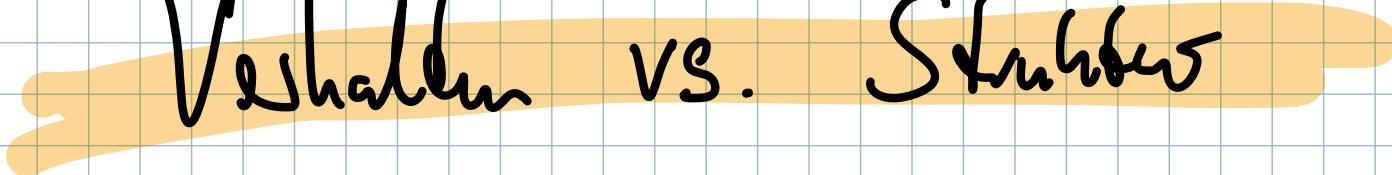




Befehlssatz



Vorhaben vs. Struktur



Architektur und Mikroarchitektur

Verhalten vs. Struktur : Architektur und Mikrowerkzeuge

Komponenten der Rechnerarchitektur (Wiederholung aus GdTI)

Einführung und Verhalten

Befehlszypern

Operandenadressierung

Speicheradressierung

Mikroprogrammierung

Lernziele:

Unterschied zwischen Verhalten und Schaltungsprinzip Rockwells beschreiben

Zusammenhang Architektur und Mikroarchitektur benennen

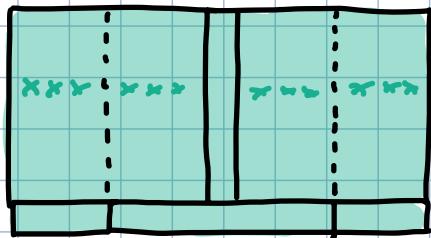
Grundlegende Architekturen beschreiben zu können

Verskribe wie Operand aus dem Speicher transparent werden

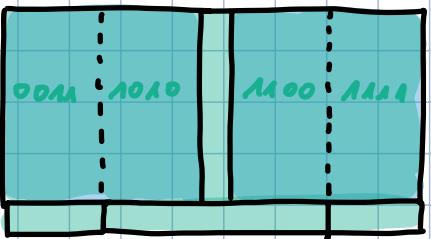
Adressierung des Speichers unter technologisch Randbedingungen  
berechnen und bewerten können.

# Komponenten des Recherausführers (beruhend aus Einmallege der Technischen Informatik)

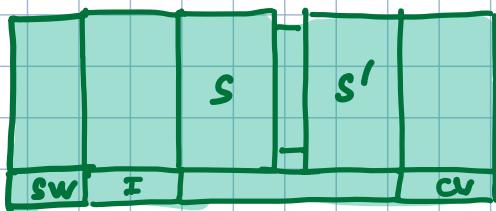
Ablaufzähler (Steuerwerte)



PLA



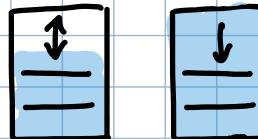
Mikroprogramm Steuerwerte



Arithmetische Schaltungen

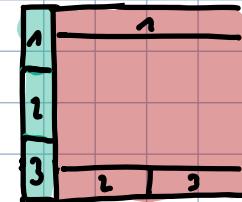
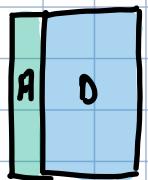


Speicher



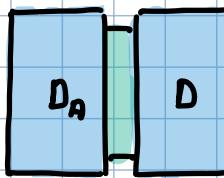
LIFO      FIFO

Stack

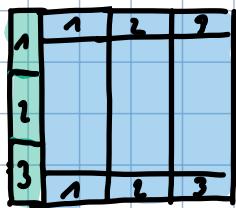


Mehrstu - Register - Satz

Vereinbarungen



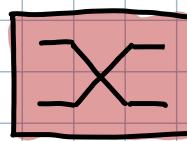
Associativ - Speicher  
CAM



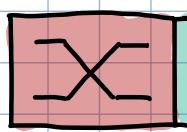
Mehrstu - Speicher



Multiplizierer



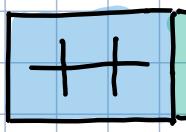
Inverter



Inverter mit DMA



Bus



Bus mit DMA

# Befehlssatz : Beschreibung des Verhaltens

Befehl : Operation (ins)  
(Instruction)

Source

Argumente  $S_1, S_2$  des Befehls

Speicherelle des Ergebnisses D

Speicherstellen

$$z = a + b$$

Beispiel :

$$d = a \text{ ins } b$$

$$d \quad S_1 \text{ ins } S_2$$

$$\text{lhs} \quad d, S_1, S_2$$

$$\text{ins} \quad S_1, S_2, d$$

Speicherelle Operation Speicherelle

RTL ( Register Transfer Level )

Befehl steht vorne !

# Entwurf eines Rechners

- 1) Architekturprinzip
  - 2) Operandauswahl
  - 3) Adressierungswahl
- 
- The diagram consists of three numbered steps on the left, each with a vertical line pointing downwards to a yellow box on the right. Step 1 points to a yellow box containing 'Architekturprinzip'. Step 2 points to a yellow box containing 'Operandauswahl'. Step 3 points to a yellow box containing 'Adressierungswahl'.

## Struktur des Mikroarchitekten

Da wir die Operandentypen nicht kennen:

- i) Befehle (Liste Beispielhafter Befehl)
- ii) Operandentypus (Rechenwahrschau)
- iii') Adressierungswahl

## Auswahl der Befehle

Befehlsvölker (Formate)

Beschreibung der Funktion (Verhalten der Befehle)

$d \leftarrow \text{add } S_1, S_2$

Operandenauswahl  
Speziesadressierung

# Auswahl typischer Befehle

## Arithmetisch-Logische Befehle

add addition

sub subtraction

and Logical and

nor Logical nor

or Logical or

sll Shift left logical

srl Shift right logical

nop no operation

## Transport

ld load

st store

lw move

ps push

pp pop

## Sprunge

j jump

jal jump and link

jr jump register

jsr jump to subroutine

rts return from subroutine

## Vergleichen

bne branch not equal

beq branch on equal

bfs branch on <flag> set

blt branch less than

bgt branch greater than

Befehle + Dashes

Machine + Sprinkler

Befehle haben Argumente

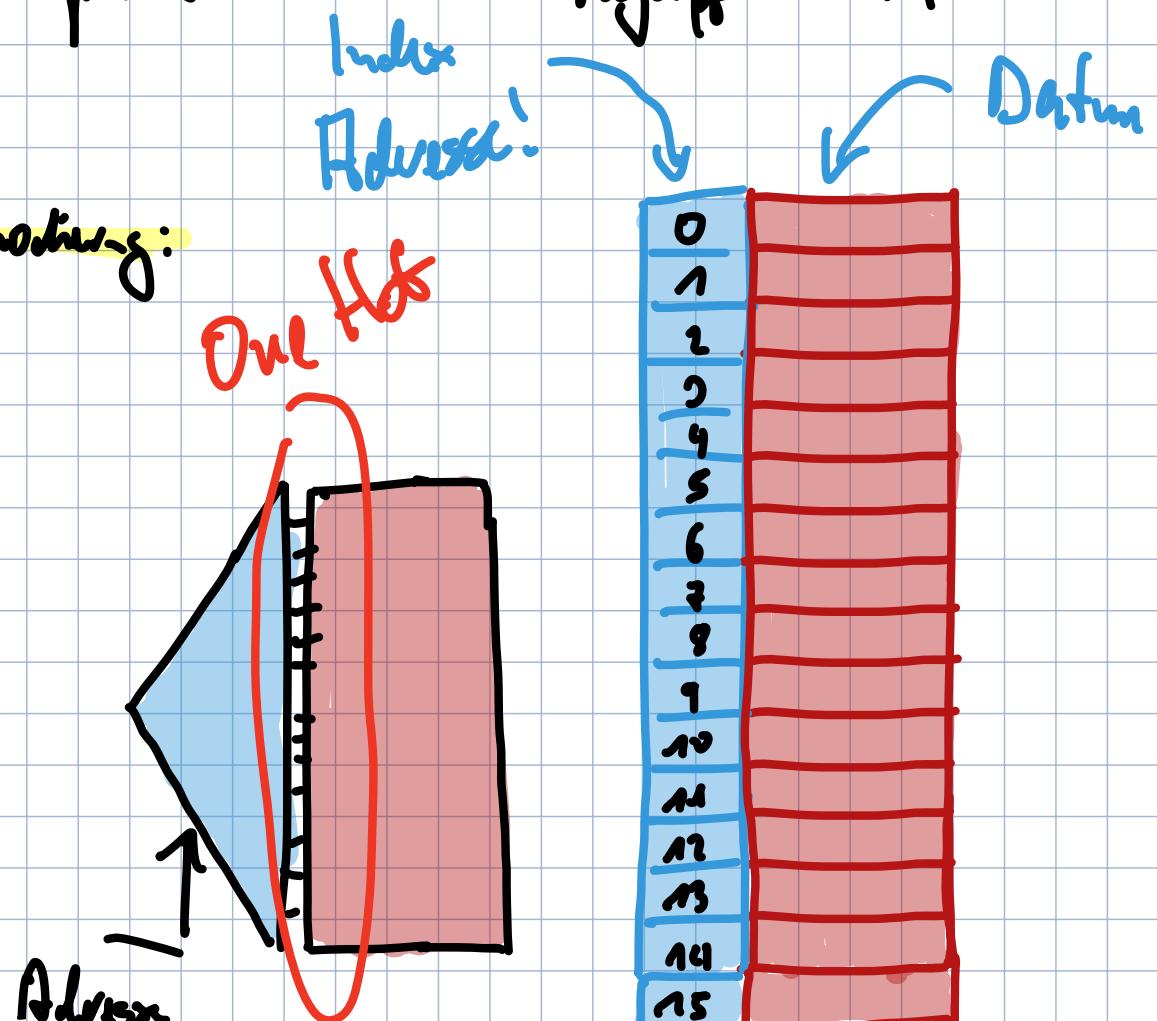
$$d = t + s$$

Befehl und Daten stehen in gleichen Speicher

v. Neumann Prinzip

Wahlweise Speicher benötigt Adressdekodierung:

