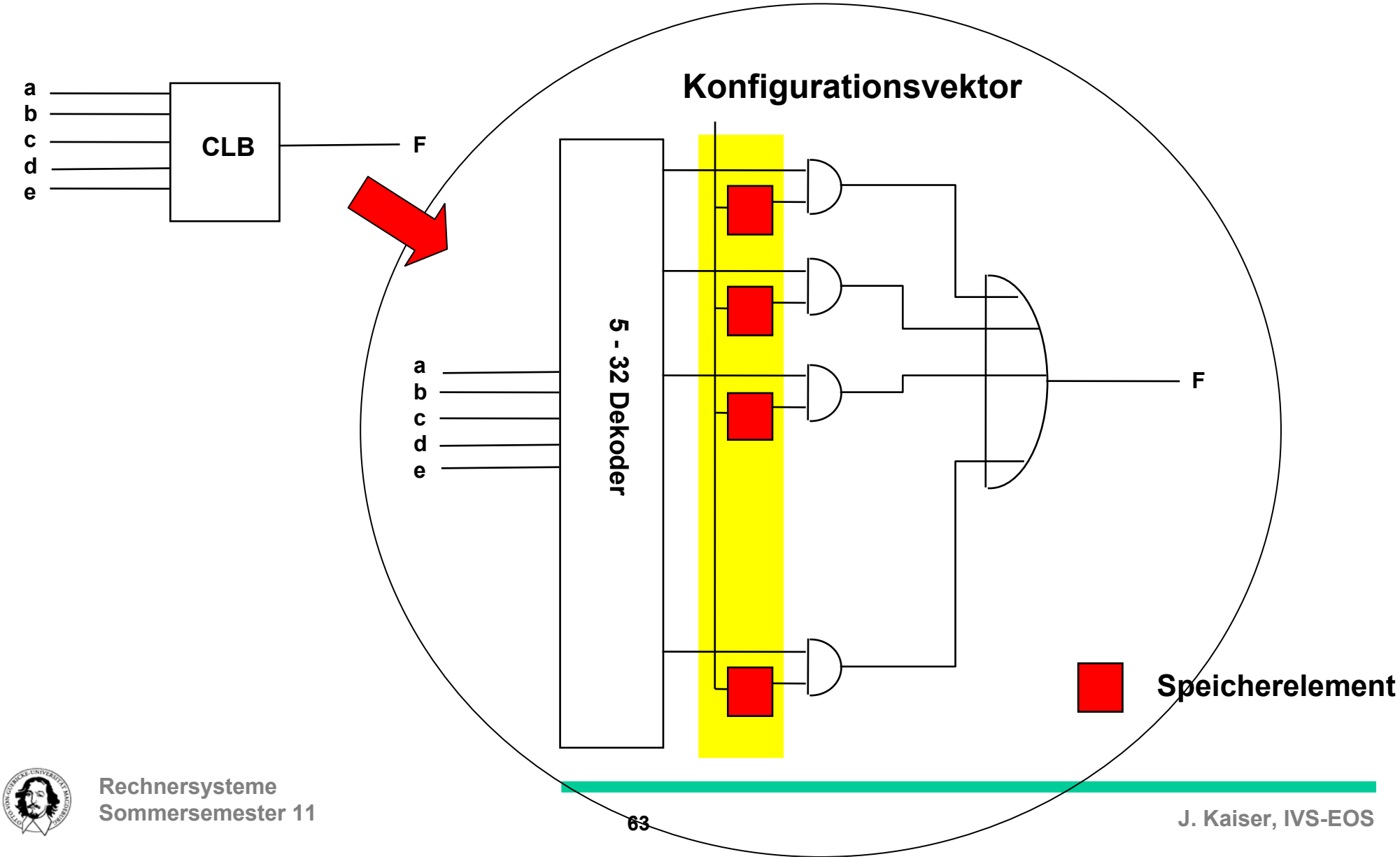
- Höhere Flexibilität
- Mehr Struktur
- Trennung von internen Funktionen und Ausgabe
- Komplexes Verbindungsnetzwerk
- Reprogrammierbarkeit

