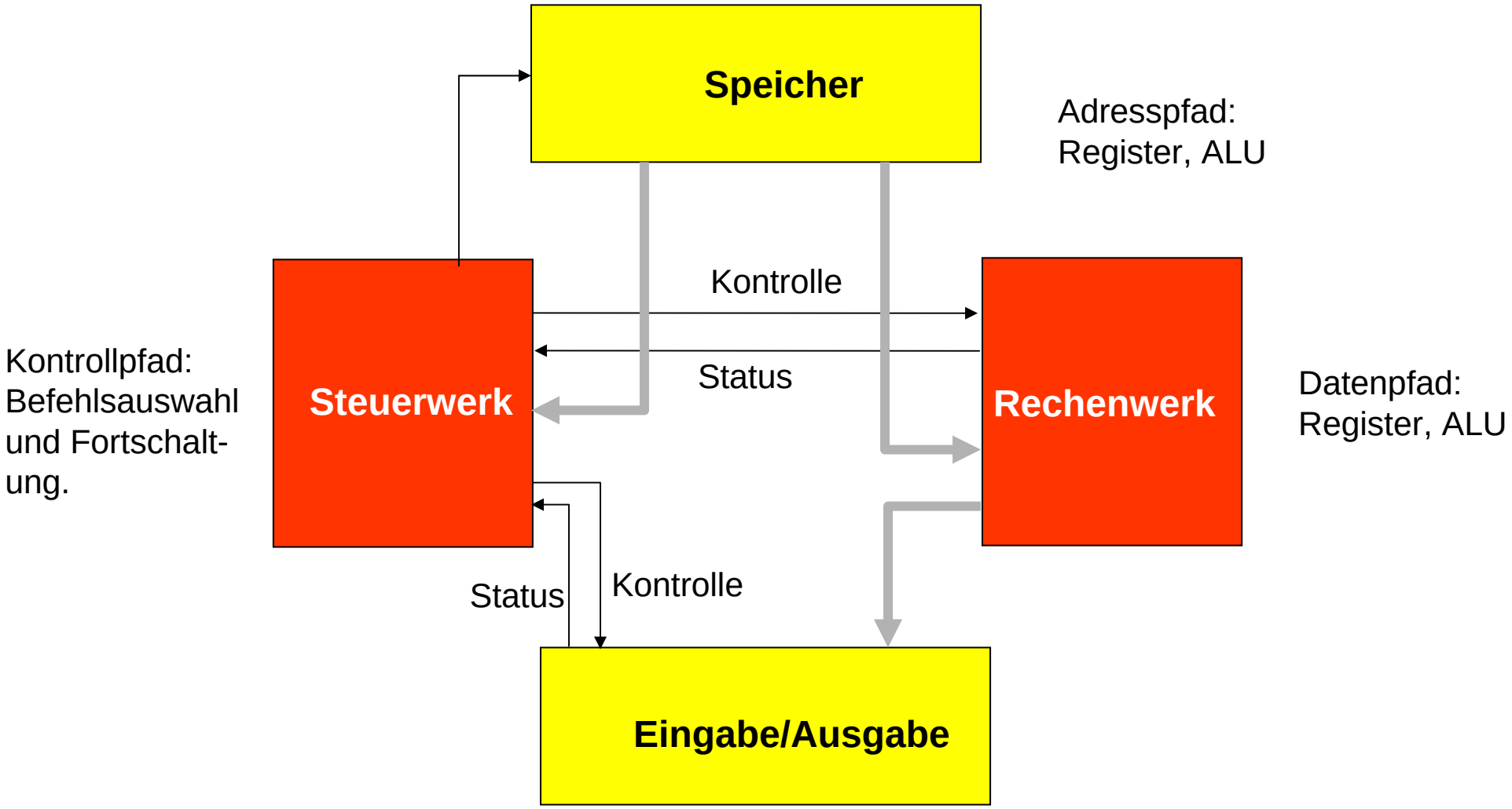
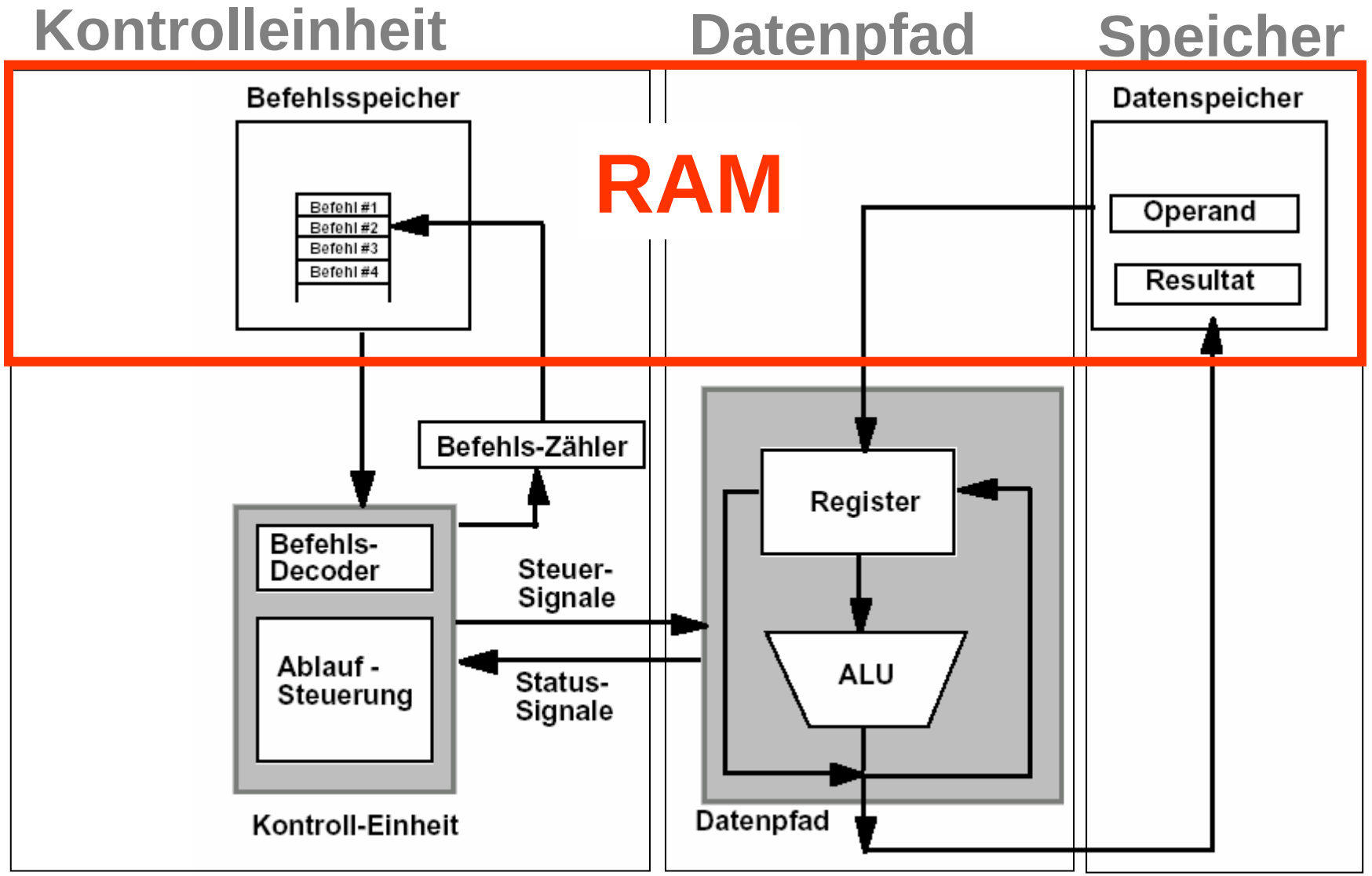
Realisierungsalternativen für eine Zentraleinheit

Otto-von-Guericke-Universität Magdeburg

Komponenten des klassischen Rechners



Prinzip des sequentiellen programmgesteuerten Prozessors



Der Entwurf einer CPU

Where shall I begin?

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

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

Wortformate

Wie groß sind meine Operanden?

Welcher Adreßraum soll unterstützt werden?

Wie viele Befehle müssen codiert werden?

sehr einfacher Rechner:

Wortorientiert,

16-Bit Worte

16 Befehle

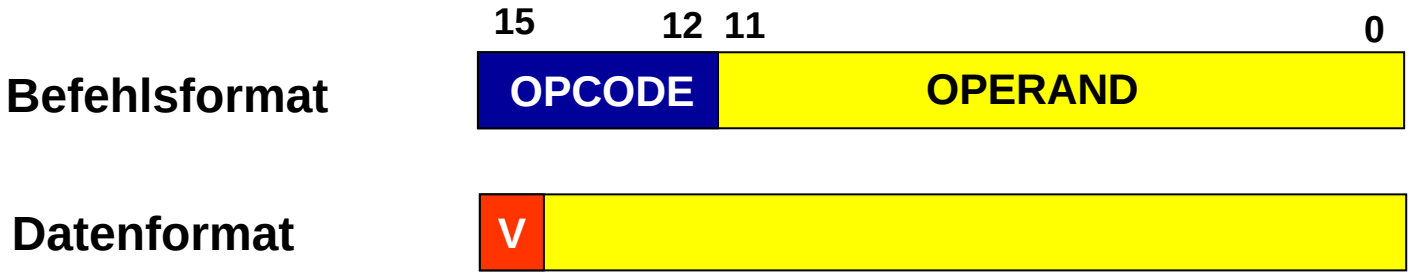
Ein (sehr) einfacher Befehlssatz

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

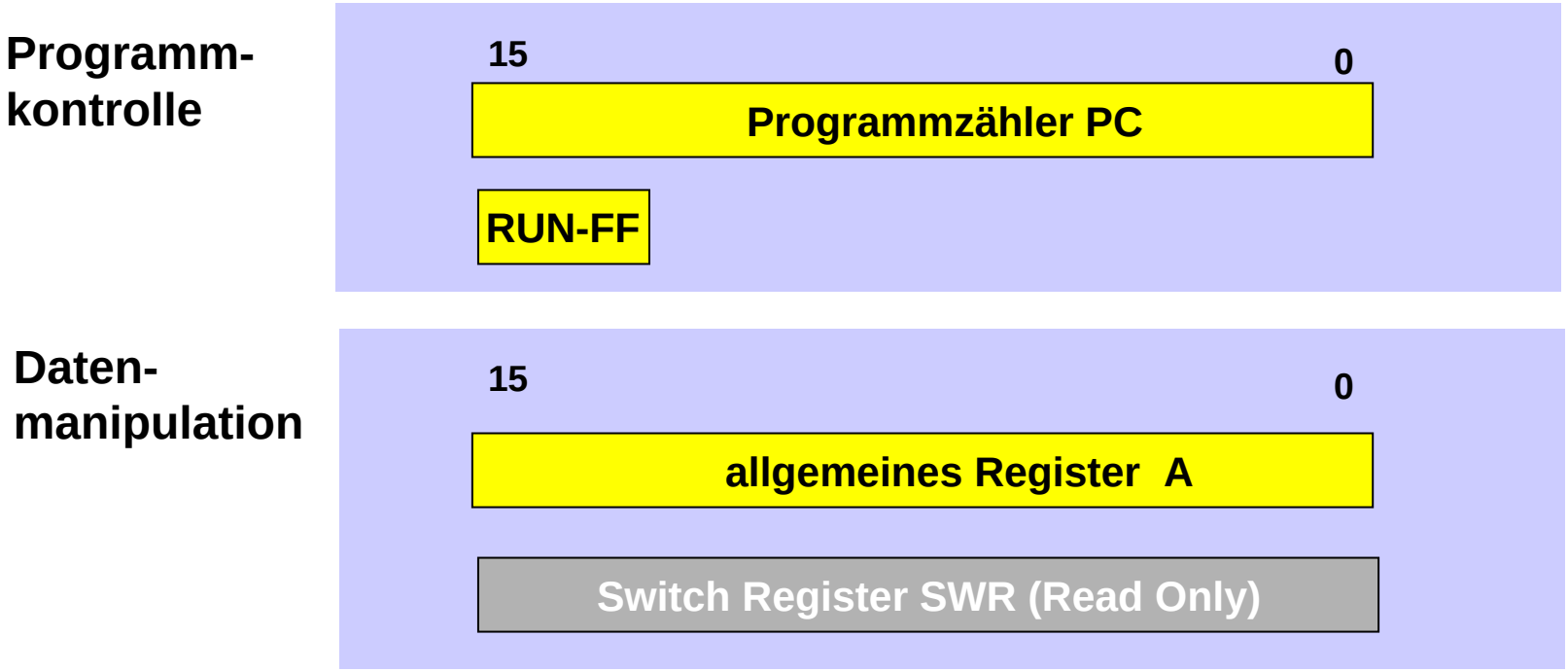
0000	HLT		Halt
0001	JMA		Jump on Minus
0010	JMP		Jump to Address
0011	JSR		Jump Subroutine
0100	CSA		Copy Switches to A-Register (Accumulator)
0101	RAL		Rotate A Left
0110	INP		Input Instruktion
0111	OUT		Output Instruktion
1000	NOT		Komplementbildung
1001	LDA	Adresse	Load A from Memory Address
1010	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	CSA		(Copy Switch Register to A) Der Zustand der Schalter wird in das Register A geladen.
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.