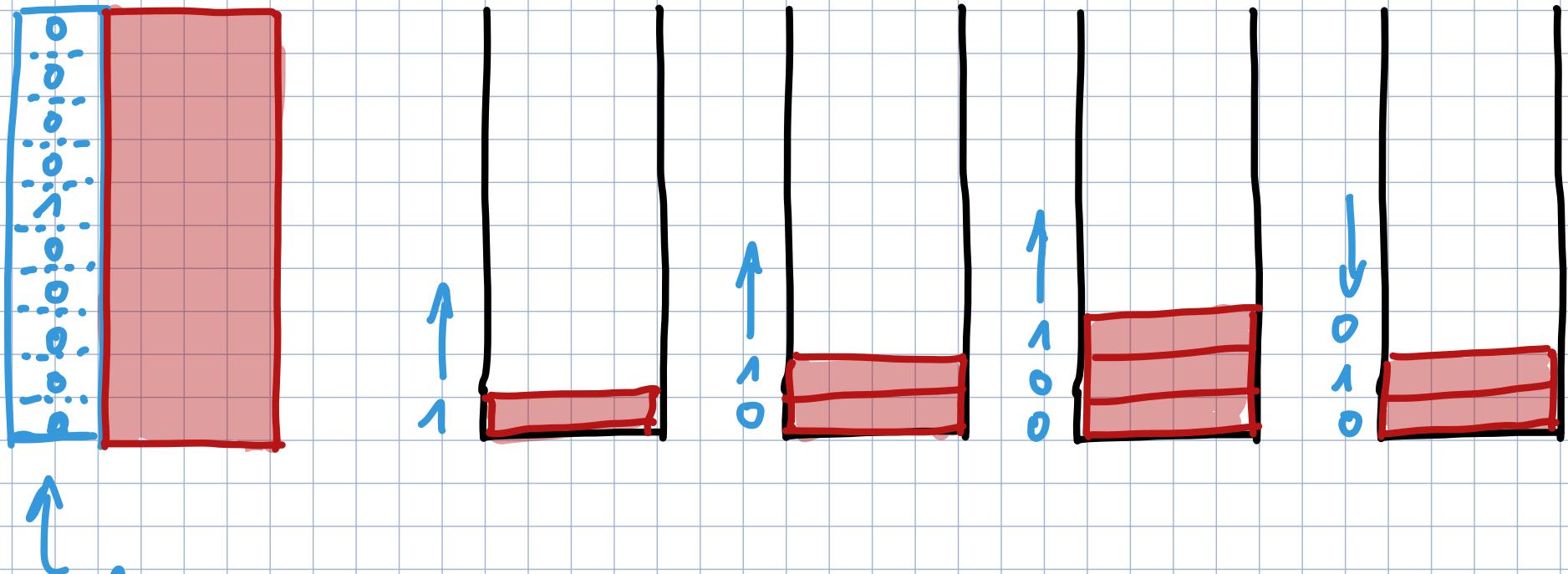
0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 1
0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1 0
0 0 1 0	0 0 0 0 0 0 0	0 0 0 0 1 0 0 0
0 0 1 1	0 0 0 0 0 0 0	0 0 0 0 1 0 0 0
0 1 0 0	0 0 0 0 0 0 0	0 0 0 1 0 0 0 0
0 1 0 1	0 0 0 0 0 0 0	0 0 1 0 0 0 0 0
0 1 1 0	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0
0 1 1 1	0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
1 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0 0
1 0 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0
1 0 1 0	0 0 0 0 1 0 0	0 0 0 0 0 0 0 0
1 0 1 1	0 0 0 0 1 0 0	0 0 0 0 0 0 0 0
1 1 0 0	0 0 0 1 0 0 0	0 0 0 0 0 0 0 0
1 1 0 1	0 0 1 0 0 0 0	0 0 0 0 0 0 0 0
1 1 1 0	0 1 0 0 0 0 0	0 0 0 0 0 0 0 0
1 1 1 1	1 0 0 0 0 0 0	0 0 0 0 0 0 0 0



Speicher mit Wahlfunktion  
Zugriff: RAM  
Liste, Band oder Turing  
maschine

Adressdecodierung benötigt viele Schalter

Wenn Schalter fehlt: Einfallsh. Sprühs: Stopel - Stark



Adressung durch  
Schiebungsschalter

# Stackelmaschine : Befehlsatz und Mikroarchitektur

\$a Speicheradresse \$A Register  
a Konstante

opc = opcode

Befehlswort

opc

Beispiele

ps  
pp

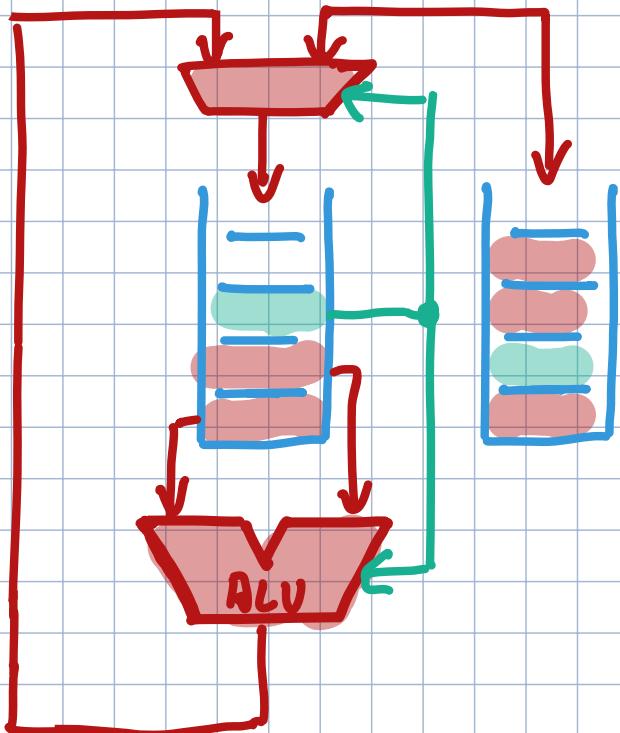
odd  
a  
b

O. Adressmaschine

Eine Stackelmaschine  
benötigt kein Stackreg.

UPN-Semantik

Struktur (Mikroarchitektur)



Direct Zugriff speicher sind aufwendig, da eine grosse Rechenzeit.  
Logisch implementiert werden muß.

## Universal Rechner

$$d = r + s$$

add \$d, \$r, \$s

## Vierstufiger (Befehlszyklus)

IF Hole Befehl  
Adresse

    cold  
    d  
    r  
    s

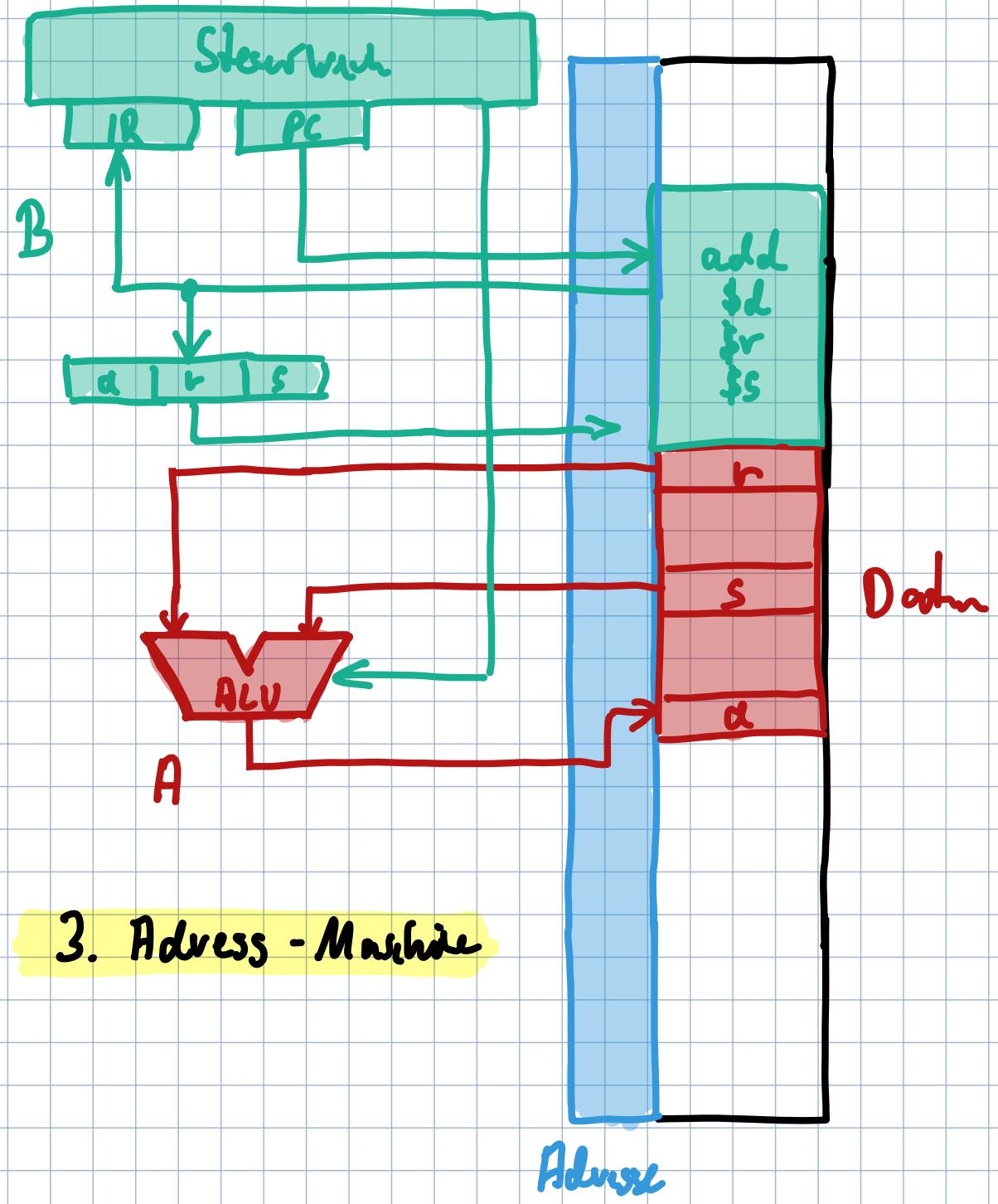
ID Befehl dekodieren

OF Hole Operant aus  
    Operant an

    r  
    s

EX Operation ausführen

OS Speichere Ergebnis in d



## 3. Adress-Maschine

**Rechenleistung:** Ausführen eines Befehls

$$d = r + s$$

**Vierstufiger (Befehlszyklus)**

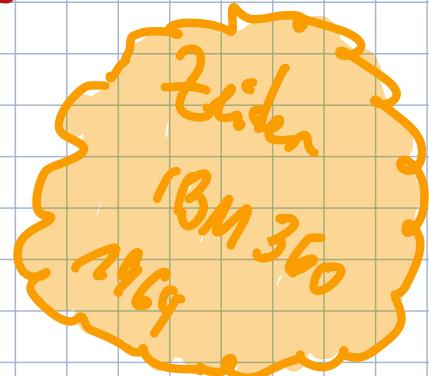
IF	Hole	Befehl Adresse	gold	2 $\mu$ s
			d r s	6 $\mu$ s
ID		Befehl abholen		0,7 $\mu$ s
OF	Hole	Operande aus r		
		Operande an s		4 $\mu$ s
EX		Operation ausführen		0,7 $\mu$ s
OS		Speichere Ergebnis in d		2 $\mu$ s

**Ausnutzen:** Adressbereich =

Wortbreite = 8-Bit

add \$d, \$r, \$s

**Technologie:**



Potential zu Zeit  
einsparen!

