

# IT-Infrastruktur

WS 2014/15

**Hans-Georg Eßer**  
Dipl.-Math., Dipl.-Inform.

**Foliensatz D:**

- Rechnerstrukturen, Teil 1

v1.0, 2014/10/21

Vorlesungsübersicht

Seminar

Wiss. Arbeiten

Datenformate und Wandlung

PC als Arbeitsplatz

Ergonomie und Arbeitsschutz

Rechnerstrukturen

(Telekommunikation)

Infrastruktur-Technologie

Zentrale / verteilte IT-Infrastrukturen

Folien D

- Prinzipien und Methoden für Analyse, Implementierung, Bewertung und Klassifikation von Rechnerarchitekturen
- Architekturprinzipien und Merkmale moderner RISC- und CISC- (Mikro-) Prozessoren wie
  - Befehlssätze
  - Pipelining
  - Superskalarität
  - Cache-Organisation
- Organisationsprinzipien von Multiprozessor-Systemen und wichtige Architekturmodelle

## **1. Grundlagen**

- Begriffsbestimmung, Abgrenzung
- Historische Entwicklung
- Chipentwurf/Technologie

## **2. Instruction Set Architecture (ISA)**

- Registerstruktur, Adressierungsarten
- Maschinenbefehlssätze, Spezialbefehle
- Stack-Maschinen

## **3. Leistungsbewertung und Leistungsmessung**

## 4. Pipelining

- Klassische Fünf-Stufen-Pipeline
- Pipeline-Hemmnisse
- Superskalarität, *out of order execution*
- Spekulative Befehlsausführung, Sprungvorhersage

## 5. Speichersysteme

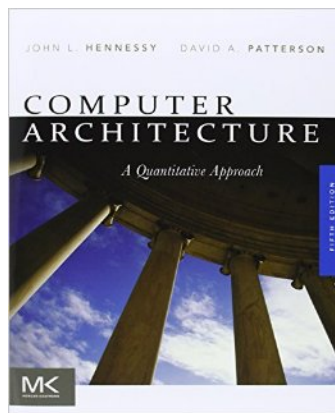
- Speichertypen, Caches

## 6. Sprungvorhersage

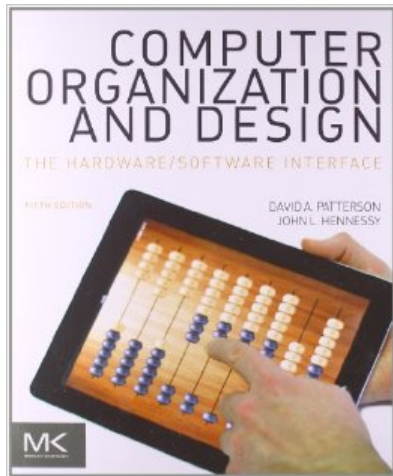
## 7. Mehrprozessorsysteme



**Rechneraufbau und Rechnerarchitektur**  
Axel Böttcher (Hochschule München)  
ISBN: 3540209794 (2007)



**Computer Architecture: A Quantitative Approach**  
5th ed.  
John L. Hennessy, David A. Patterson  
ISBN: 012383872X (2011)



## **Computer Organization and Design, The Hardware/Software Interface**

5th ed.

David A. Patterson, John L. Hennessy

ISBN: 0124077269 (2013)