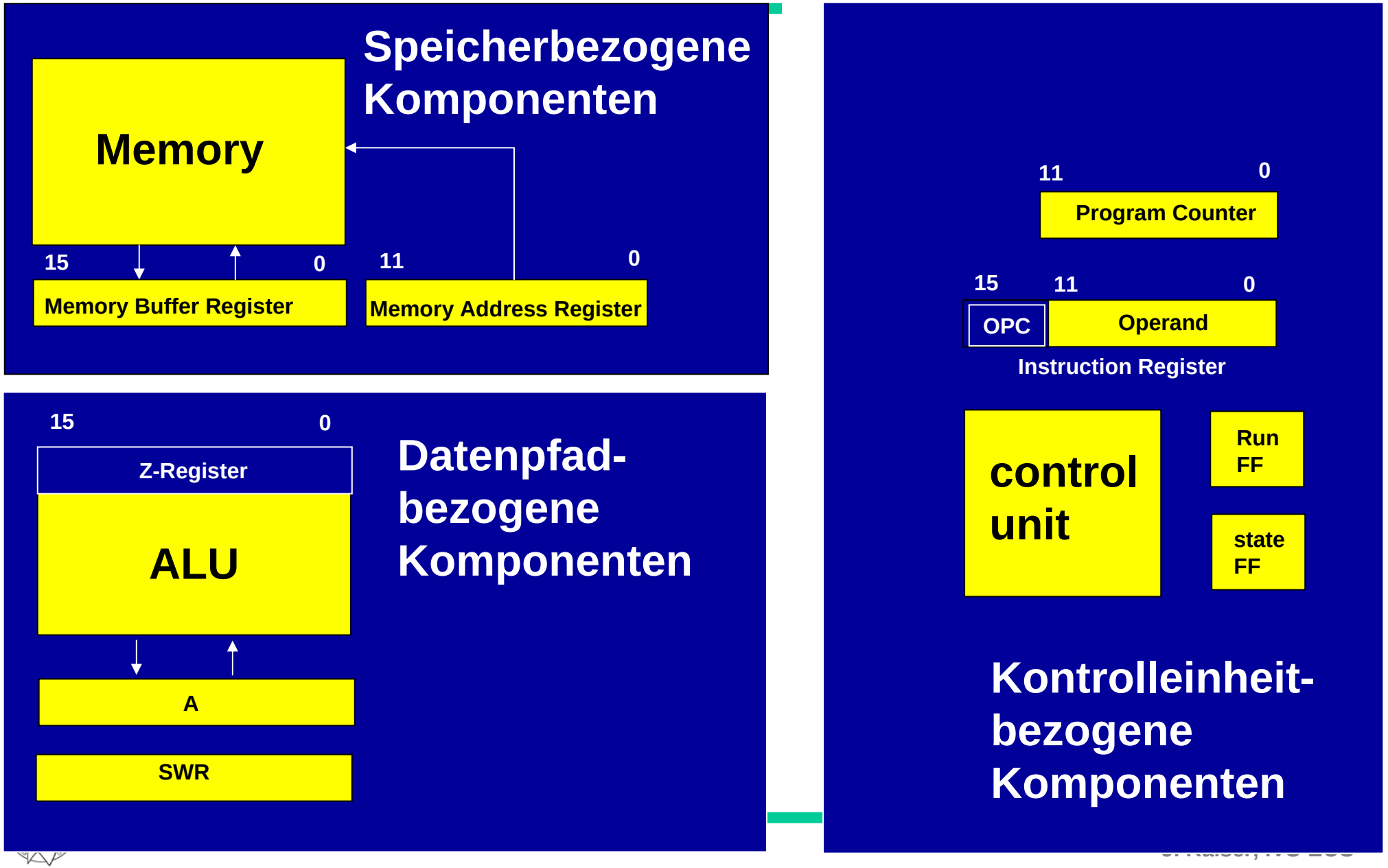
Wortformate



.. und Programmiermodell

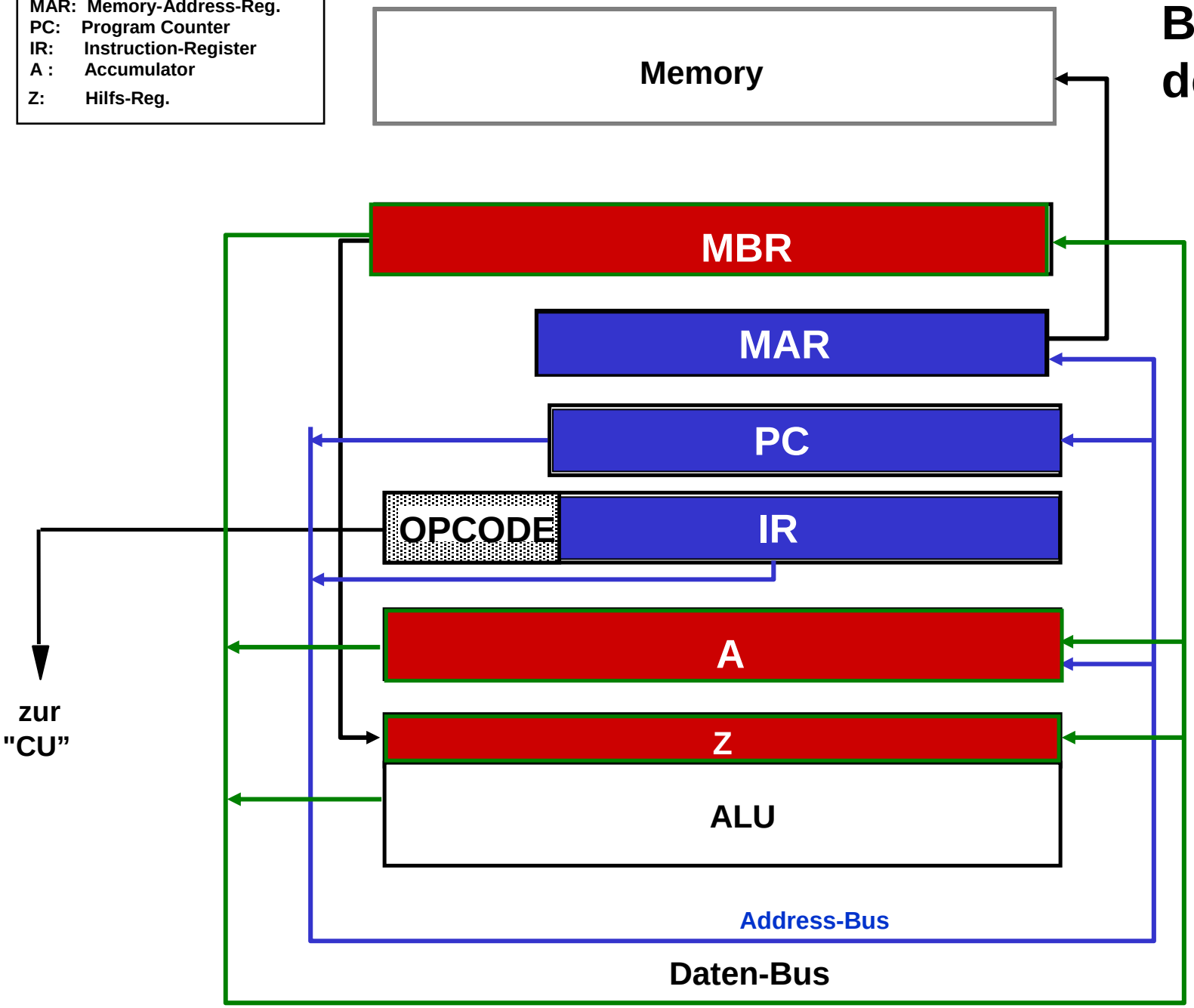


Vom Programmiermodell ..... zum funktionsfähigen Rechner



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

Bus-Struktur des Datenpfads

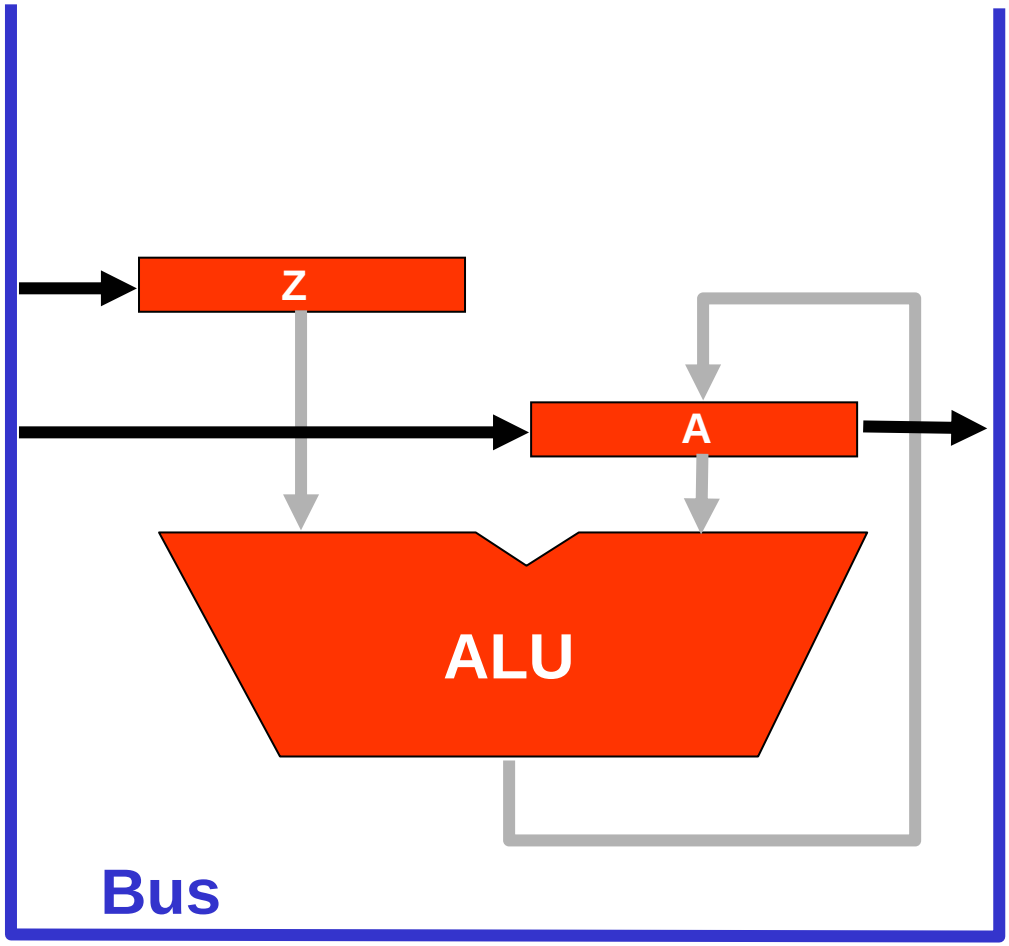


Datenpfad

Durchführen einer arithm. oder logischen Operation:

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

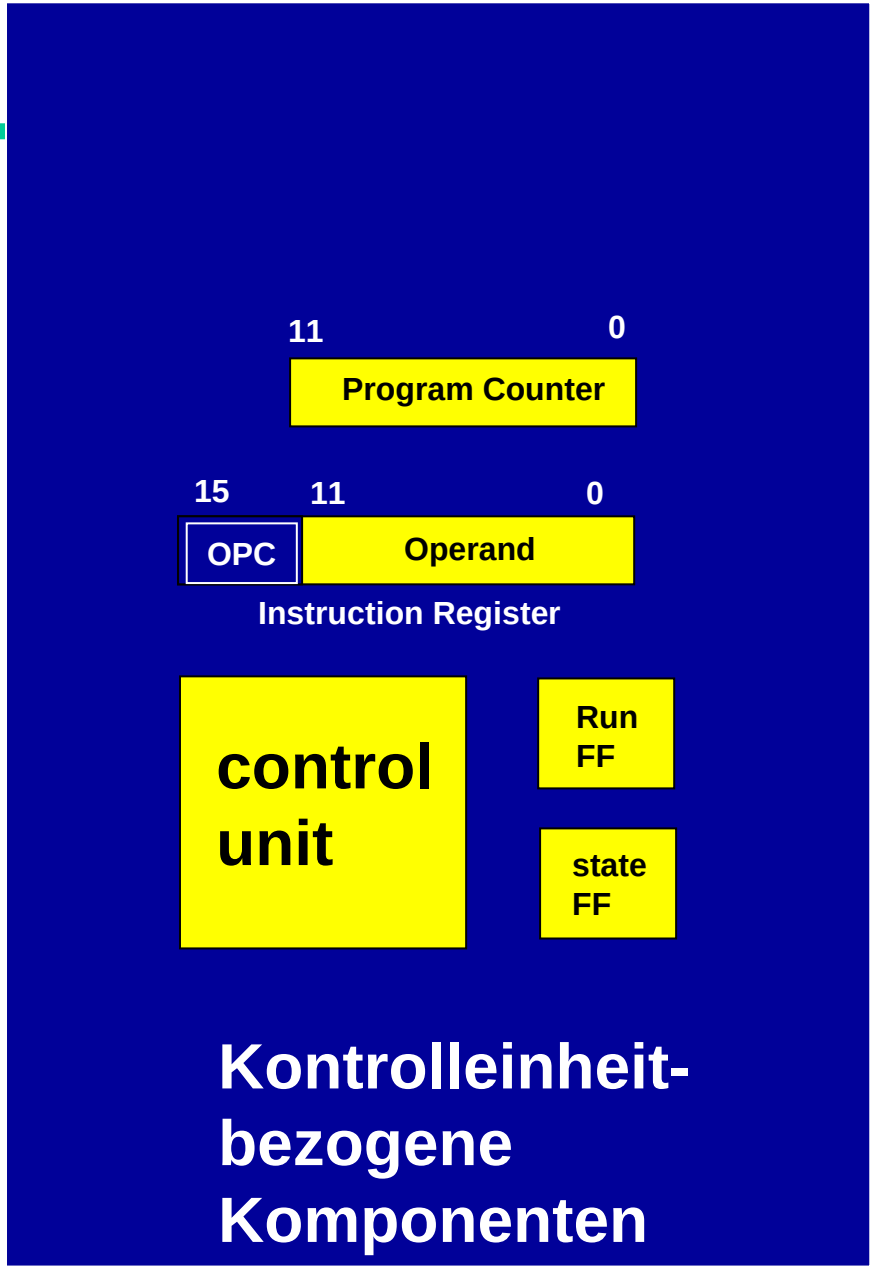
- **Transferiere 1. Operanden von A nach Z**
- **Lade 2. Operanden nach A**
- **Führe ALU-Operation aus**
- **Transferiere Ergebnis nach A**