Wie?

Speicherzugriffe: 14  $\mu$ s  
Rechnen: 1,4  $\mu$ s

1:10

\$d  
\$r  
\$s

Register : Sprüche in gleicher Technik

$$d = r + s$$

add \$s

Vorhalte (Befehlszyklus)

IF Hole Befehl  
add  
Adress

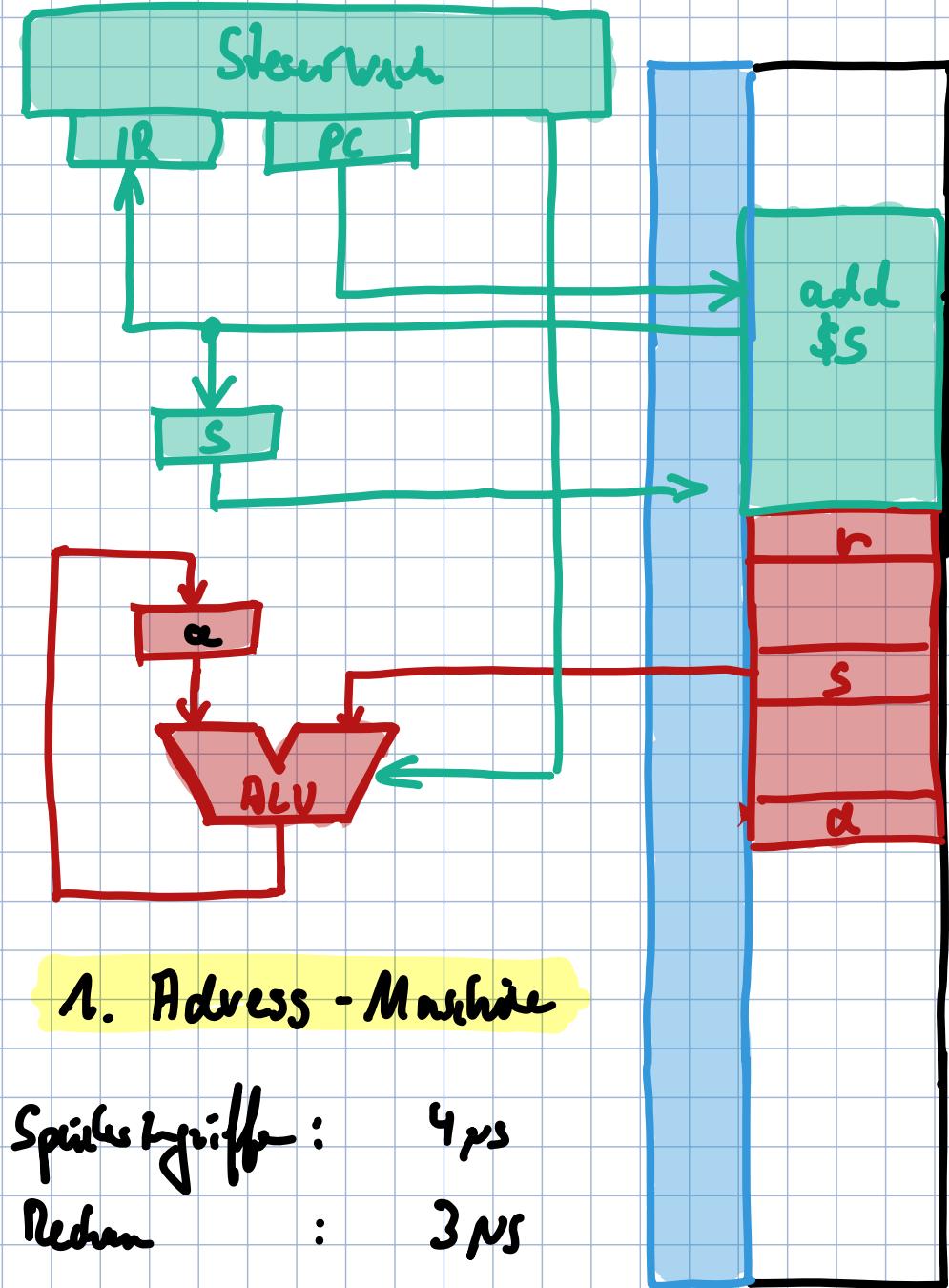
r  
s

ID Befehl dekodieren

OF Hole Operat an s

EX Operation ausführen

OS Speichere Ergebnis in d



1. Adress - Maschine

Sprüchezygiffe : 4 ns

Reduzen : 3 ns

~ 3:4

Adresse

Register sind kleine Speicher im Rechenraum

Aus	add \$d, \$r, \$s wird	lde \$r	3,4 $\mu$ s
		add \$s	7 $\mu$ s
		sta \$d	3,6 $\mu$ s
Aber:	64 kB Speicher $\rightarrow$ 8-Bit		13,8 $\mu$ s
	128 kB Speicher $\rightarrow$ 16-Bit		<u><u>  </u></u>

3. Adressmaschine:

add (64 kB Speicher)

15,4  $\mu$ s

add (128 kB Speicher)

27,4  $\mu$ s

2. Adressmaschine

add (64 kB)

7  $\mu$ s

add (128 kB)

8,8  $\mu$ s

Durch den Adressmauker wird add nicht immer geladen als  
gespaltet werden.

17,8  $\mu$ s  
26,6

# Vervielfachung des Sprungbefehle: Mehr Register

Programm:

ld x \$s

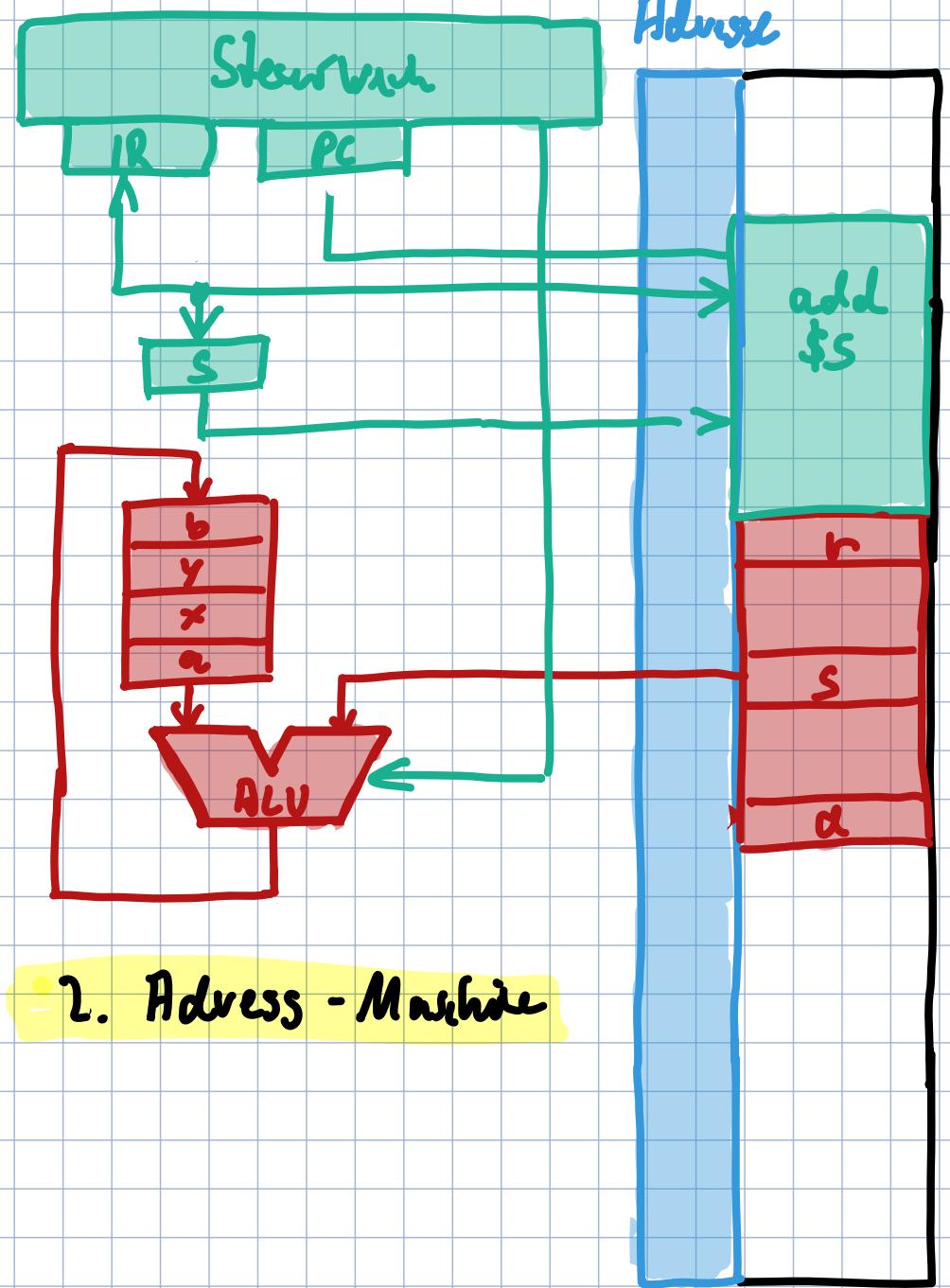
add x,\$r

tay

add y,\$s

sta \$d

mov a,y



2. Adress - Maschine

# 1. Adress-Maschine ( Mikromaschine ) : Befehlsatz und Mikroarchitektur

Von Neumann-Architektur

\$a Speicheradresse \$A Register

a Konstante

opc = opcode

Befehlswort

opc \$a

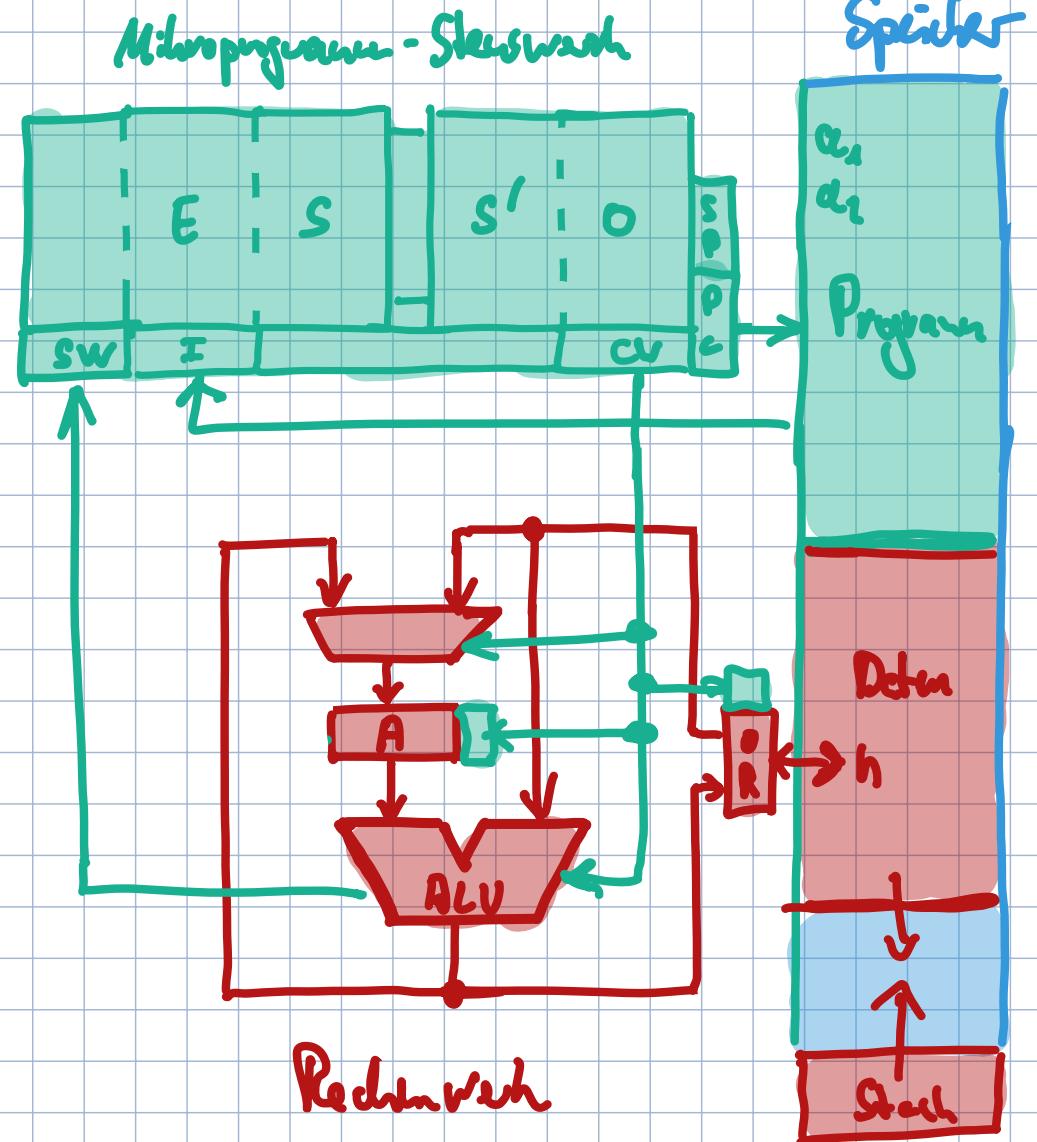
Beispiele

ld \$250

add \$781

Struktur ( Mikroarchitektur )

Mikroprogramm-Schreier



## 2. Adress-Maschine (Registermaschine) : Befehlssatz und Mikroarchitektur

$\$a$  Speicheradresse  $\$A$  Register

$a$  Konstante

$opc = \text{opcode}$

### Befehlswort

$opc \$A, \$a$

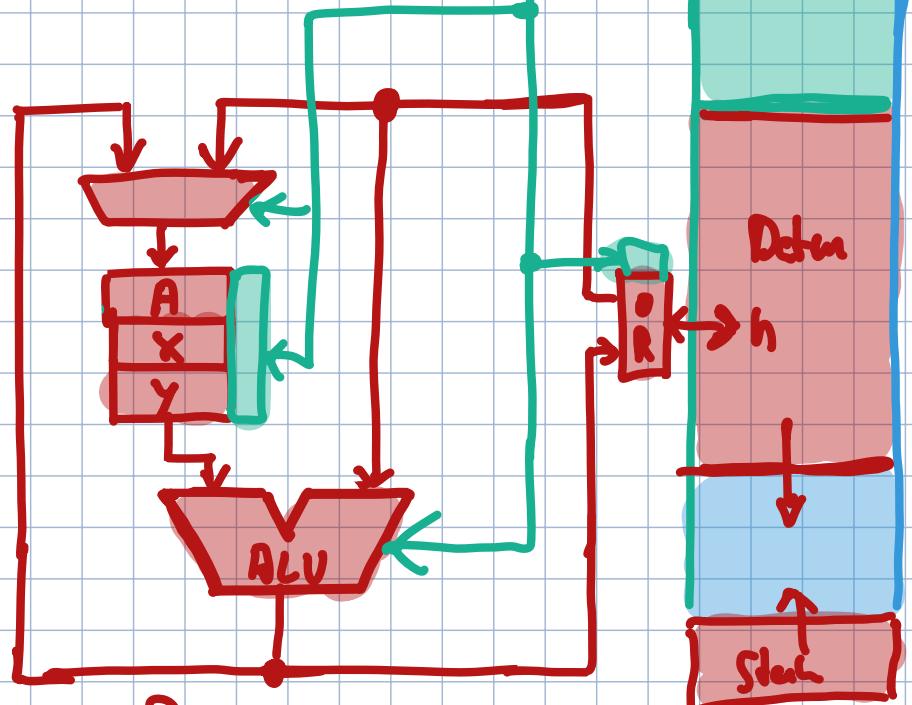
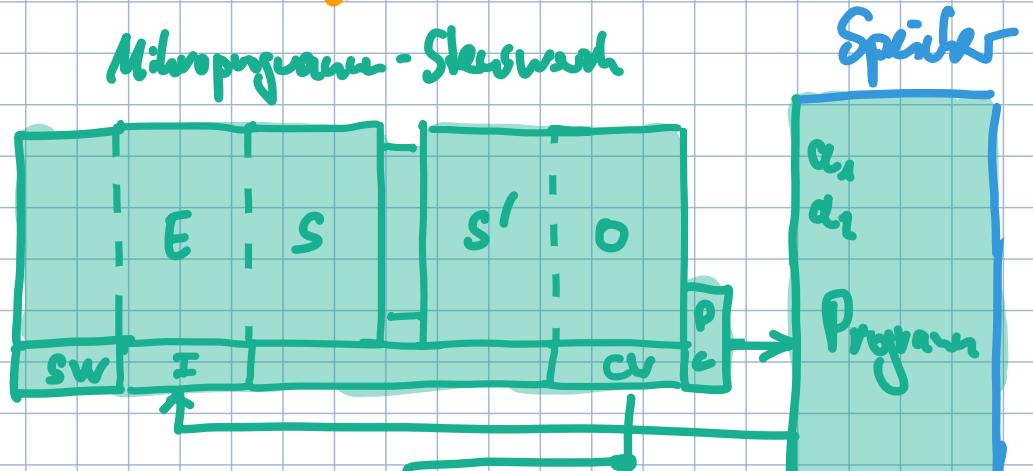
### Beispiele

$lda \$450$

$add \$x, \$781$

Stuktur (Mikroarchitektur)

Mikroprogramm-Schrein



Rechnerwerk

# I. Adress-Maschine (Universelle Registermaschine) : Befehlsatz und Mikroarchitektur

\$a Speicheradresse \$A Register

a Konstante \$D Ziel (Dest.)

opc = opcode \$S Quelle (Same)

\$T Quelle (Swap)