## **Rechnerarchitektur: Von der digitalen Logik zum Parallelrechner**

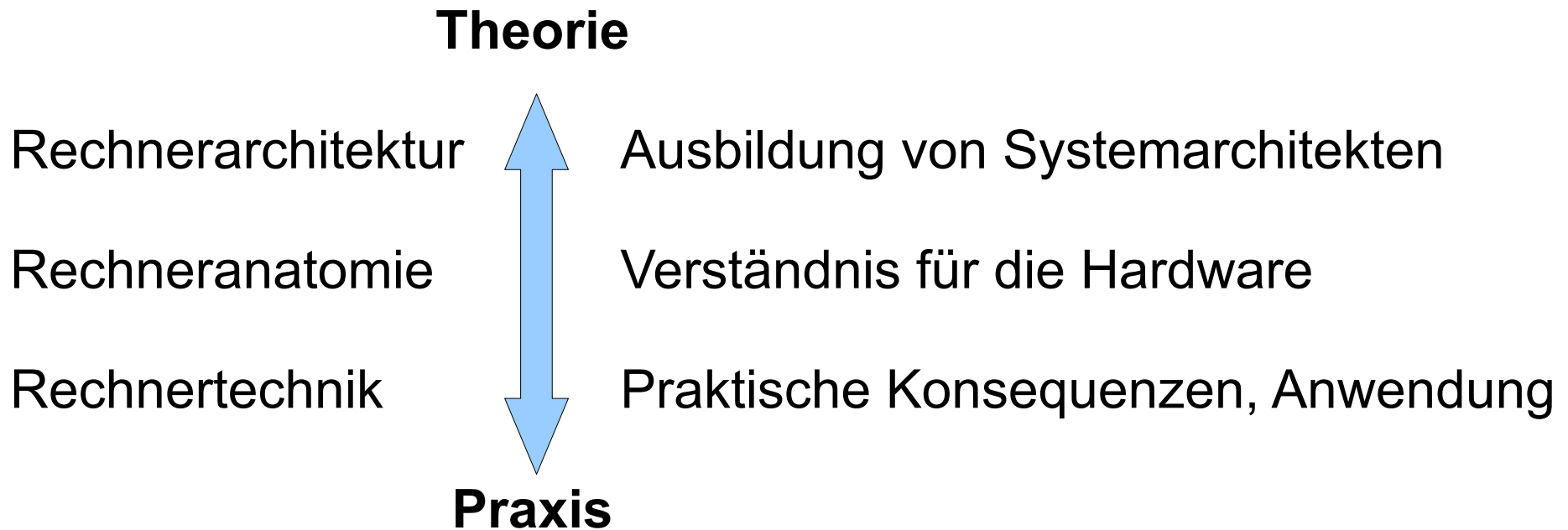
Andrew S. Tanenbaum

ISBN: 3868942386 (2014)

- Einen Großteil der Folien habe ich von  
**Prof. Dr. Axel Böttcher**  
(FH München) übernommen, der diese  
Vorlesung 2009 gehalten hat und mir sein  
Material freundlicherweise zur Verfügung  
gestellt hat.



# 1. Grundlagen



## Rechnerarchitektur umfasst

- die Analyse
  - die Bewertung
  - den Entwurf
  - die Synthese
- von Rechnern und Rechnerkomponenten.

Dazu:

- strukturelle
- organisatorische
- implementierungstechnische

Aspekte berücksichtigen und auf der

- globalen Systemebene
- Maschinenbefehlssatzebene
- Mikroarchitekturebene

untersuchen

Zwischen den beteiligten Ebenen und den verschiedenen Teilaspekten der Rechnerarchitektur sind Rückkopplungen möglich.

Auf allen Ebenen umfangreiche Wechselwirkungen mit anderen Disziplinen der

- Informatik,
- Ingenieur- und Naturwissenschaften
- Mathematik

## Verschiedene Betrachtungsebenen

Globale Systemebene (Prozessoren, Busse, Speicher)

- *Für:* System-Architekt für Chip/Motherboard, Vertrieb
- *Sicht auf:* ganzes System
- *Elemente:* Welche Hauptelemente besitzt das System, und wie sind diese miteinander verbunden.

Maschinenbefehlssatzebene

- *Für:* Compilerbauer, Assembler-Programmierer, Vertrieb
- *Sicht auf:* Prozessor-Funktionen
- *Elemente:* Satz von Befehlen (Maschineninstruktionen), den der Prozessor beherrscht.