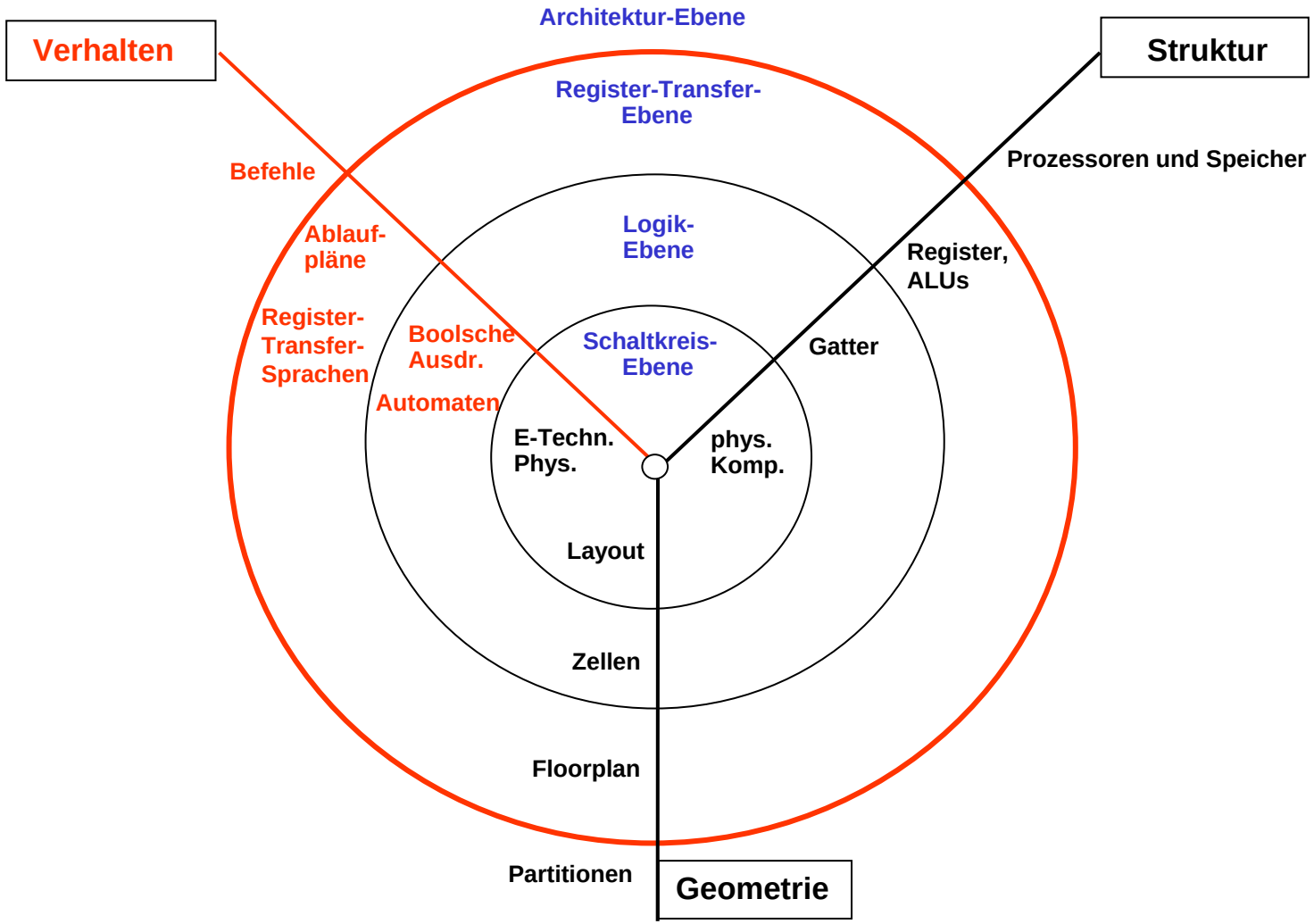
A..K: sichtbare Register
L,Z: Hilfsregister

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

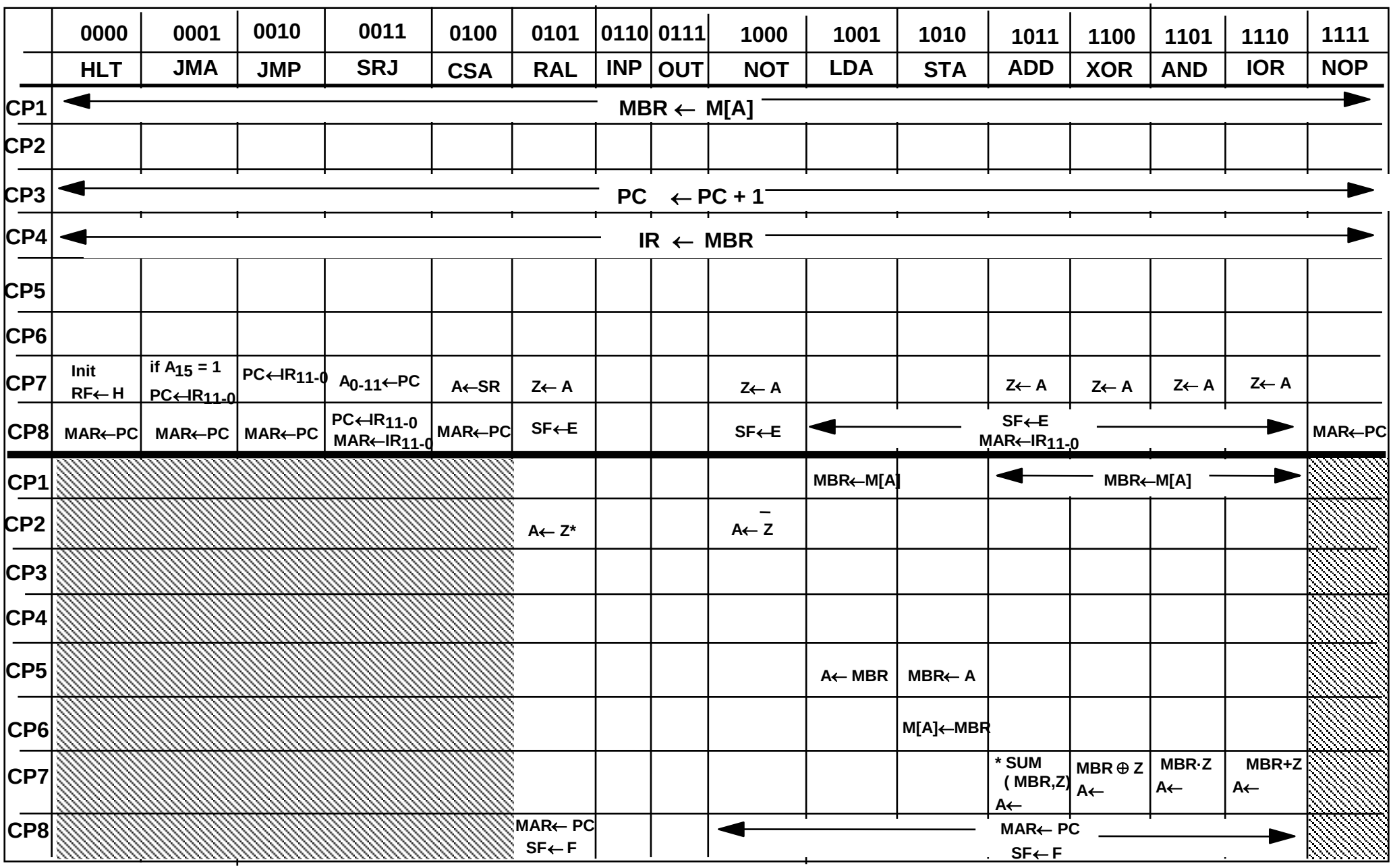
Register	**Rn-0**	bezeichnet die Bitstellen n,n-1,. . . , 0 des Registers R
Transfer:	←	A ← B bezeichnet den Transfer des Inhalts von Register B nach A
Speicher:	**M[addr]**	bezeichnet den Inhalt der Speicherzelle mit der Adresse "addr"
Bedingungen:	**B:**	z.B. R = 0: A ← B, CP7 • RAL: A ← Z

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.

Detaillierter Ablauf und Timing der Maschinenbefehle

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



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

Ausführungsphasen

Befehlsholphase (Speicherzugriff)

Befehlsdekodierung

Befehlsausführung

Ausführung benötigt keine Operanden

Beisp. Shift-Op.s
Sprünge, NOP.

Ausführungsphase

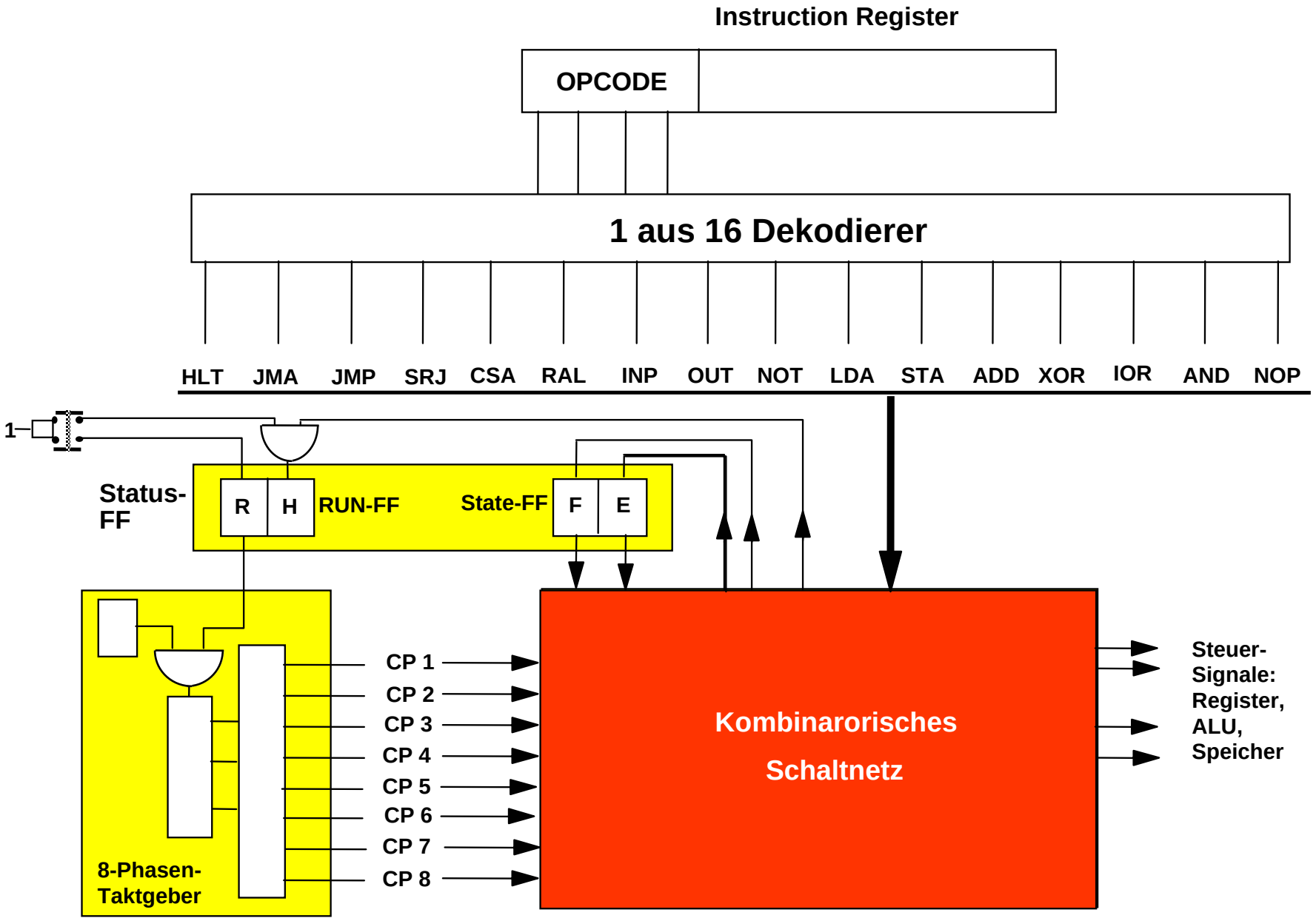
Ausführung benötigt zusätzliche Operanden