## Befehlswort

opc \$D, \$S, \$T

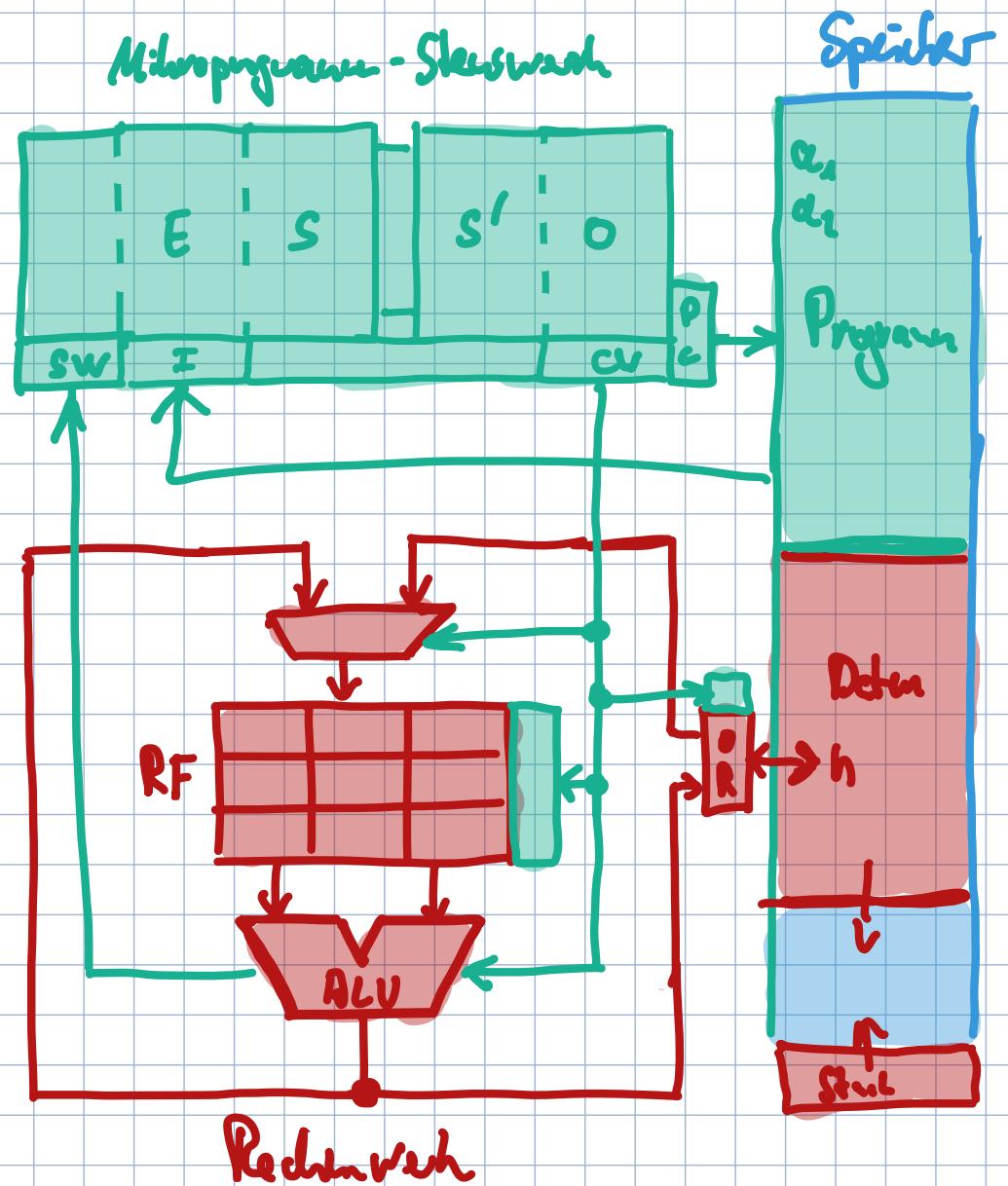
## Beispiele

ld \$D, \$78056

st \$x, \$D

add \$D, \$S, \$T

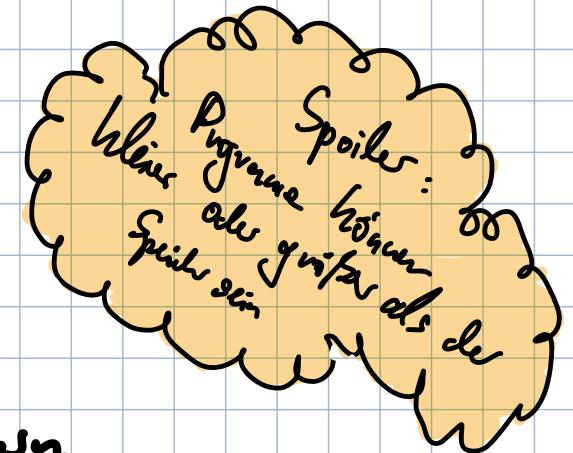
## Mikroprogramm-Schaltung



Adressierung

Befehlswort vs. Adressraum

## Technologische Herausforderung:



Wort Größe eines Rechners ist begrenzt

Das v. Neumann - Prinzip verlangt aber Adressen zu spezifizieren

Wort Größe

8-Bit

1 Byte

16-Bit

1 Halbwort

32-Bit

1 Wort

Adressraum

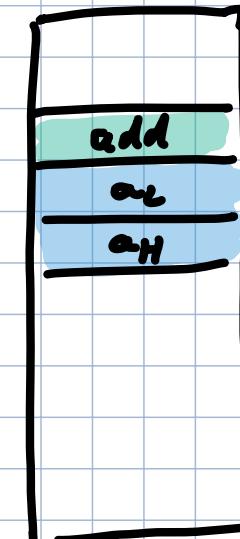
64 kB

128 kB

256 kB

Speichergröße

16 kB ?



Speichern hier  
16-Bit Adressen

Adressing Idee:

Register und im und schelle  
Maschine auswerte, in die  
Adressen zu berechnen!

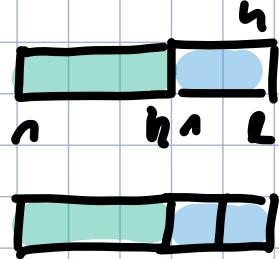
## Inherente Adressierung

Die Adresse des Operanden ist direkt im OPCODE kodiert.

## Befehlswort

$$\langle O_h z \rangle_{n=h+2}$$

$$\langle O_h X_1 Y_1 \rangle_{n=h+21}$$



Operand  $\circ$

$h \geq \text{Länge}$

Argument  $\exists$

$h, l, n = \text{Länge}$

## Beispiele:

H

ldax \$a

Lade Akkumulator

mvxa  
mvxy

Inhalt aller in X-Register  
Inhalt Y- in X-Register

## Implizite oder unmittelbare Adressierung

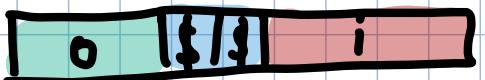
Der Befehl enthält den unmittelbaren Wert. So können Konstanten effizient verarbeitet werden.

### Befehlswort

$\langle O_k \, z, i_m \rangle_n$

 0 | z | : |

$\langle O_k \, D_l \, S_l \cdot i_m \rangle_n$

 0 | \$ | : |

Operand o       $h = \text{Länge}$

Argument z       $h, l, n = \text{Länge}$

u

### Beispiele

lda 5

Lade 5 in den Akkumulator

idx 8

addi \$D,\$S,5

Beispiel 1 und 2 kombinieren lokale  
mit impliziter Adressierung.

Beispiel 3 kombiniert die implizite  
mit der Direkten Adressierung.

## Direkte oder absolute Adressierung

In Befehlswort stehen direkt die Register- oder Speicheradressen der Operanden.

Operand 0

$h = \text{Länge}$

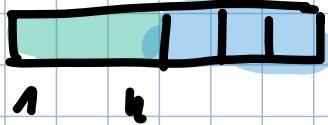
Operand 2

$h, l, n = \text{Länge}$

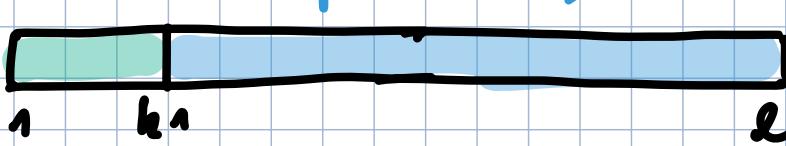
## Befehlswort

$\langle O_h D_e R_e S_e \rangle_n$

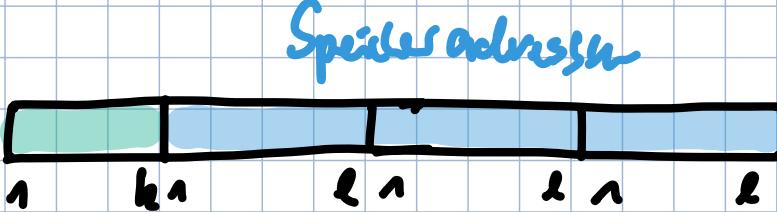
$\langle O_h \rangle_n D_e R_e S_e$



Registeradresse



Speicheradresse



Speicheradresse

Beispiele:

add \$D,\$S,\$T

add \$S

D

## Indizierte Adressierung

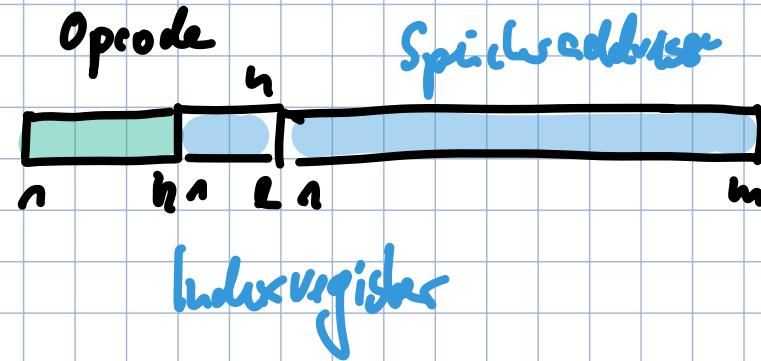
In einem Indexregister ist ein Offset gespeichert.

Dieser wird zu der des im Befehlswort angegebenen

Adresse addiert.

## Befehlswort

$\langle O_h z_i \rangle_n S_m$



Beispiele:

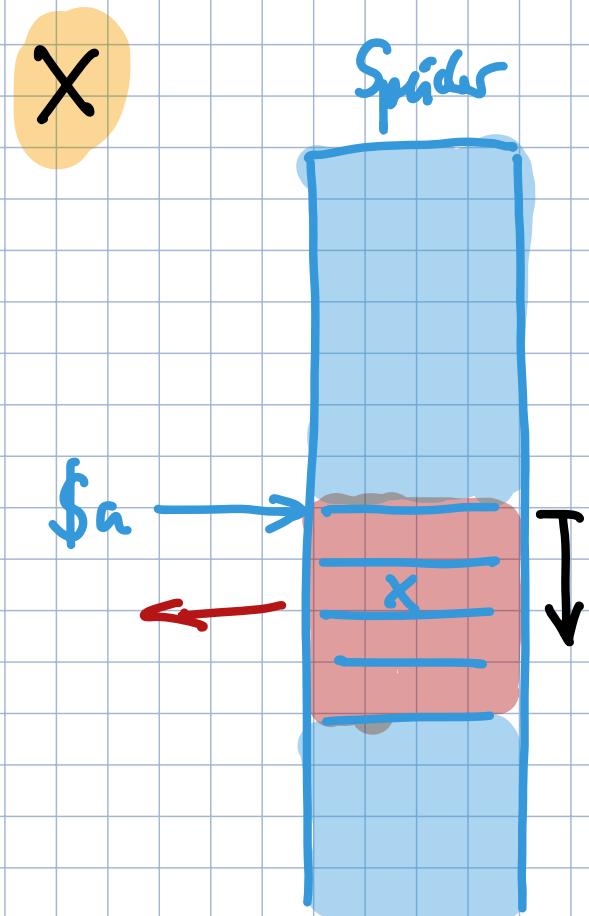
add 7576, \$X

add \$a, \$X

7576 + X

a + X

Operand 0       $h \geq \text{Länge}$   
Argument 2       $h, l, m = \text{Länge}$

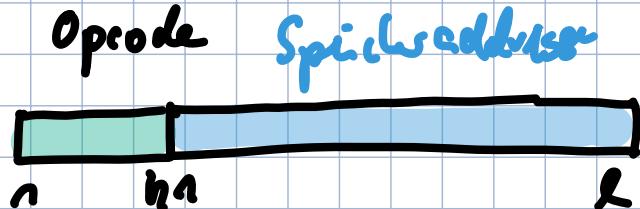


# Indirekte Adressierung

Im Befehl wird auf einen Speicher- oder Register-Eintrag verwiesen. Dieser Eintrag ist ein Verweis auf die Daten.