Die Struktur eines Gate Arrays

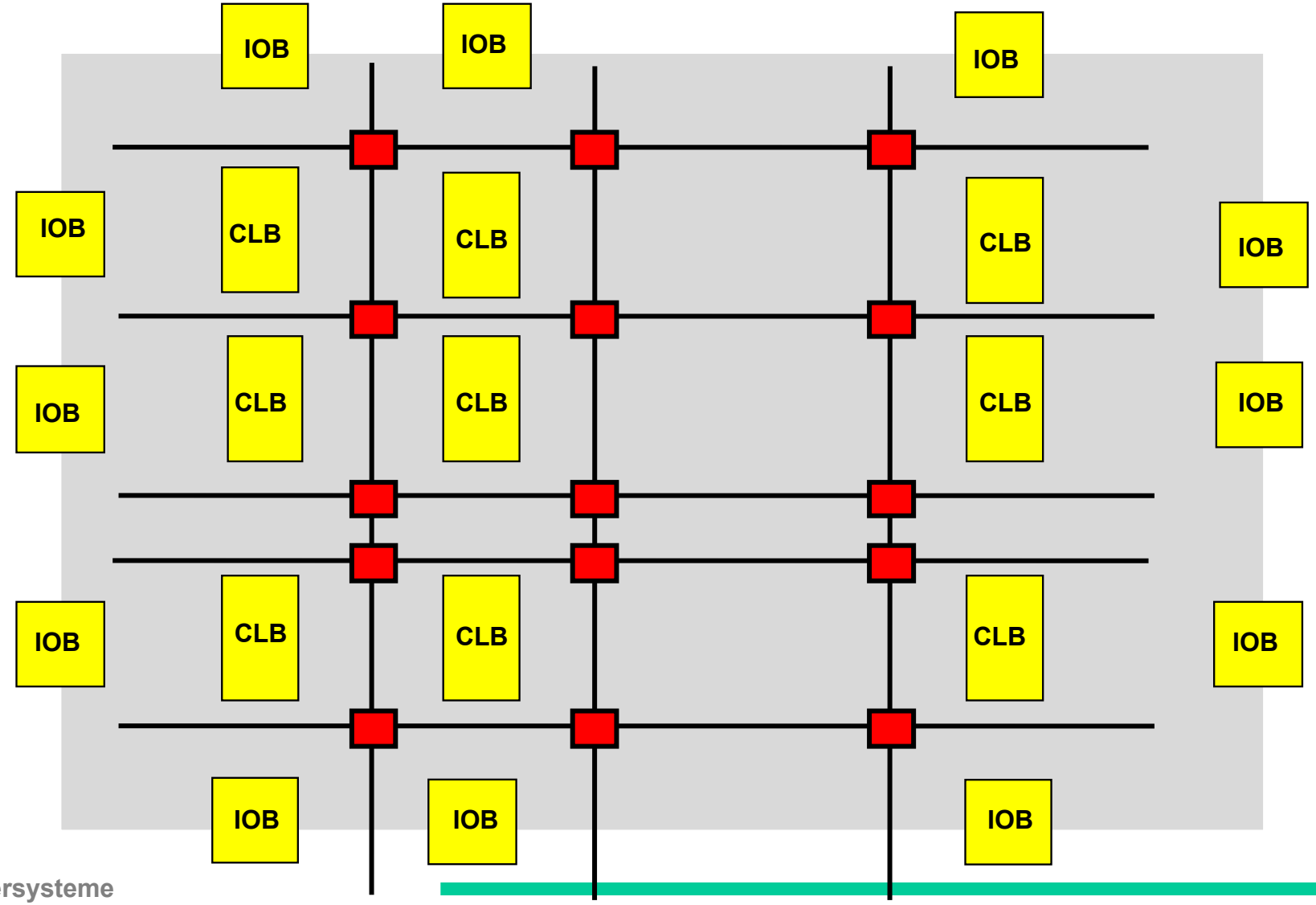


CLB: Configurable Logic Block
IOB: I/O Block

Der konfigurierbare logische Block (CLB)