Beisp.: zweistellige arithmetisch/logische Operationen

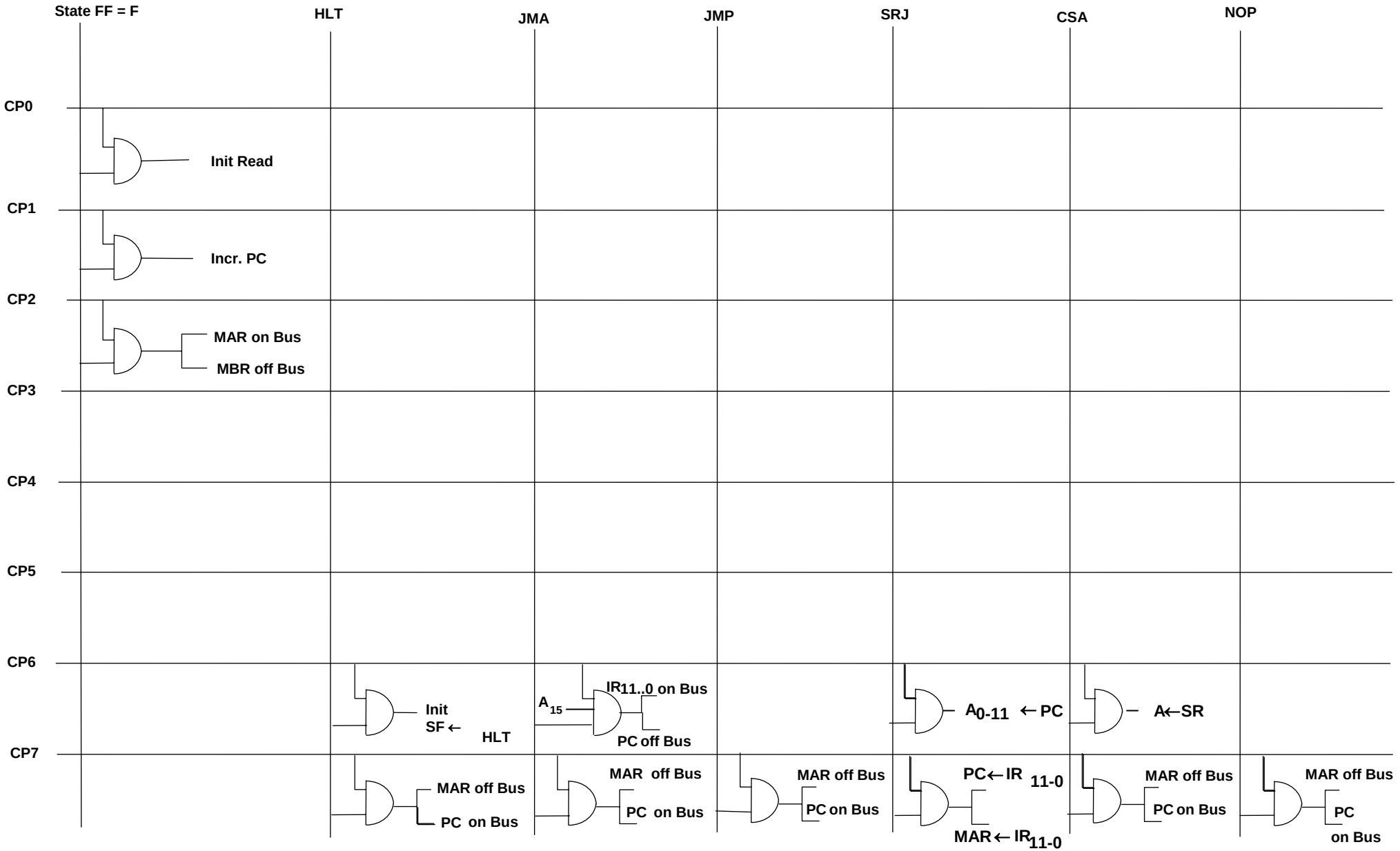
Operandenholphase
(Speicherzugriff)