## Befehlswort

$\langle 0_k \rangle \underline{t}_l S_r$



$\langle 0_k \underline{t}_l \rangle_n$

## Beispiele:

add (\$x)

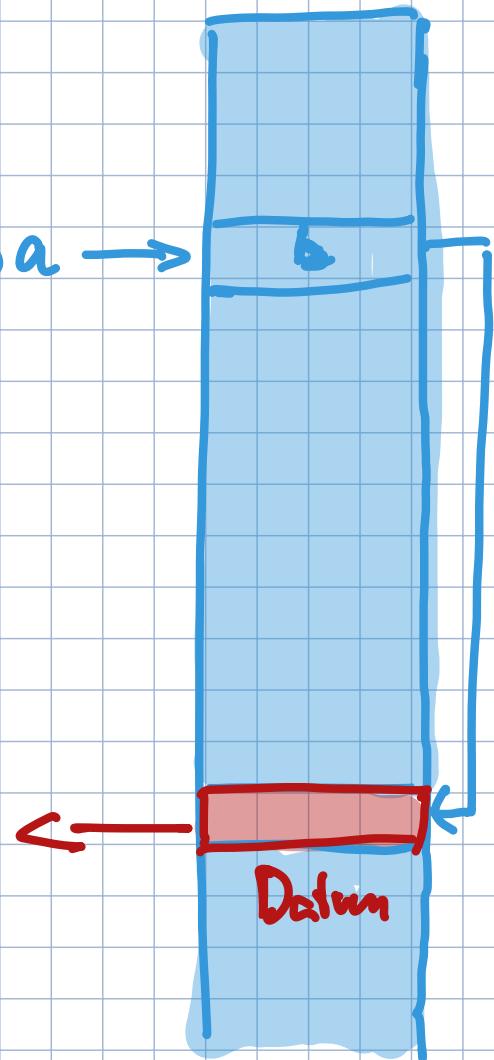
add (\$a)

Register - Indirekt  
Speicher - Indirekt

Operand 0       $h = \text{Länge}$   
Argument 2       $h, l, n = \text{Länge}$

I

\$a → b



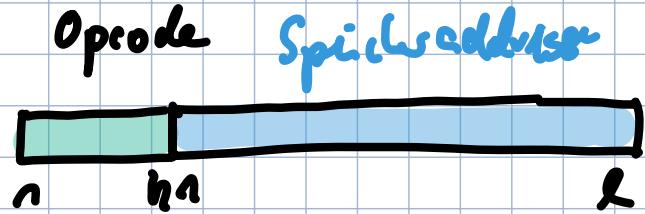
# Relative Addressierung

Die im Befehlswort angegeben Adresse wird zur Inhalt eines Registers oder einer Speicherzelle addiert.

## Befehlswort

$\langle O_h \rangle, S_L$

$\langle O_h z_1 \rangle_n$



Beispiele:

beq \$S, \$T, \$b

bne \$b

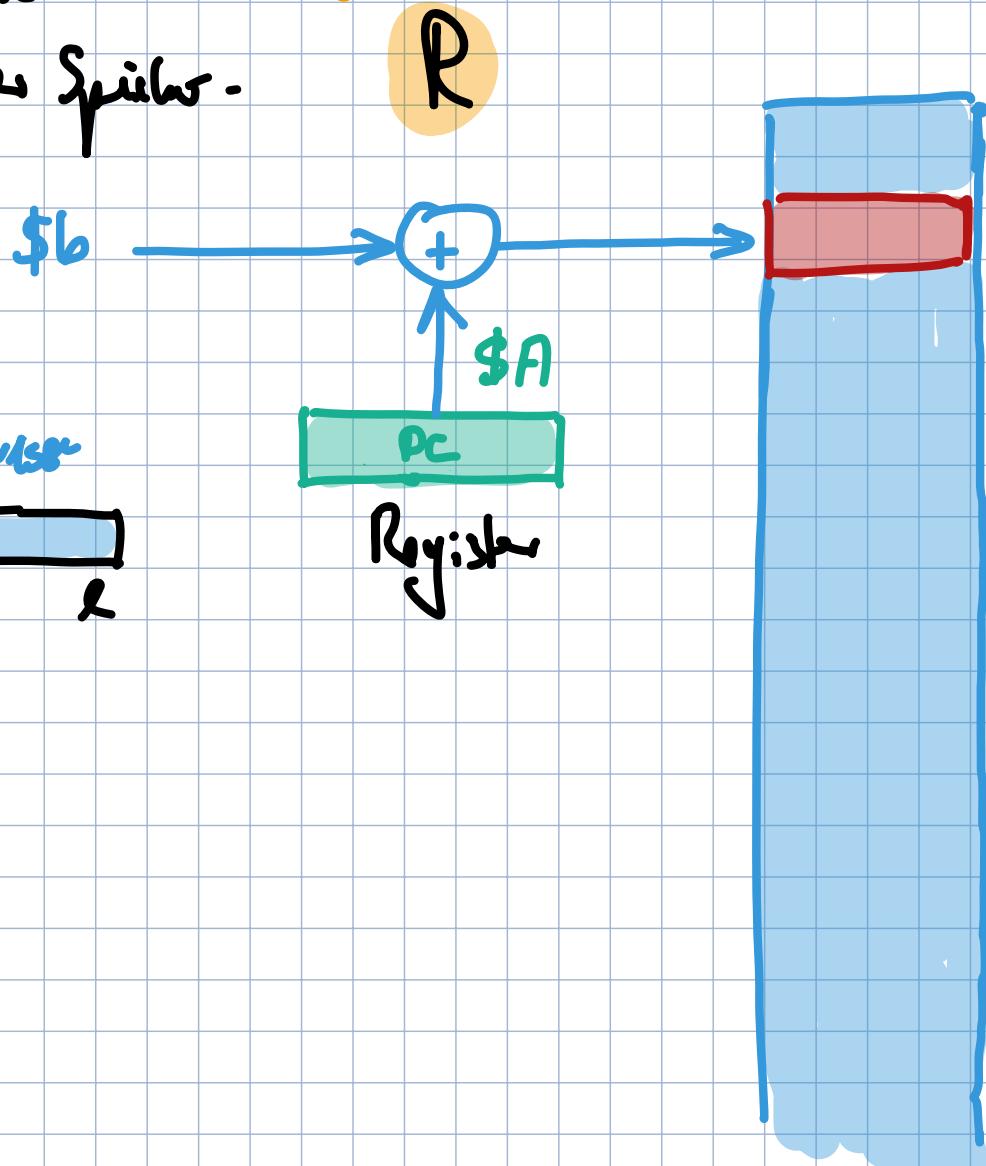
jmp \$b

Operat. O

$h = \text{Länge}$

Argument z

$h, l, n = \text{Länge}$



# Kombinierte Adressierungen

Die einzelnen Adressierungsmodi können auch kombiniert werden:

Beispiele:

add (\$a,\$x)

Indirekt-Indirekt

add (\$a),\$y

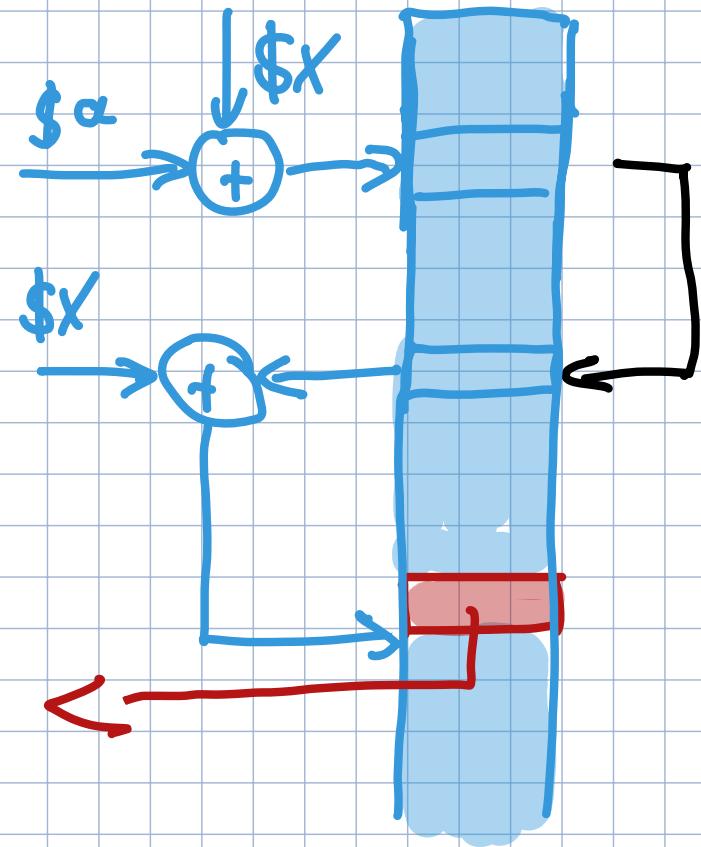
Indirekt-Indiziert

Indirekte-Adressing mit Post-konstant

Indirekte-Adressing mit Pre-konstant

Relative Adressing mit Post-konstant

Relative Adressing mit Pre-konstant



# Auswahl typischer Befehle

## Arithmetisch-Logische Befehle

addition

Subtraction

Logical and

Logical nor

Logical or

Shift left logical

Shift right logical

No operation

## Transport

load

store

move

push

pop

## Sprunge

j jump

jal jump and link

jr jump register

jsr jump to subroutine

rts return from subroutine

## Vergleichen

bne branch not equal

beq branch on equal

bfs branch on <flag> set

blt branch less than

bgt branch greater than

# Bispiel Befehlsatz

Bofill

add  
sub  
and  
hor  
or  
sll  
srl  
lop  
ld  
st  
uv  
ps  
pr  
bba  
bug  
bfs  
j  
jsv  
rts

# Addressing service

**Orthogonale Befehlssatz:**  
Fast jede Befehl unterscheidet  
fast jede Adresse von sonst.



Microprogrammierung

Idee: Sowohl wie möglich in Hardware,  
sowohl wie möglich in Software!