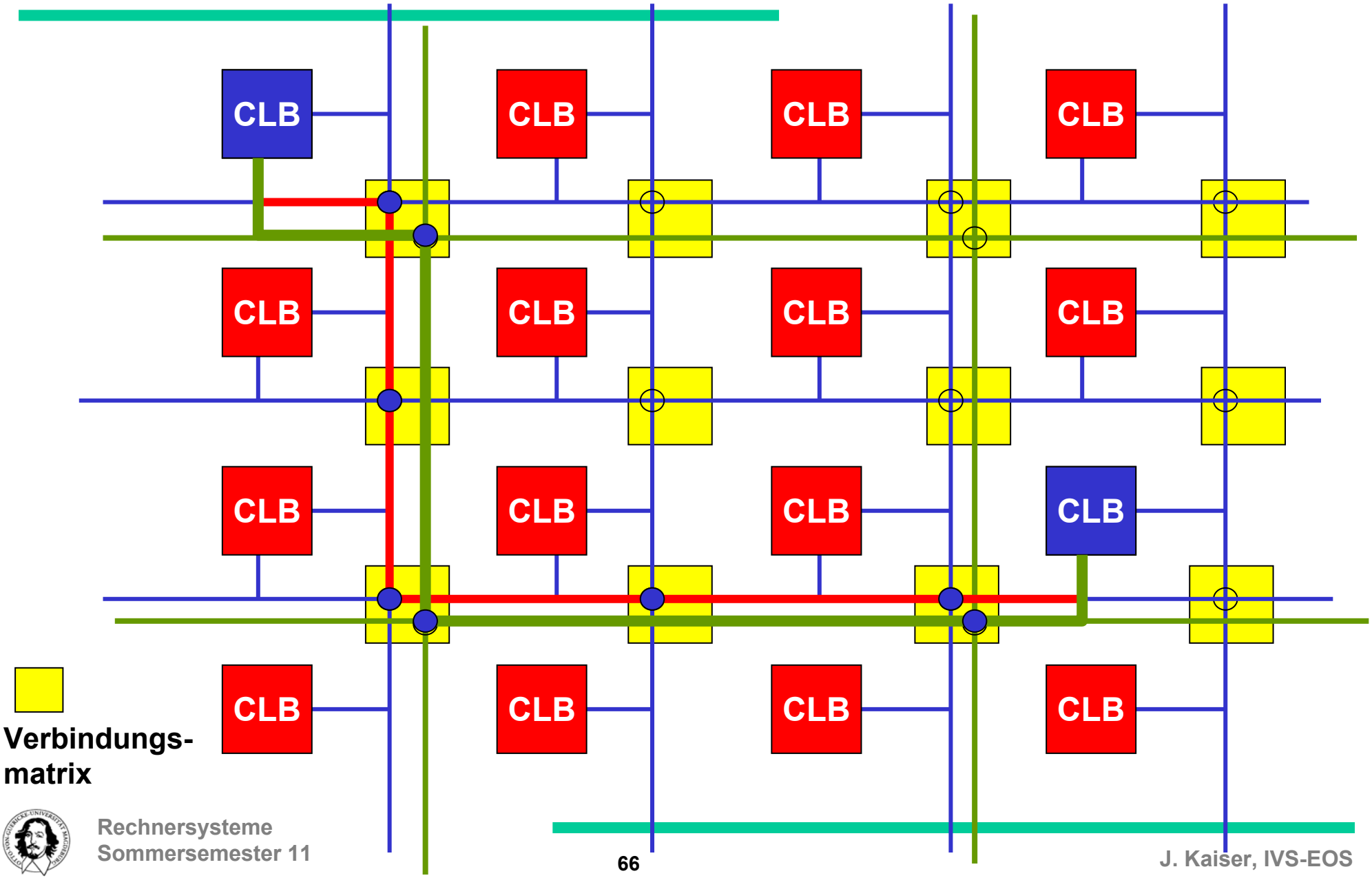
Die Verbindungsstruktur



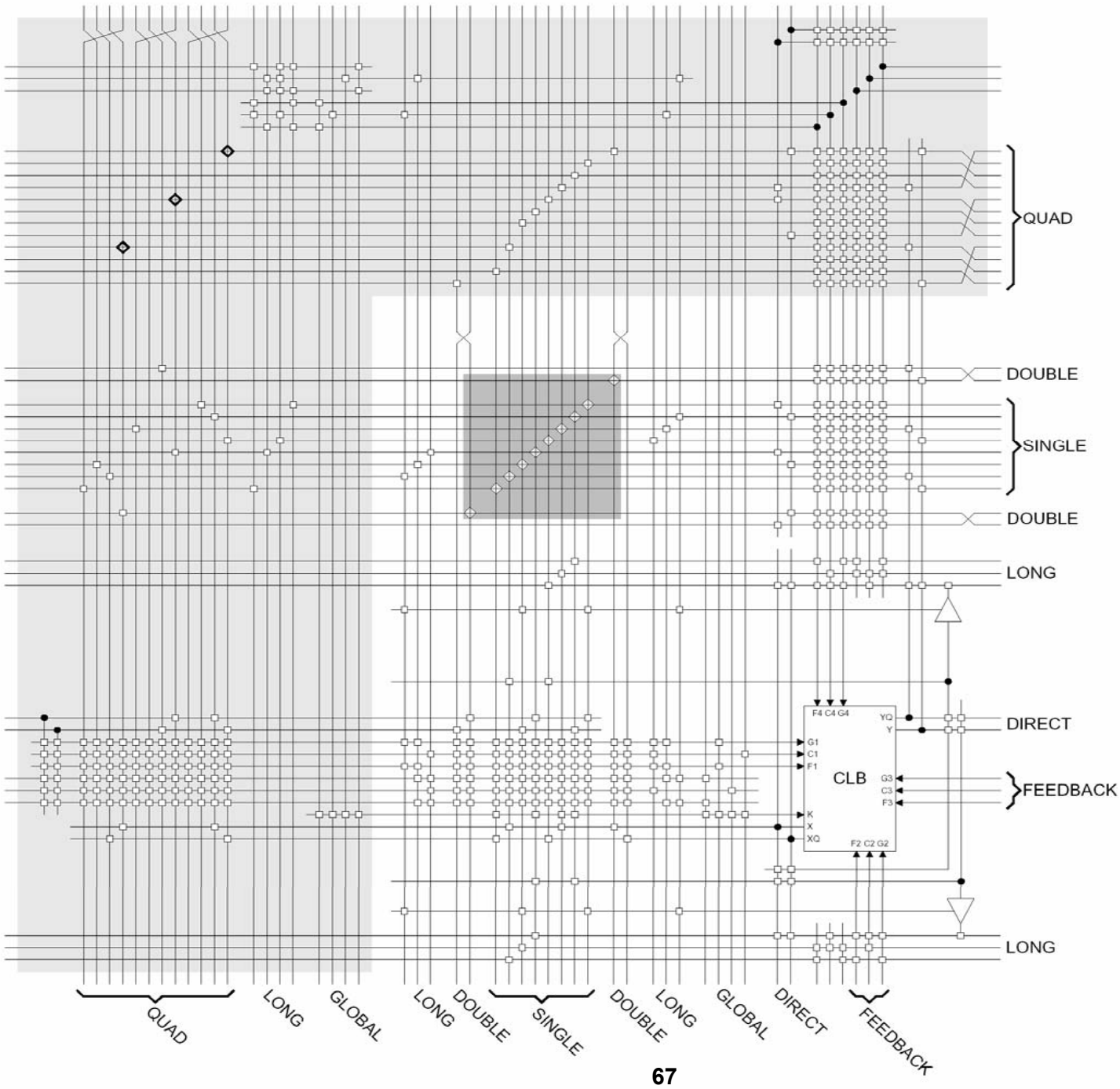
Verbindungstypen:

- direkte Verbindungen zwischen benachbarten CLBs
- lokale Verbindungen („Spannweite: 1,2,4)
- globale Verbindungen

Die Verbindungsstruktur



Verbindungs-struktur für einen CLB



Gate Array Eigenschaften:

Device	XC4002XL	XC4005	XC4010	XC4013	XC4020	XC4028	XC4036	XC4044	XC4052	XC4062	XC4085
Max Logic Gates	2,000	5,000	10,000	13,000	20,000	28,000	36,000	44,000	52,000	62,000	85,000
CLBs (Row x Column)	64 (8 x 8)	196 (14 x 14)	400 (20 x 20)	576 (24 x 24)	784 (28 x 28)	1,024 (32 x 32)	1,296 (36 x 36)	1,600 (40 x 40)	1,936 (44 x 44)	2,304 (48 x 48)	3,136 (56 x 56)
IOBs	64	112	160	192	224	256	288	320	352	384	448
Flip-Flops	256	616	1,120	1,536	2,016	2,560	3,168	3,840	4,576	5,376	7,168
Bits per Frame	133	205	277	325	373	421	469	517	565	613	709
Frames	459	741	1,023	1,211	1,399	1,587	1,775	1,963	2,151	2,339	2,715
Program Data	61,052	151,910	283,376	393,580	521,832	668,124	832,480	1,014,876	1,215,320	1,433,804	1,924,940
PROM Size (bits)	61,104	151,960	283,424	393,632	521,880	668,172	832,528	1,014,924	1,215,368	1,433,852	1,924,992

Konfiguration von Gate Arrays

Konfigurationsschnittstelle: seriell/byte-parallel

Wie konfigurieren?

1. **Internen Speicher löschen,**
2. **Konfigurationsmodus initialisieren,**
3. **Konfigurationsdaten laden,**
4. **In Arbeitszustand versetzen.**

Eigenschaften von Gate Arrays

Programmierung erfolgt in höherer Programmiersprache

Logik und Verbindungsstruktur lassen sich einfach ändern

Parallelität ist inhärent auf dem Chip möglich