Ausführungsphase

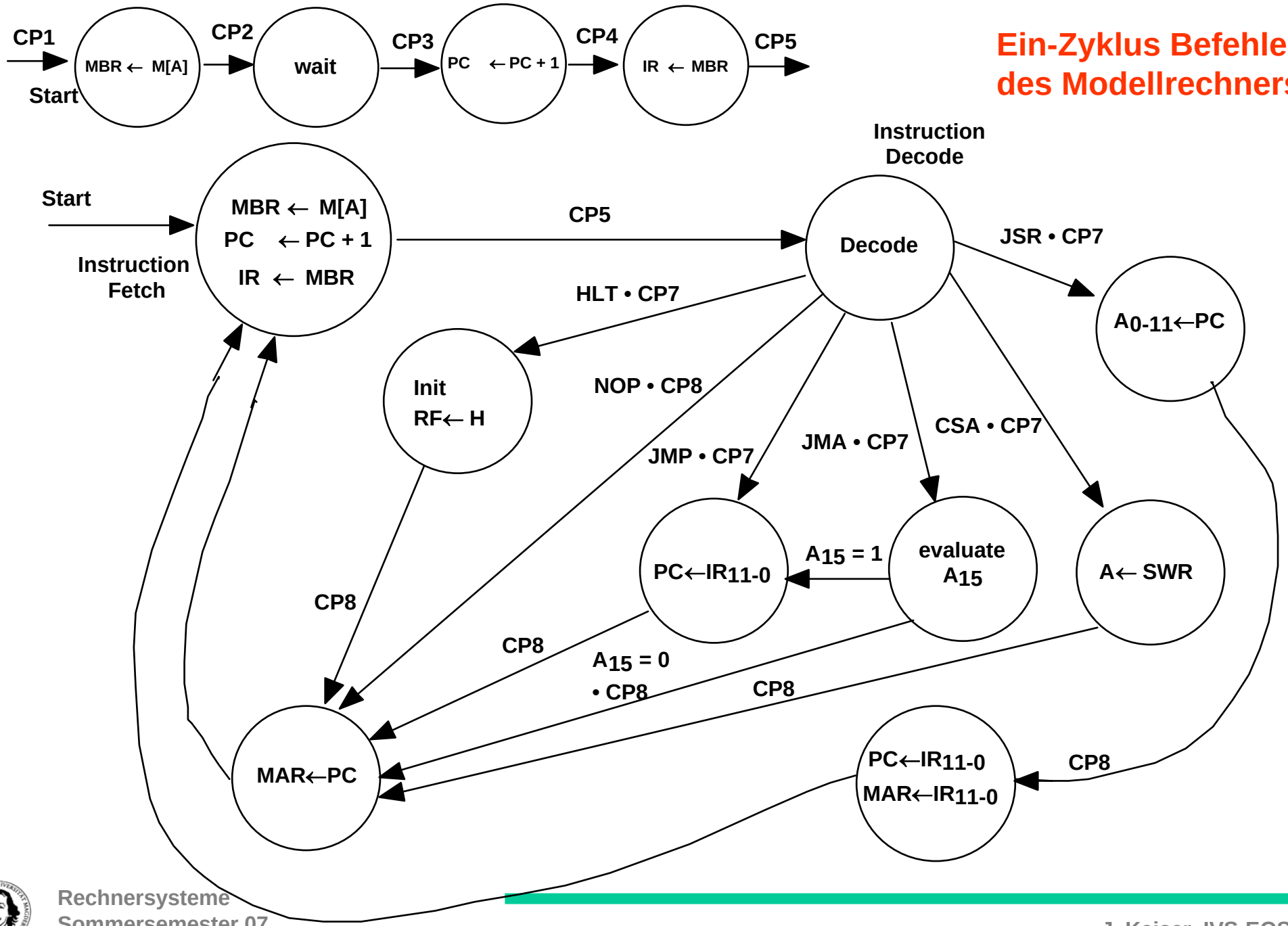
Die Kontrolleinheit des Modellrechners



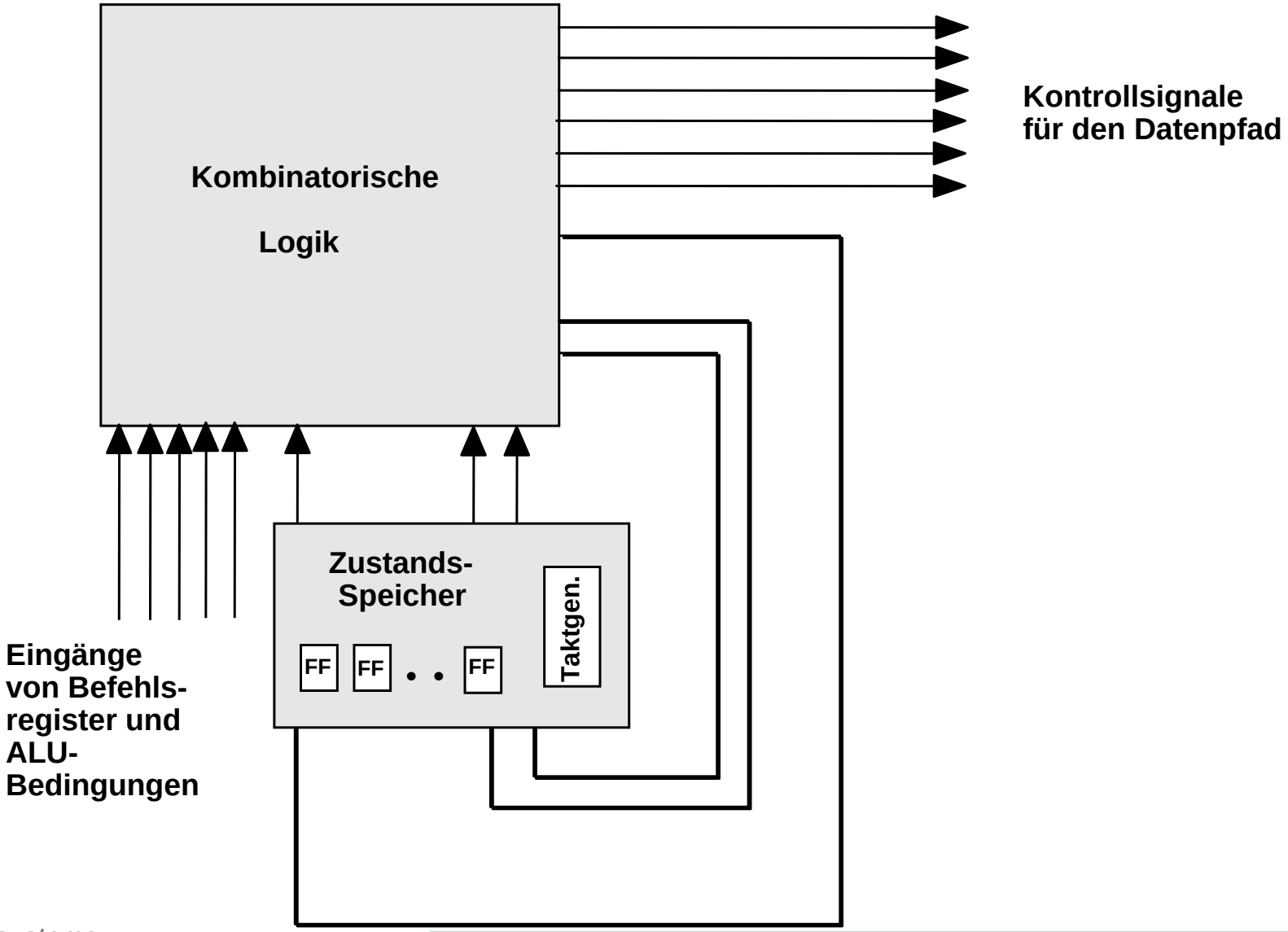
1-Zyklus Befehle



Ein-Zyklus Befehle des Modellrechners



Festverdrahtete Zustandsmaschine zur Realisierung der Kontrolleinheit

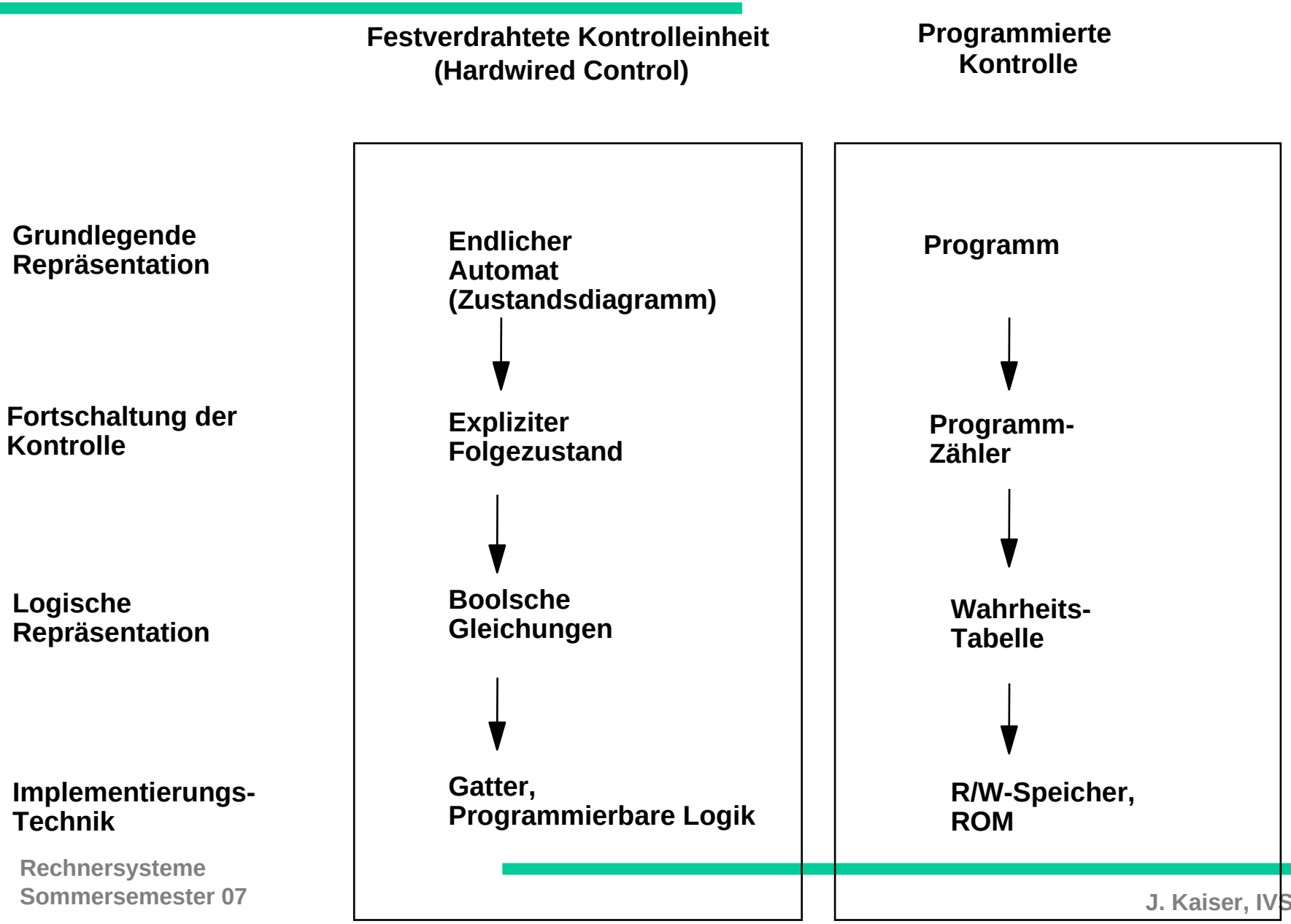


Idee der Mikroprogrammierung

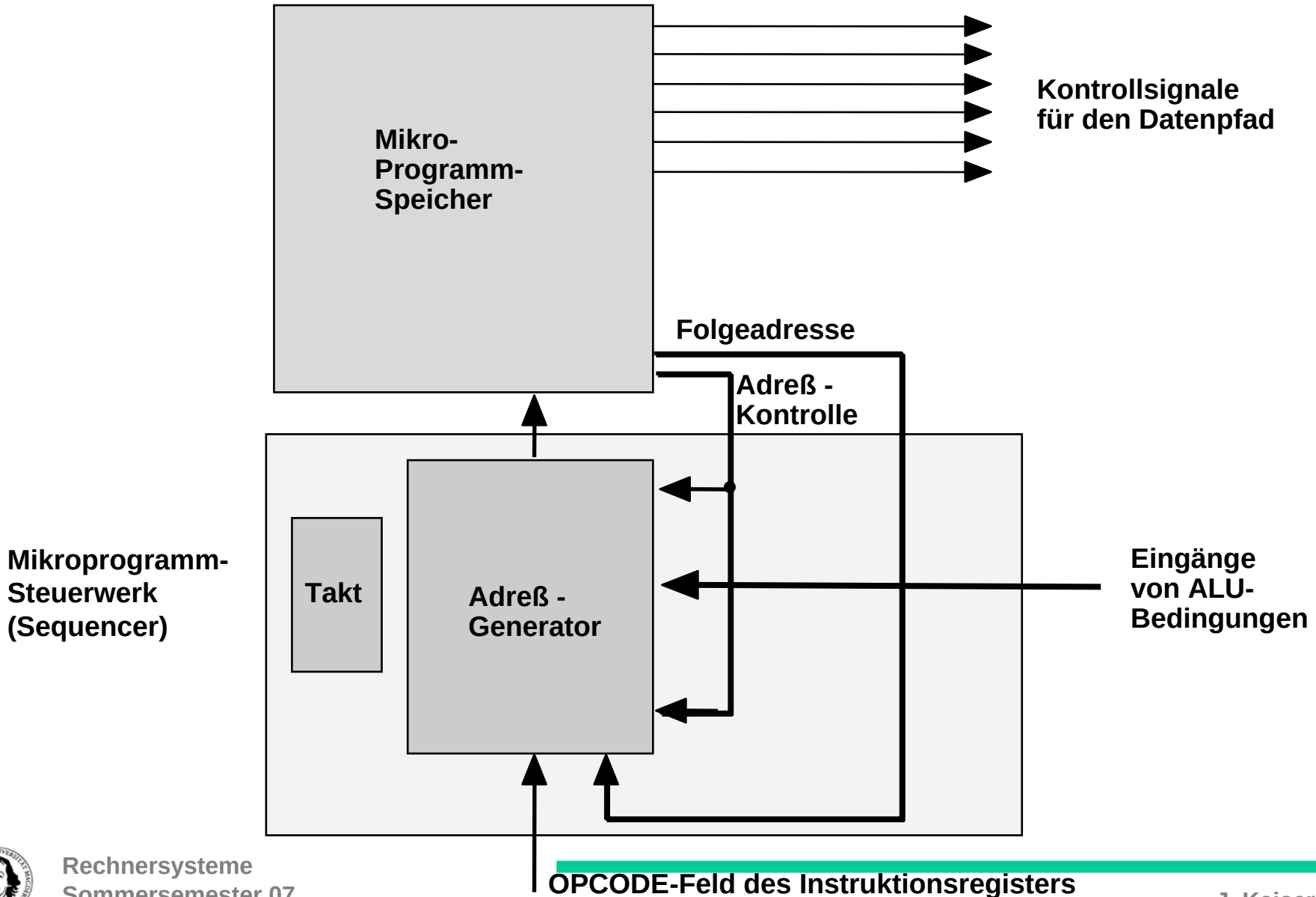
... I realized that the solution was to turn the control unit into a computer in miniature by adding a second matrix to determine the flow of control at the microlevel .....

Maurice Wilkes,
Memories of a Computer Pioneer

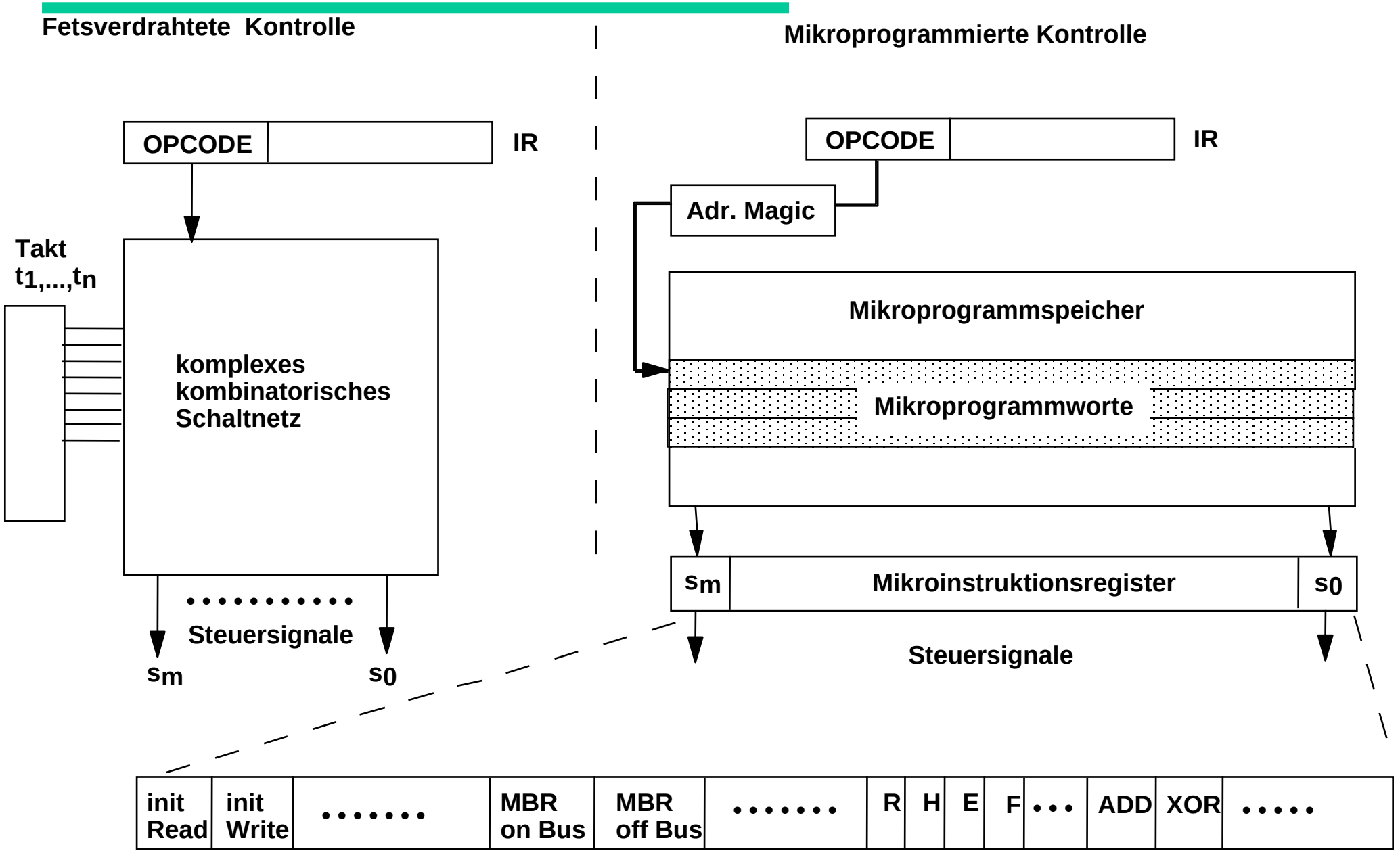
Alternativen zur Spezifikation und Realisierung der Kontrolleinheit



Mikroprogrammierte Kontrolleinheit



Alternativen beim Entwurf der Kontrolleinheit



Aufgaben des Mikroprogramm-Steuerwerks (Sequencer)

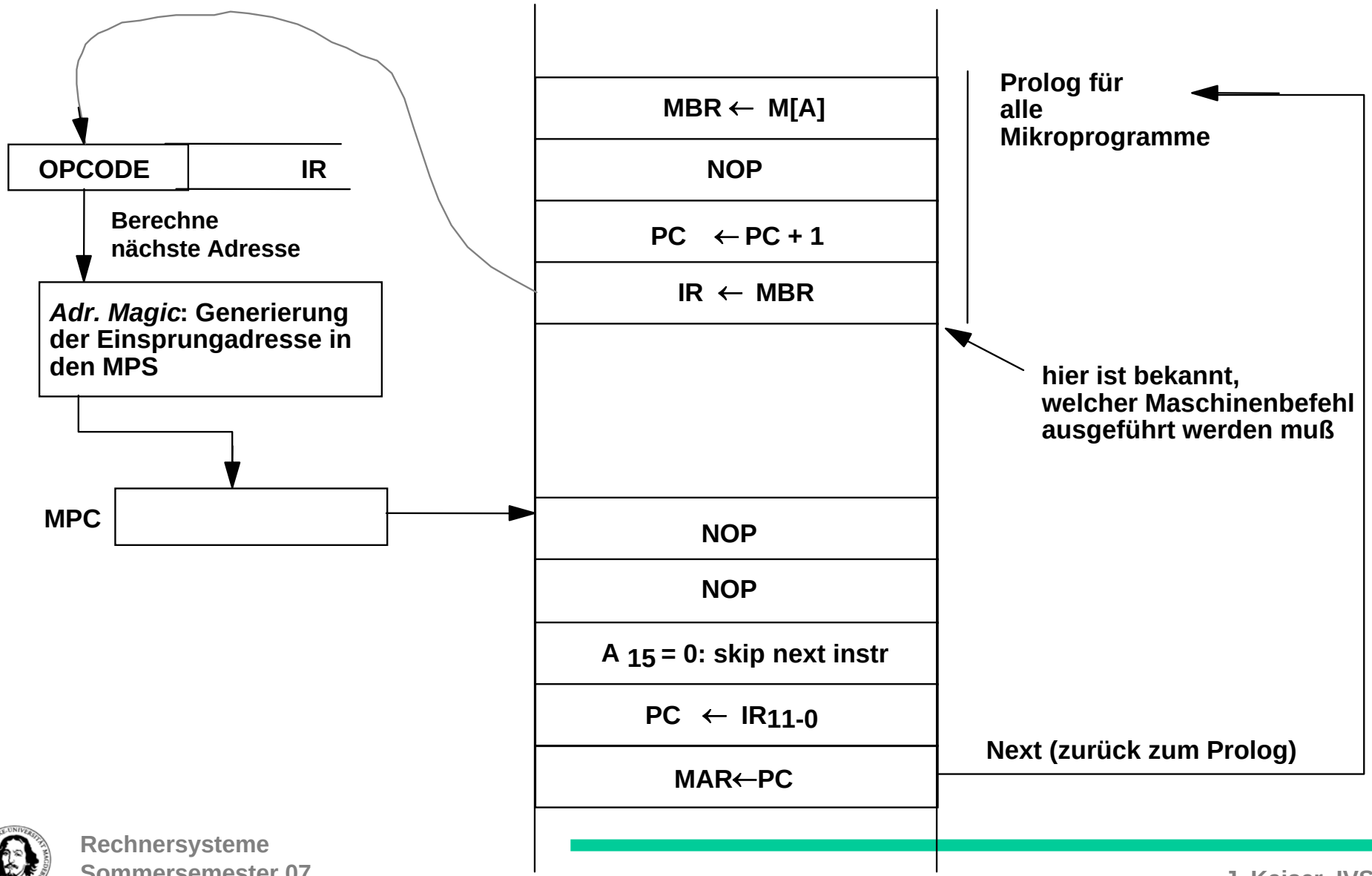
Generierung der "Adresse der nächsten Mikroinstruktion"

"Nur" Adreßberechnung !