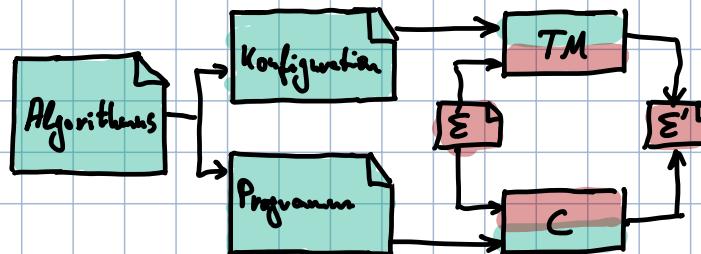
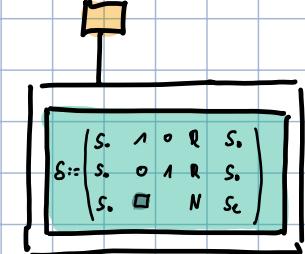
Wie bleibt das Steuern einfach?

Universelle Turing-Maschine

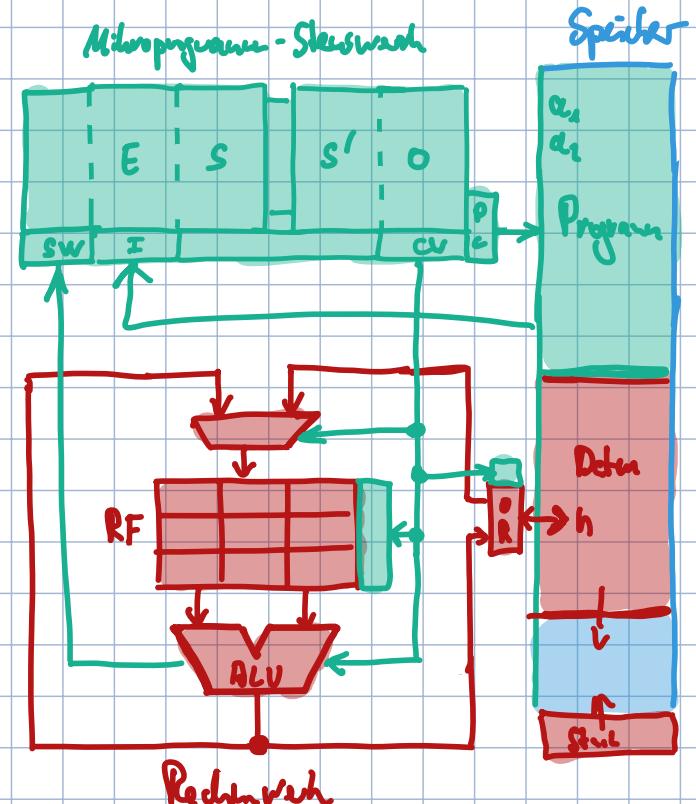


① Laden der Konfiguration von Band

② Konfiguration auf Eingabe anwenden und Pfeile erzeugen.