## Verschiedene Betrachtungsebenen

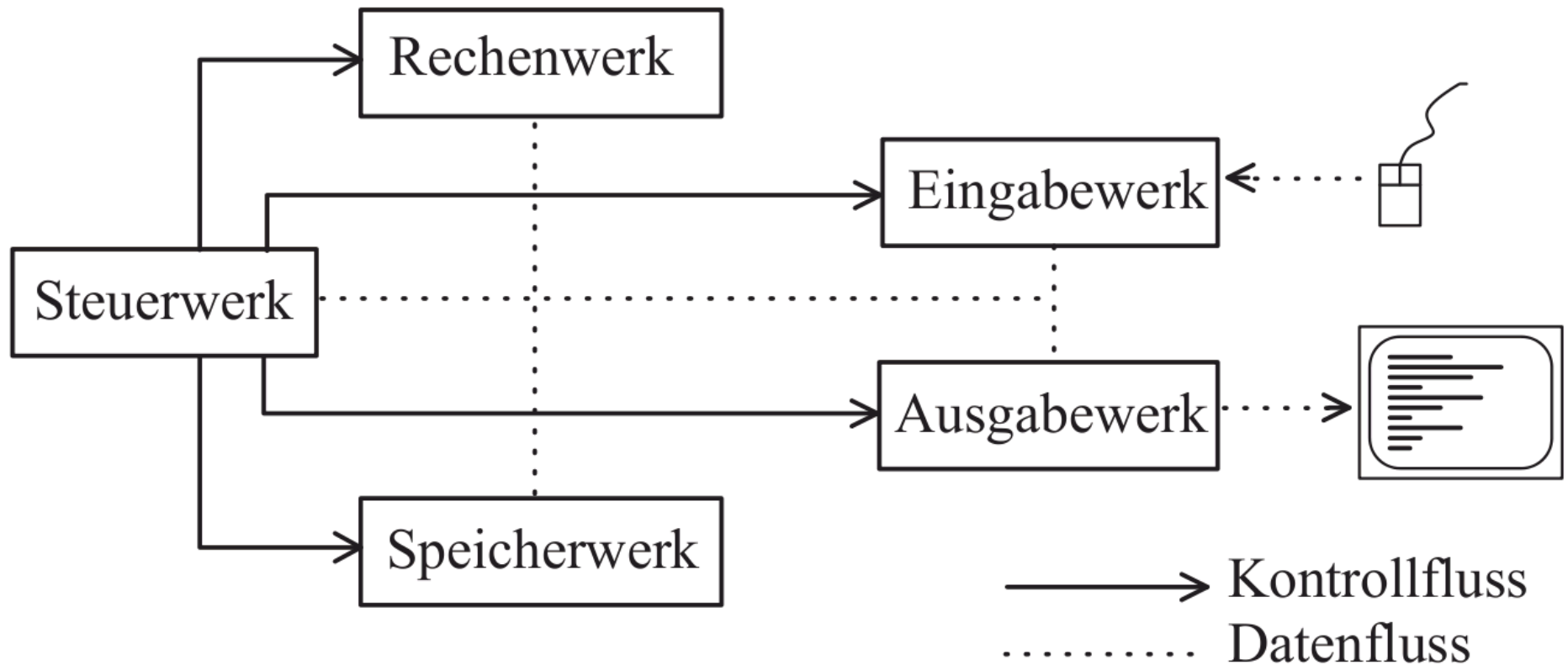
### Register-Transfer-Ebene (Mikroprogramm-Ebene)

- *Für:* CPU-Entwickler, Programmierer, Mikroprogrammierer (Informatik-Grundwissen)
- *Sicht auf:* Infrastruktur auf dem Chip
- *Elemente:* Register; wie diese mit ihrer Umgebung verbunden sind und welche Verknüpfungen vorgesehen sind, um den Befehlssatz zu realisieren.

## Teile der Mikroarchitekturebene

- Logikebene
  - *Für:* Chip-Designer
  - *Sicht auf:* Schaltung
  - *Elemente:* Gatter und Flipflops und wie diese verbunden sind.
- Hardware-Realisierungs-Ebene
  - *Für:* Designer
  - *Sicht auf:* Chiplayout
  - *Elemente:* Bauteile (z.B. Transistoren, Kondensatoren etc.) und wie diese verbunden sind, um Gatter oder Flipflops zu realisieren.

- Bestandteile eines Universalrechners (Computer) nach John v. Neumann



- 5 Funktionseinheiten: Steuerwerk, Rechenwerk, Speicherwerk, Eingabewerk und Ausgabewerk
- Aufbau unabhängig vom zu bearbeitenden Problem
- Programme und Daten, Zwischenergebnisse und Endergebnisse im gleichen Speicher
- Eingabewerk liest Daten und Programm-Code ein und legt sie im Speicherwerk ab.
- Speicher in gleich große Zellen aufgeteilt (fortlaufend nummeriert)
- Aufeinander folgende Befehle befinden sich in aufeinander folgenden Speicherzellen.
- Im Steuerwerk gibt es einen **Befehlszähler**, der immer auf den nächsten auszuführenden Befehl zeigt.