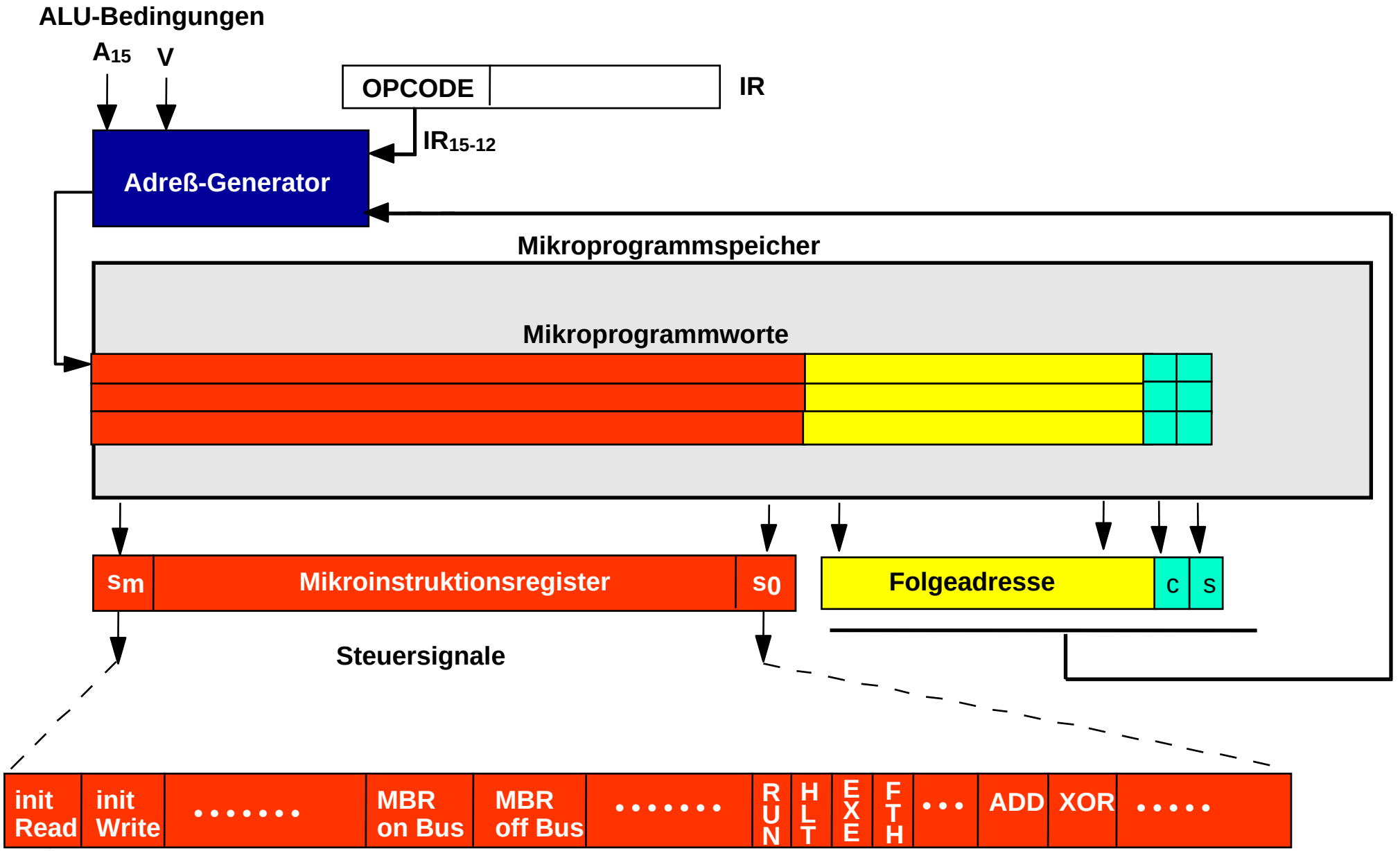
Standardeigenschaften:

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

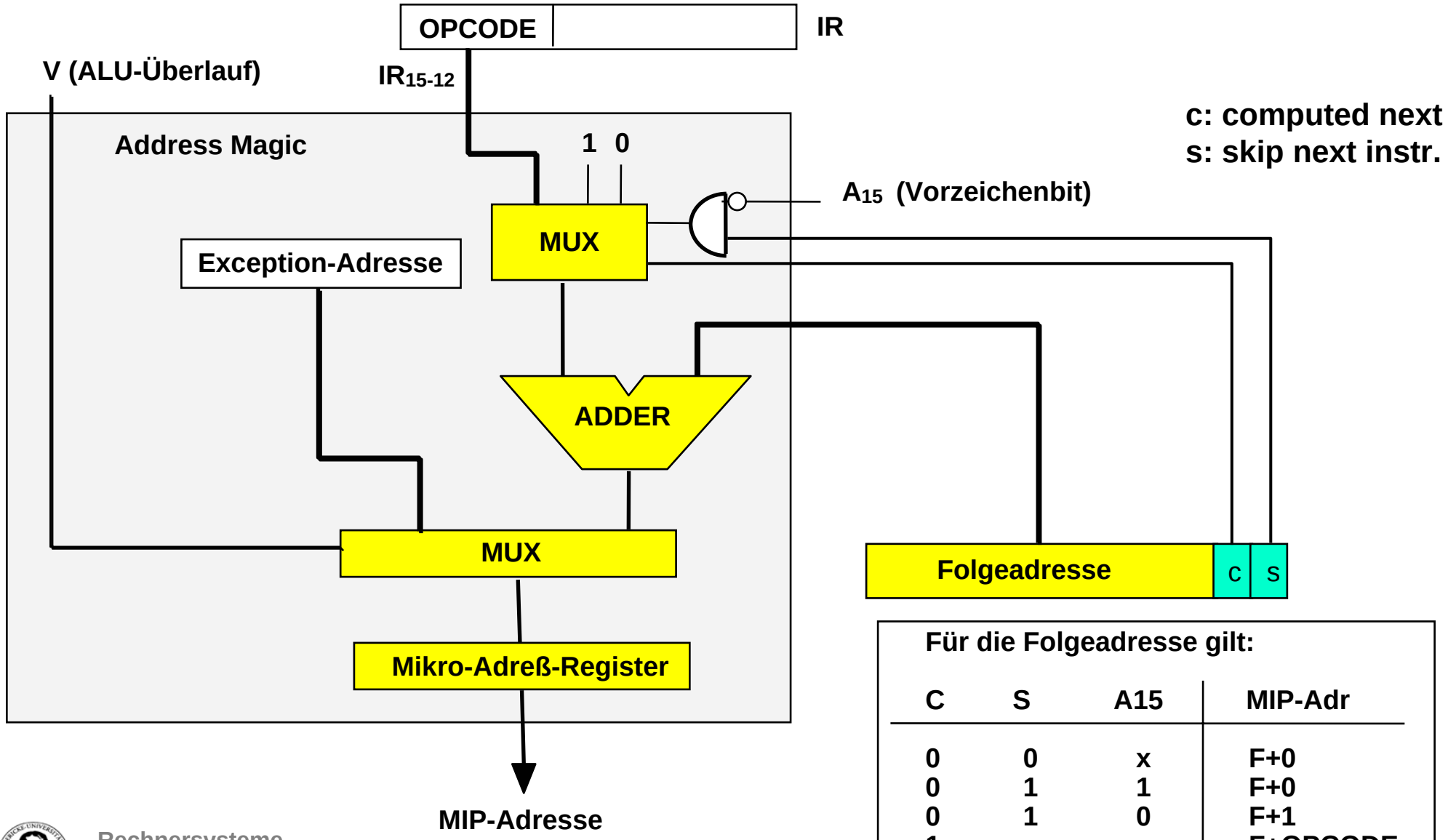
Beispiel für die mikroprogrammierte Implementierung des JMA-Befehls



Mikroprogrammierte Kontrolle



Adressierungseinheit



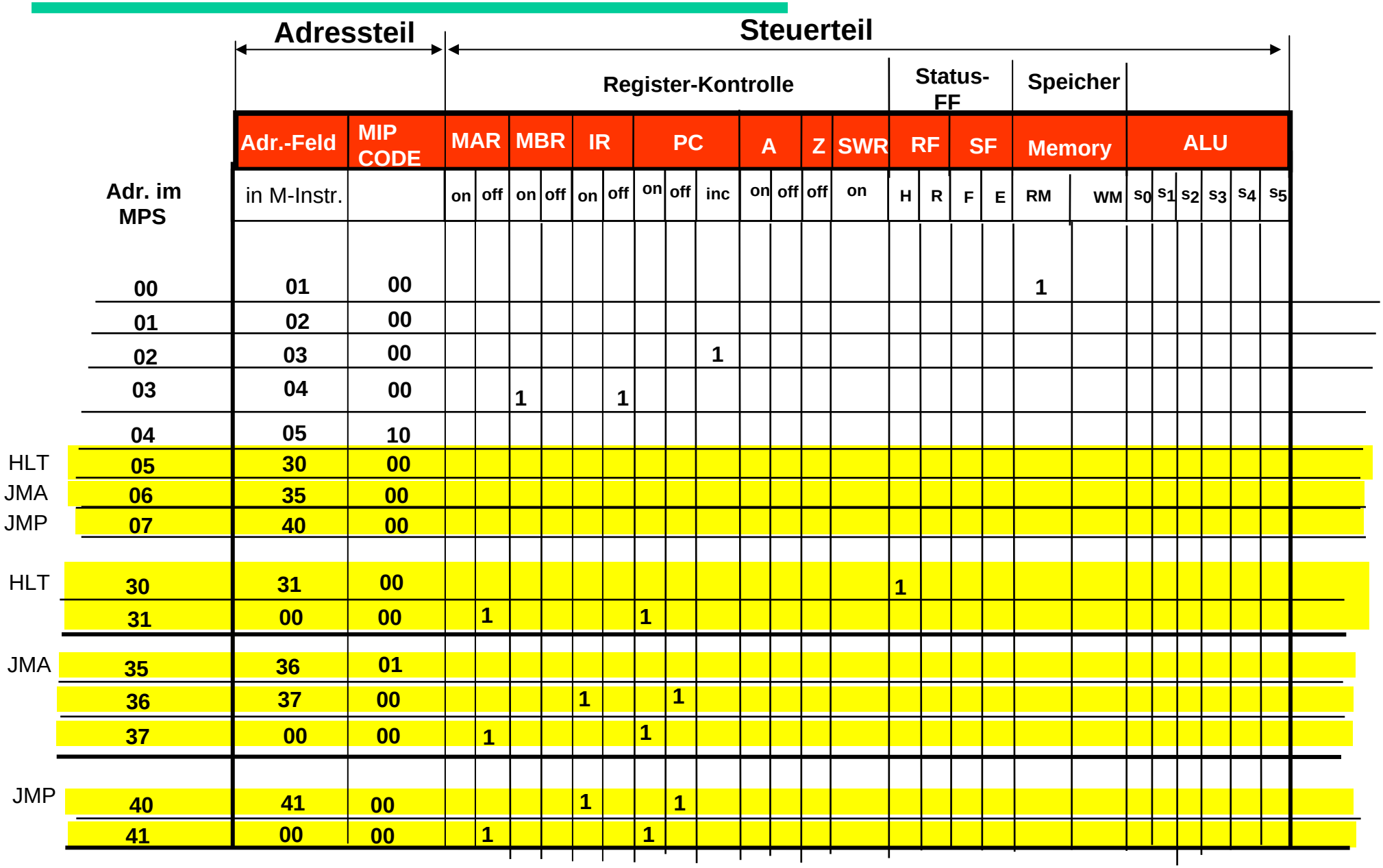
c: computed next
s: skip next instr.