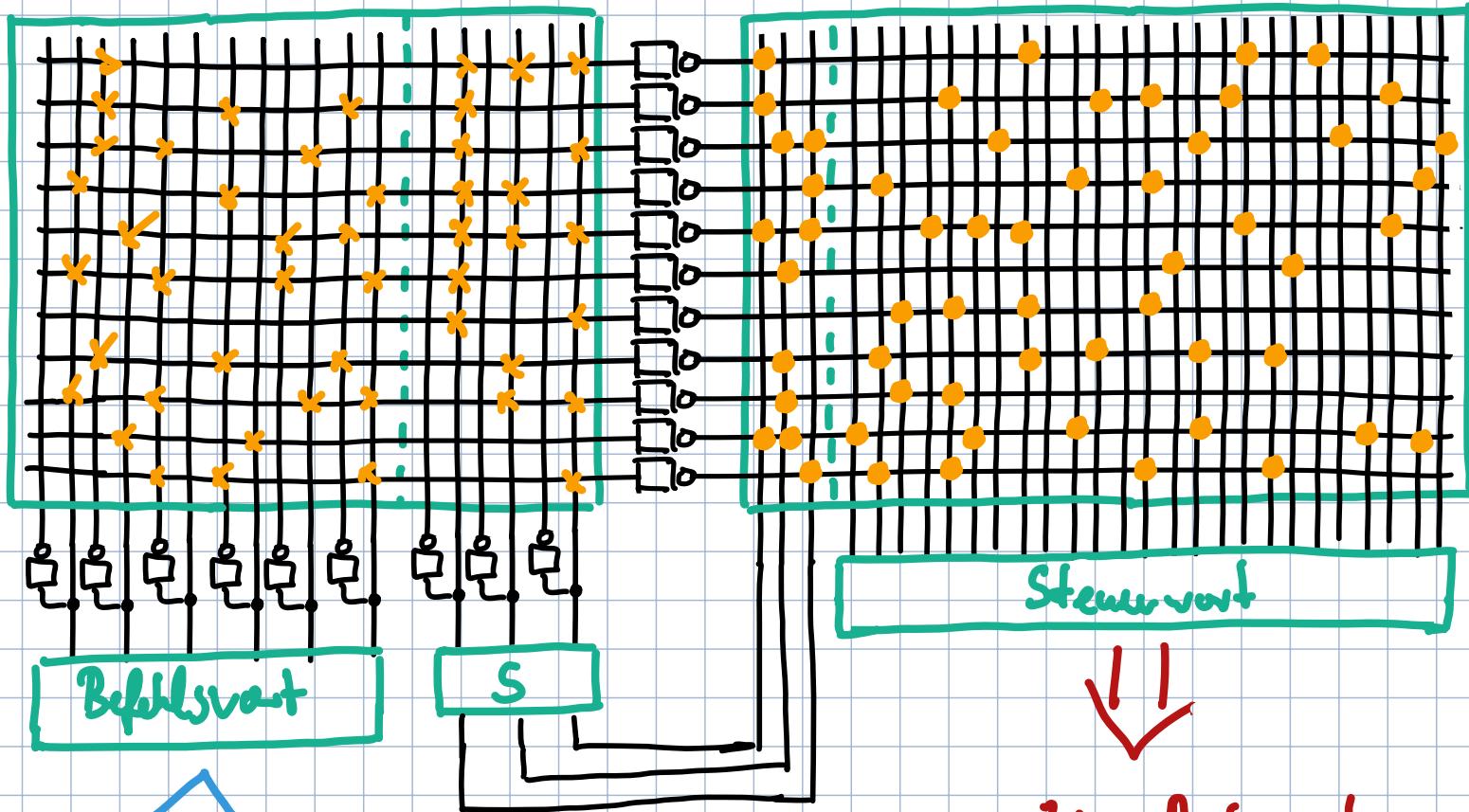


Universelle Registermaschine



Steuerwrt: Das Steuerwrt ist im Ausformat

## Programmable Logic Array



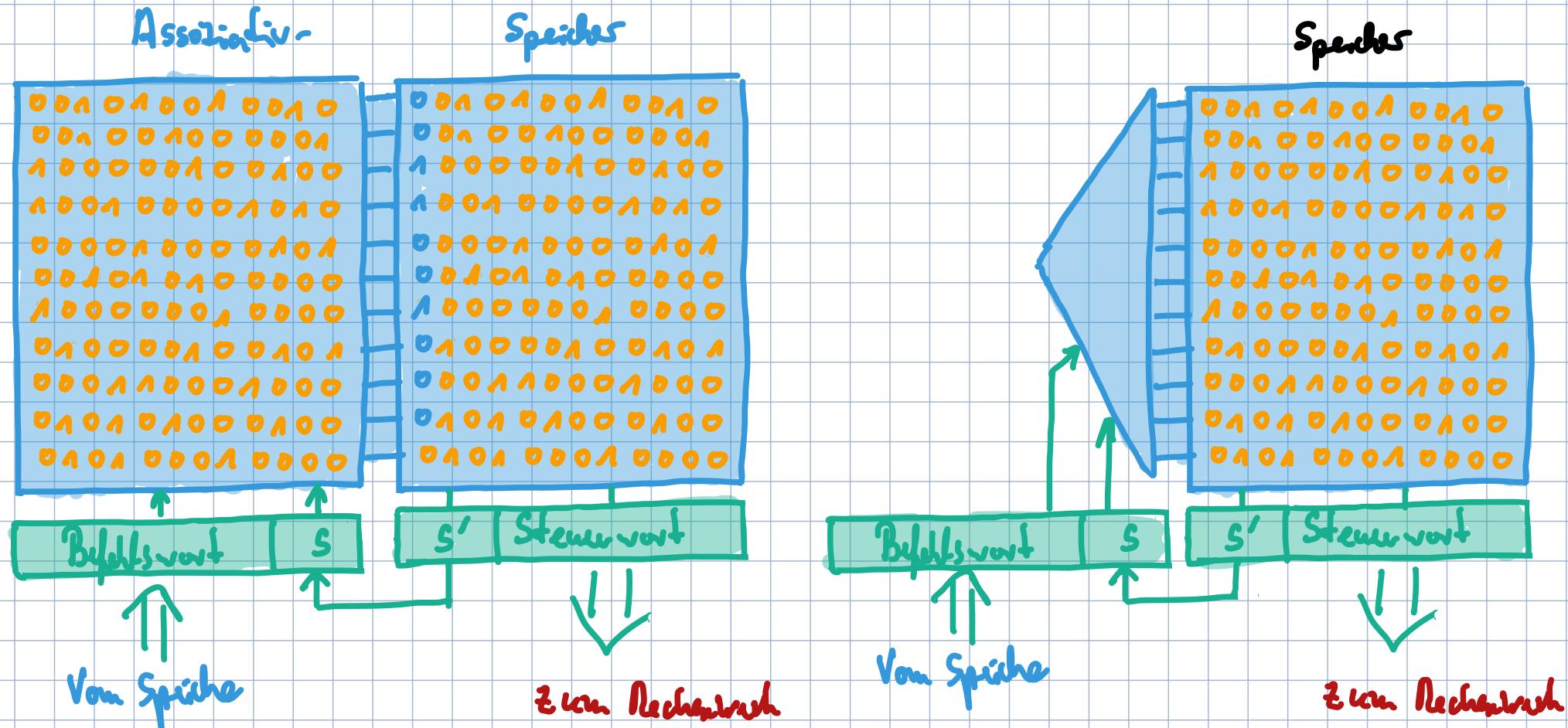
Aus dem Speicher

x } Programmierung

Programmierung durch Kontaktlöcher!  
in Hardware

# Steuerwelt im Sinne einer Konfiguration?

# Aufbau des PLA mit Hilfe von Spinles!



# Vertikale und horizontale Mikroprogrammierung

## Vertikales Mikroprogramm

IF Hole Befehl gold

001000100  
000000101  
010100000  
00100100

Adresse R

010010000  
001000101  
010010000  
101001001

ID Befehl dekodieren

01010010

OF Hole Operand aus S

010010010  
001100101  
010010000  
101001001

EX Operation ausführen

010010001  
010100100

OS Speichere Ergebnis  
in d

010010000  
001000101  
010010000  
101100101



Skewwort

## Horizontales Mikroprogramm

00100010000000001010100000100100

0100100000000000001010100000100100

010100100000000000000000000000000000000000

0100100100000000001010100000100100

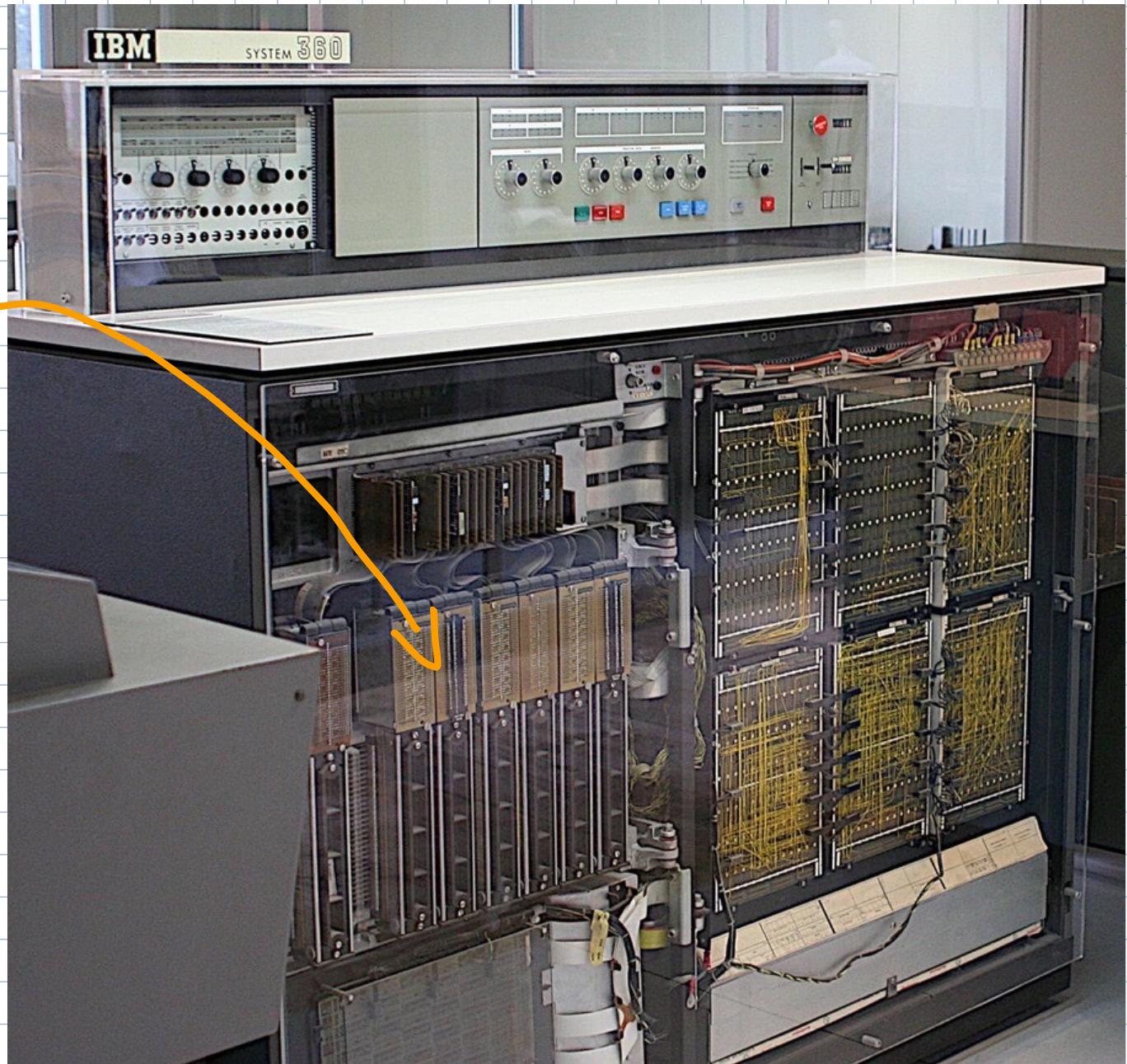
0100100010000000000000000000000000000000000000

0100100000000000001010100000100100

Die IBM 360 hatte horizontale  
Mikroprogrammierung

# Mikroprogrammierspeicher des IBM/360 Systems

Mikroprogrammierspeicher  
mit Transistoren  
(ähnlich Magnet-  
speicher)



# Microcode in Intel 8086

Microprocessor 1978

16-Bit (Hälfte Wort)

16 MB Addressraum

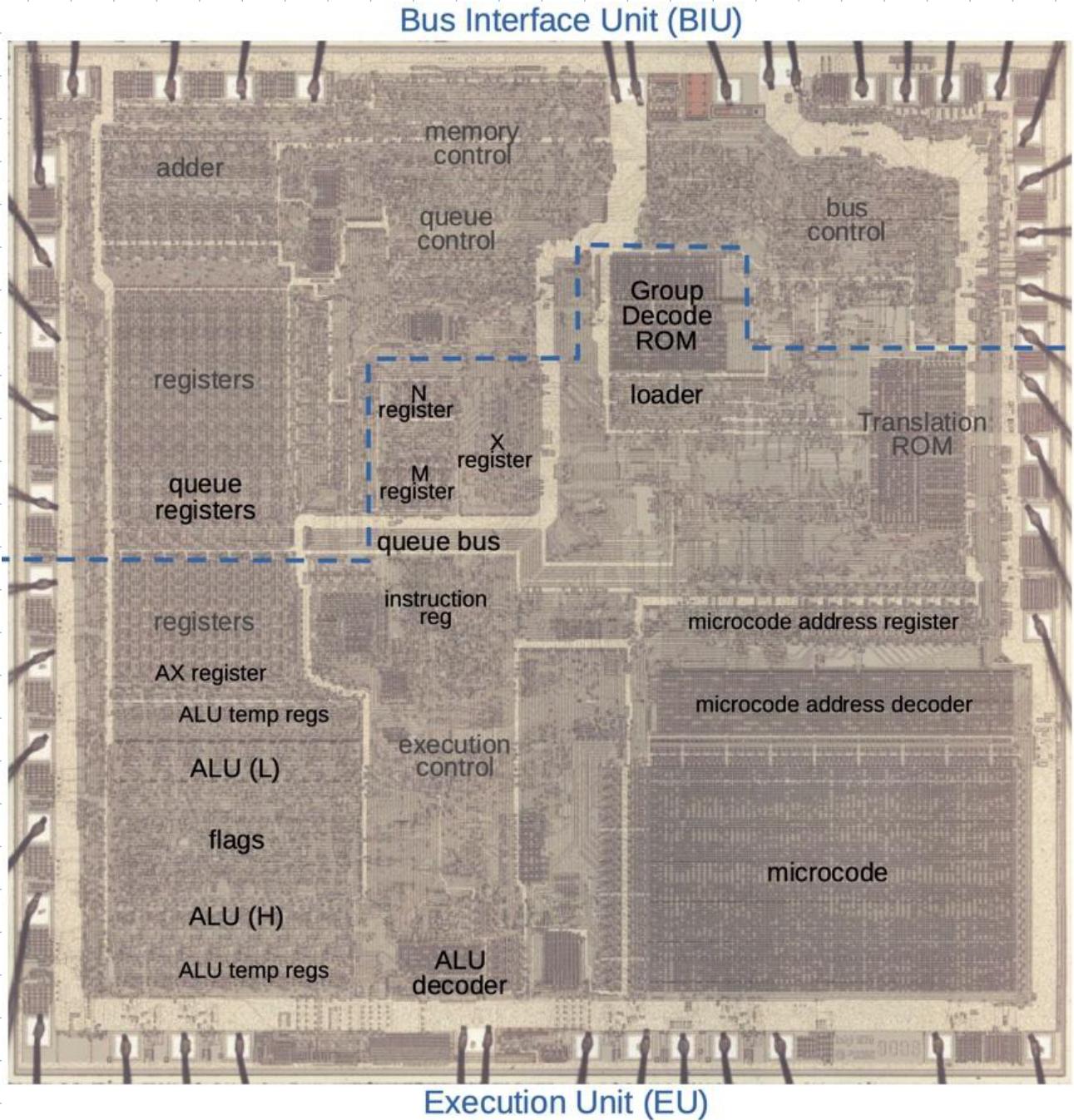
Akkumulatormaschine

Fertigungstechnologie:

3 µm NMOS

Processorfakt: 5 MHz - 10 MHz

# Die des 8086



# Microcode in Intel 8086

cycle	clock	Loader	Q bus	$\mu$ code addr reg	$\mu$ -addr	$\mu$ code out	action
1	H	First Clock	instruction (ADD imm)	instruction in prev instruction out	prev $\mu$ -addr	prev micro-instruction	ALU op to X register
	L						
2	H	Second Clock	low operand	instruction out step 0	018	prev micro-instruction	X register out
	L			step 1			
3	H			step 2	019	Q $\rightarrow$ tmpBL L8	
	L		high operand	step 3			
4	H			step 4	01a	Q $\rightarrow$ tmpBH	Q bus byte to ALU tmp B (low)
	L			step 5			
5	H	First Clock	next instruction	next instruction in current step 3 out	01b	M $\rightarrow$ tmpA, XI tmpA NXT	Q bus byte to ALU tmp B (high)
	L						
6	H	Second Clock	...	new instruction out step 0	next $\mu$ -addr	$\Sigma \rightarrow M$ , RNI, FLAGS	AX to ALU tmp A ALU op decoded
	L						ALU controls active result generated
7	H			...		next micro-instruction	AX updated flags updated