Für die Folgeadresse gilt:

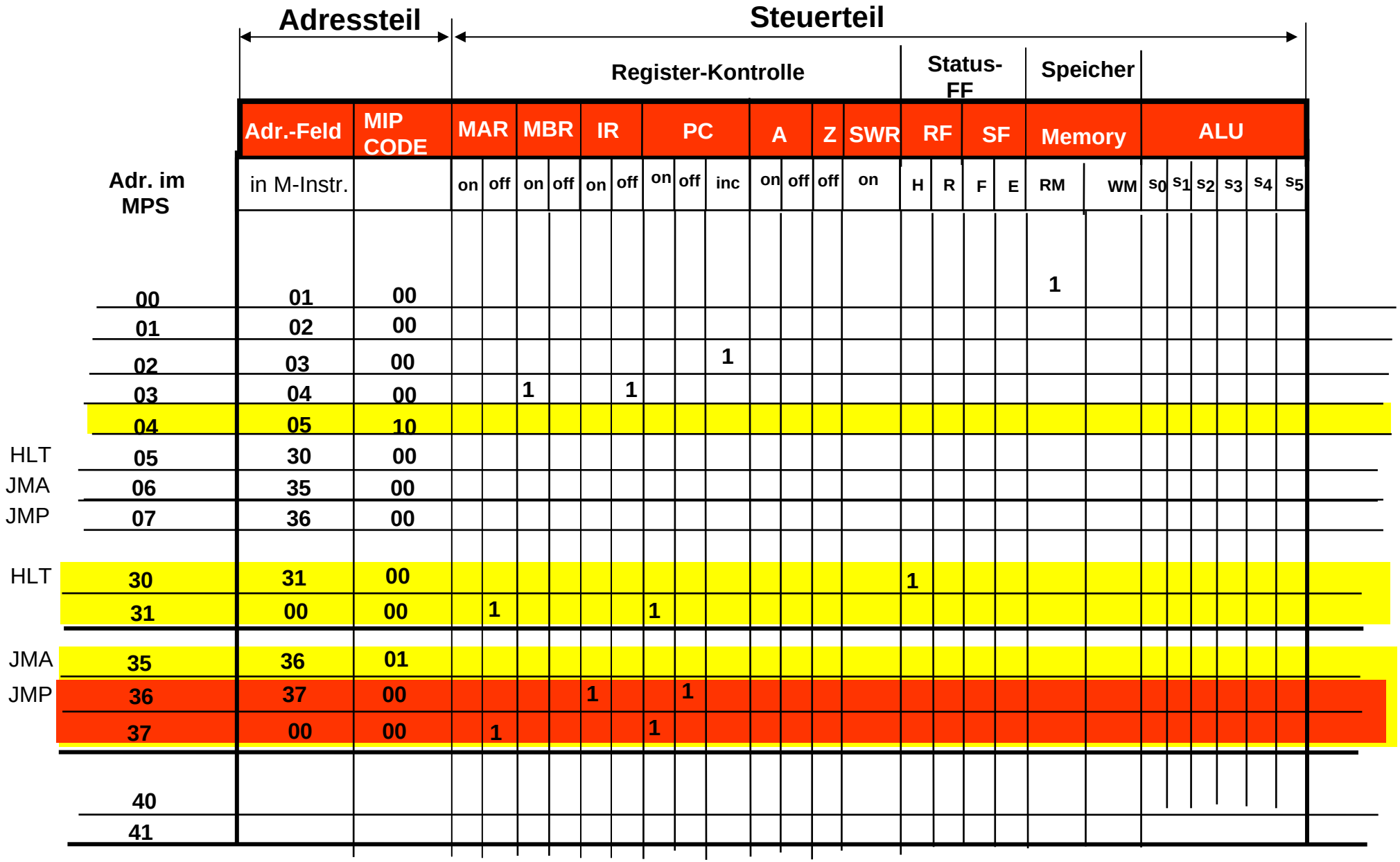
C	S	A15	MIP-Adr
0	0	x	F+0
0	1	1	F+0
0	1	0	F+1
1	x	x	F+OPCODE

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	
03	0	0	04	IR ← MBR	IR off Bus, MBR on Bus
04	1	0	05		
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



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 für die Mikroprogrammierung:

- Wie kann der M-Code optimiert werden ?

  Zielkonflikt: Hardwarekosten gegen Geschwindigkeit

- Wie kann das Mikroinstruktionswort verkürzt werden ?

  Zielkonflikt: Wortlänge gegen Dekodierungsaufwand und Flexibilität

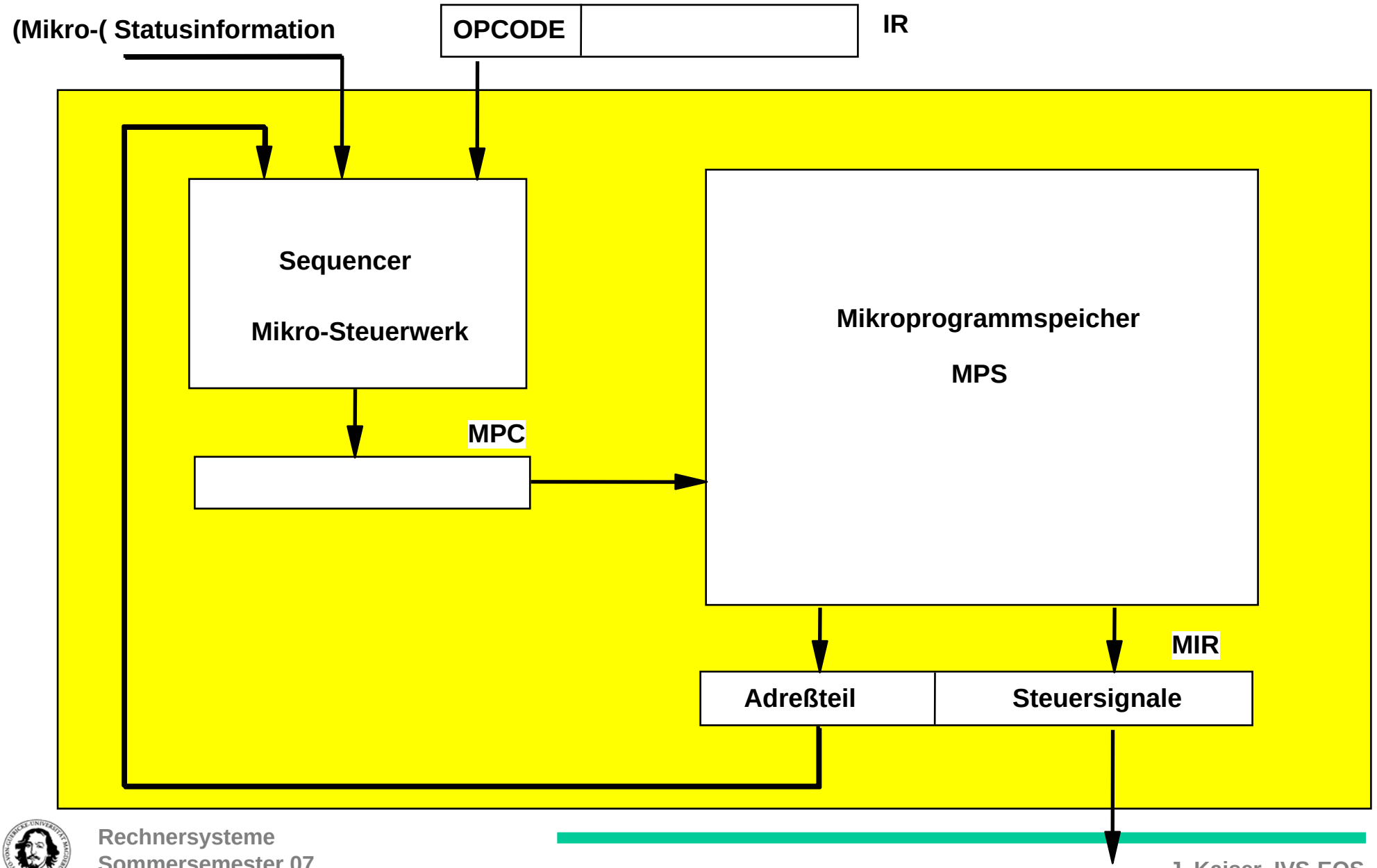
- Wie erreicht man größtmöglichen "Komfort" bei der Programmierung ?

  Zielkonflikt: Komfort gegen Effizienz

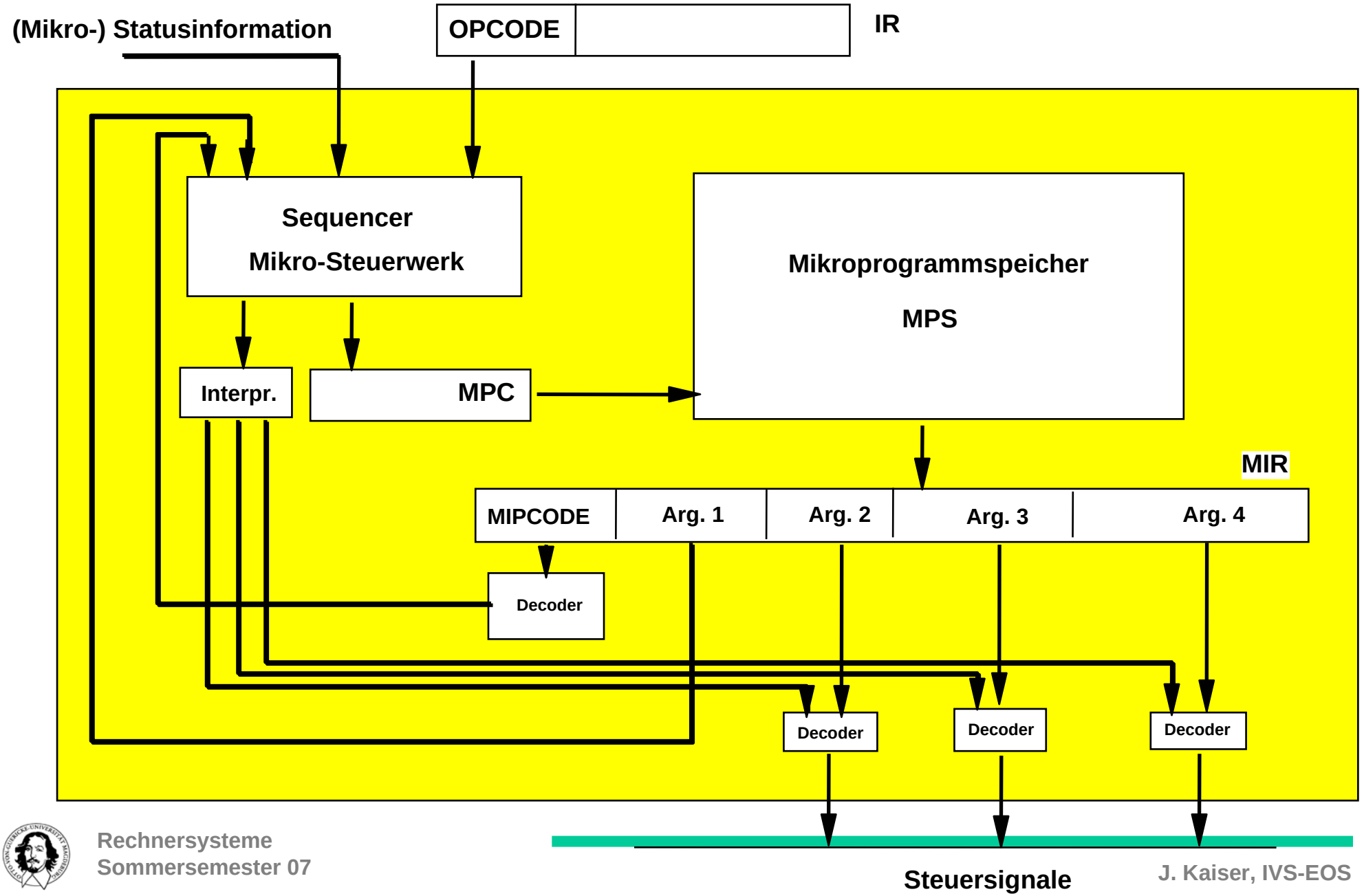
- Wie erreicht man (mit konzeptionellen Mitteln) möglichst hohe Leistung ?

  Zielkonflikt: Leistung gegen Kosten

Mikroprogrammierte Kontrolleinheit (horizontale MP)



Mikroprogrammierte Kontrolleinheit mit mehrstufiger Dekodierung



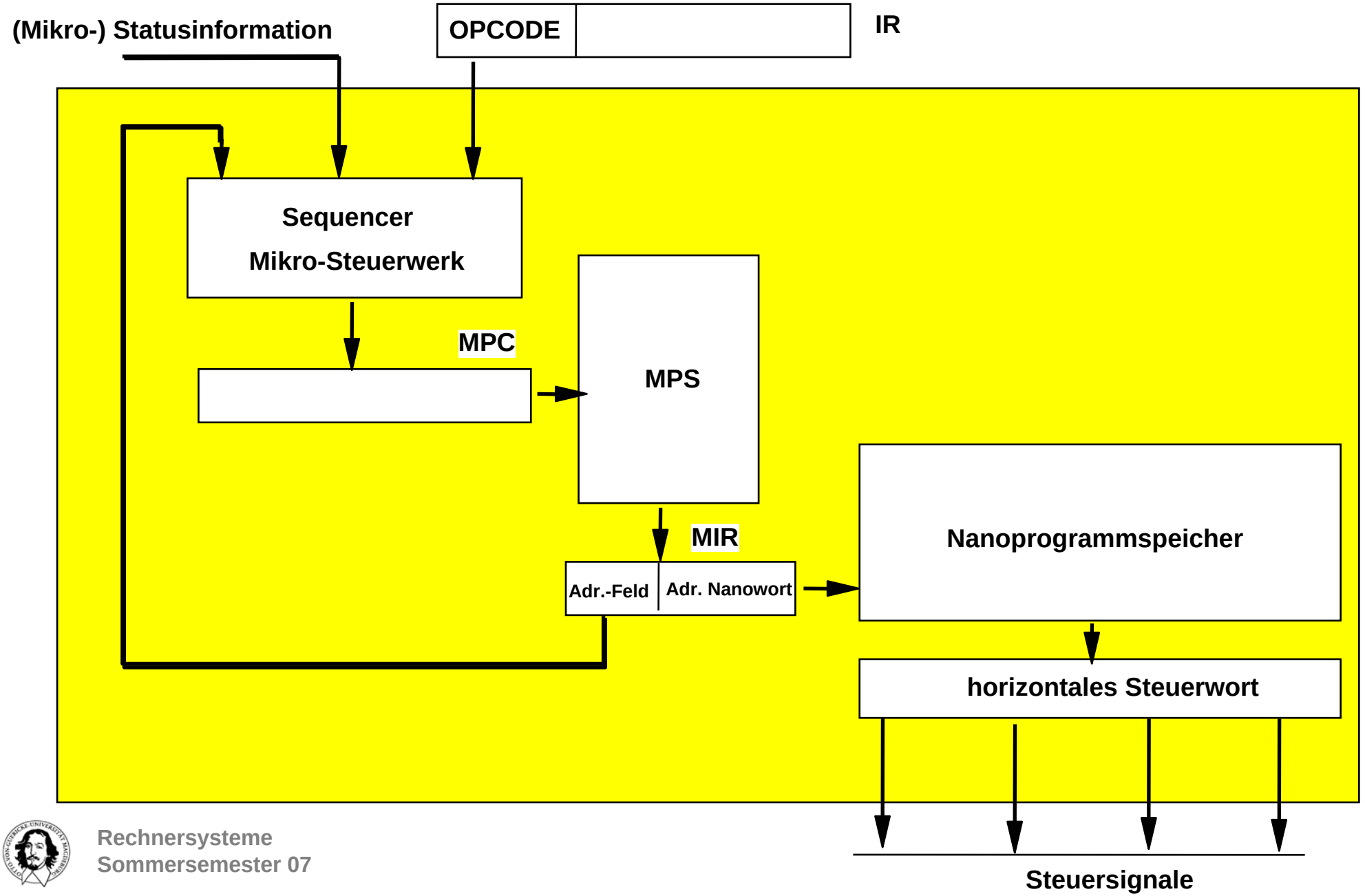
Vertikale Mikroprogrammierung

Mikroinstruktionswort ist in Felder aufgeteilt, die codierte Steuerinformation enthalten

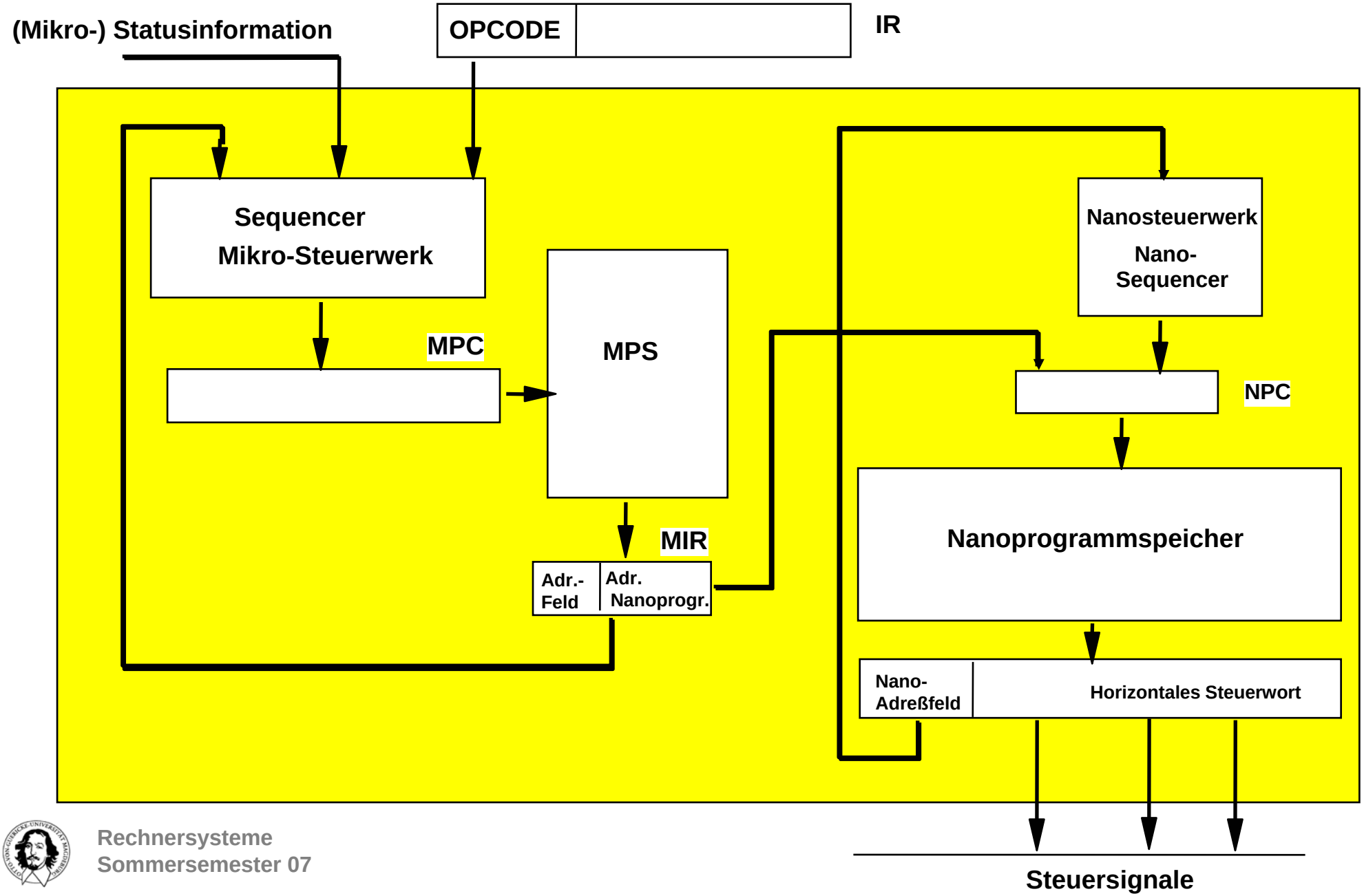
Nachteil : Zusätzliche Dekodierungshardware, Flexibilität, Geschwindigkeit

Vorteil : Kurzes Mikroinstruktionswort, kleiner Mikroprogrammspeicher
Übersichtlicher Mikroprogramme

Nanoprogrammierte Kontrolleinheit Stufe 1 (Quasi-Nanoprogr.)



Nanoprogrammierte Kontrolleinheit Stufe 2 (Echte Nanoprogr.)



Nanoprogrammierung:

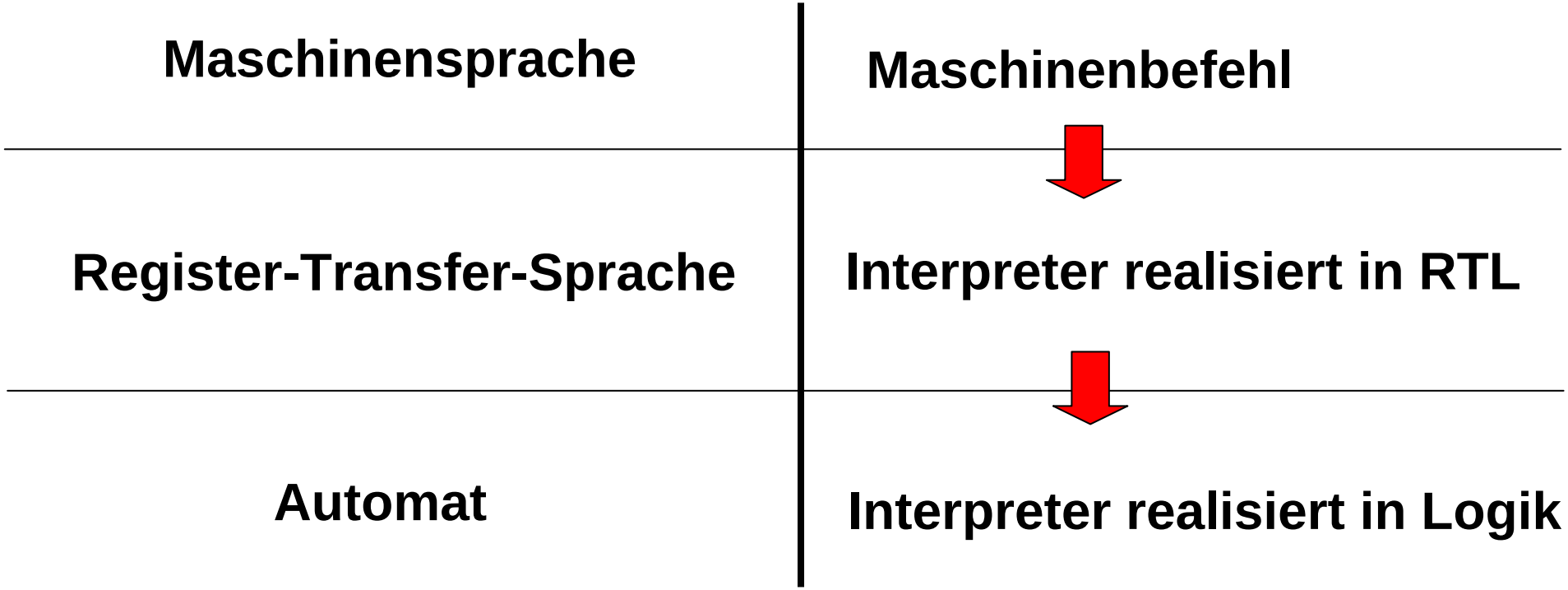
Kombination von vertikaler und horizontaler Mikroprogrammierung

Ersetzen der Dekodierungshardware durch eine weitere Stufe der Mikroprogr.

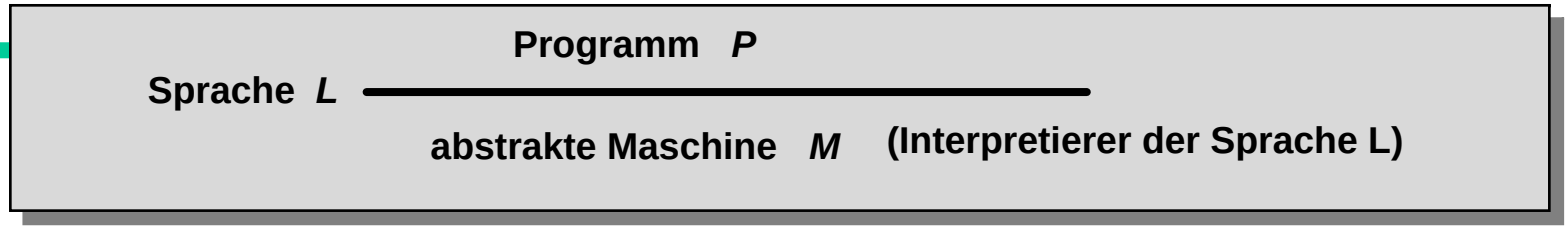
Vorteile:

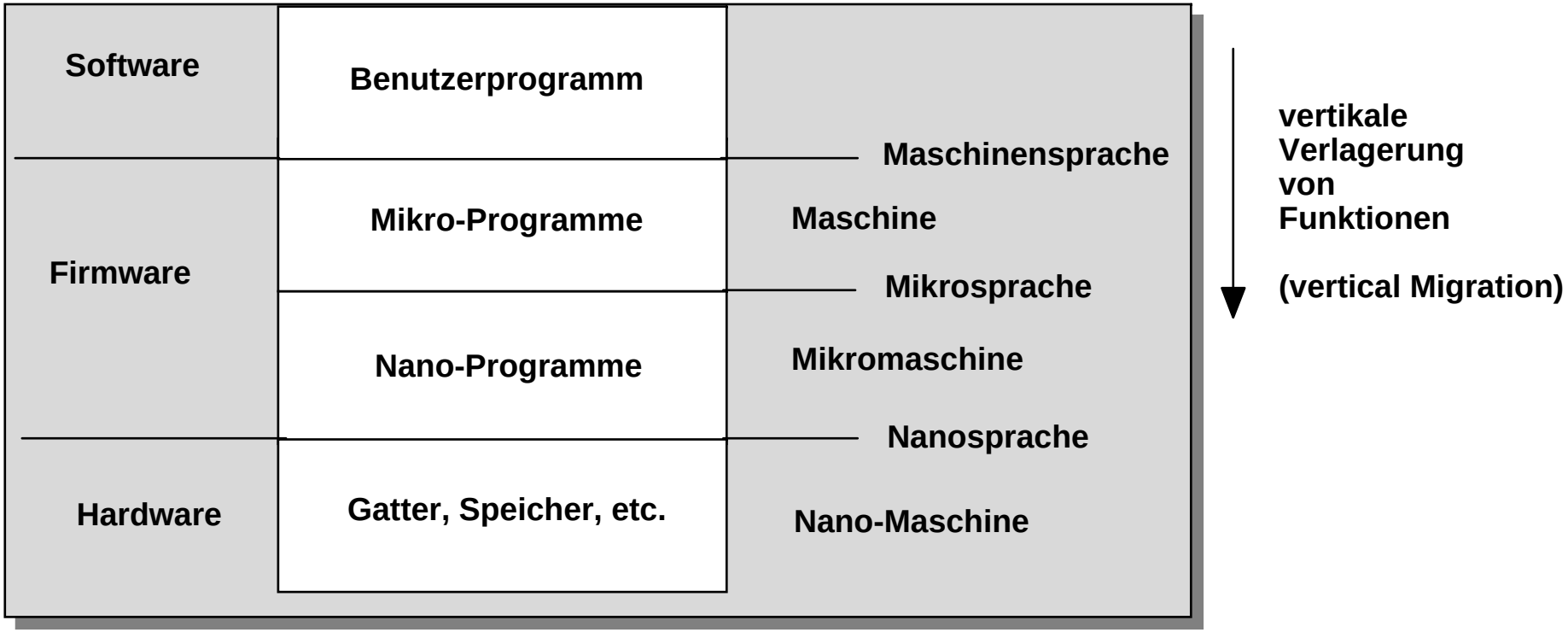
- **Flexibilität wie bei horizontaler MP**
- **Einsparung von Speicherplatz : gleiche Steuerteile können von mehreren Instruktionen genutzt werden**
- **(Festwert-) Speicher ist regulärer als ein Dekoder und leichter zu ändern**
- **Entkopplung von Mikroprogramm-Entwurf und Hardware-Entwurf**

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