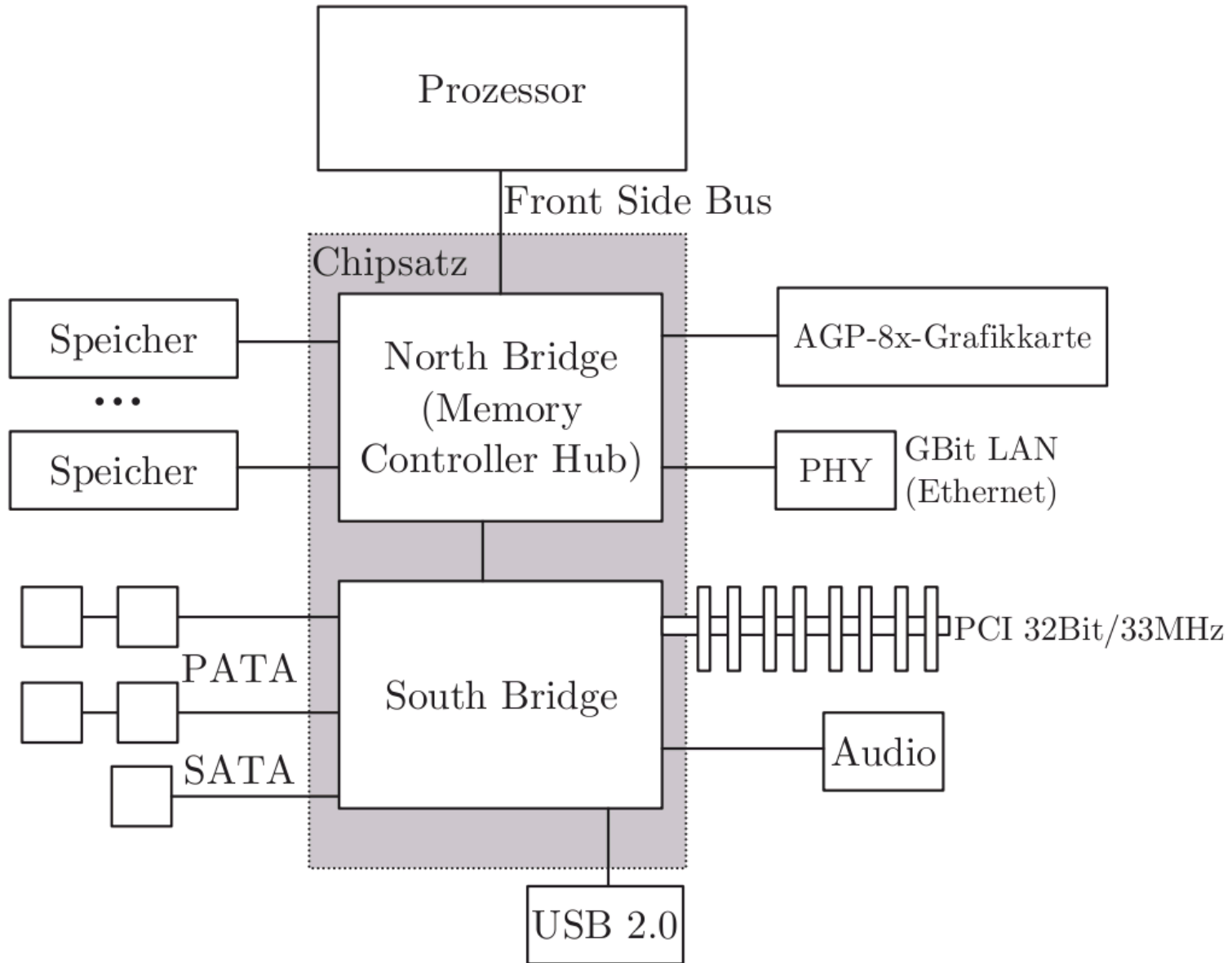


## Befehlsausführung

- Der aktuelle Befehl wird aus dem Speicher ausgelesen und im **Befehlsregister** des Steuerwerks zwischengespeichert.
- Der Befehl wird dann dekodiert, und die Ausführung des Befehls durch Steuersignale veranlasst.

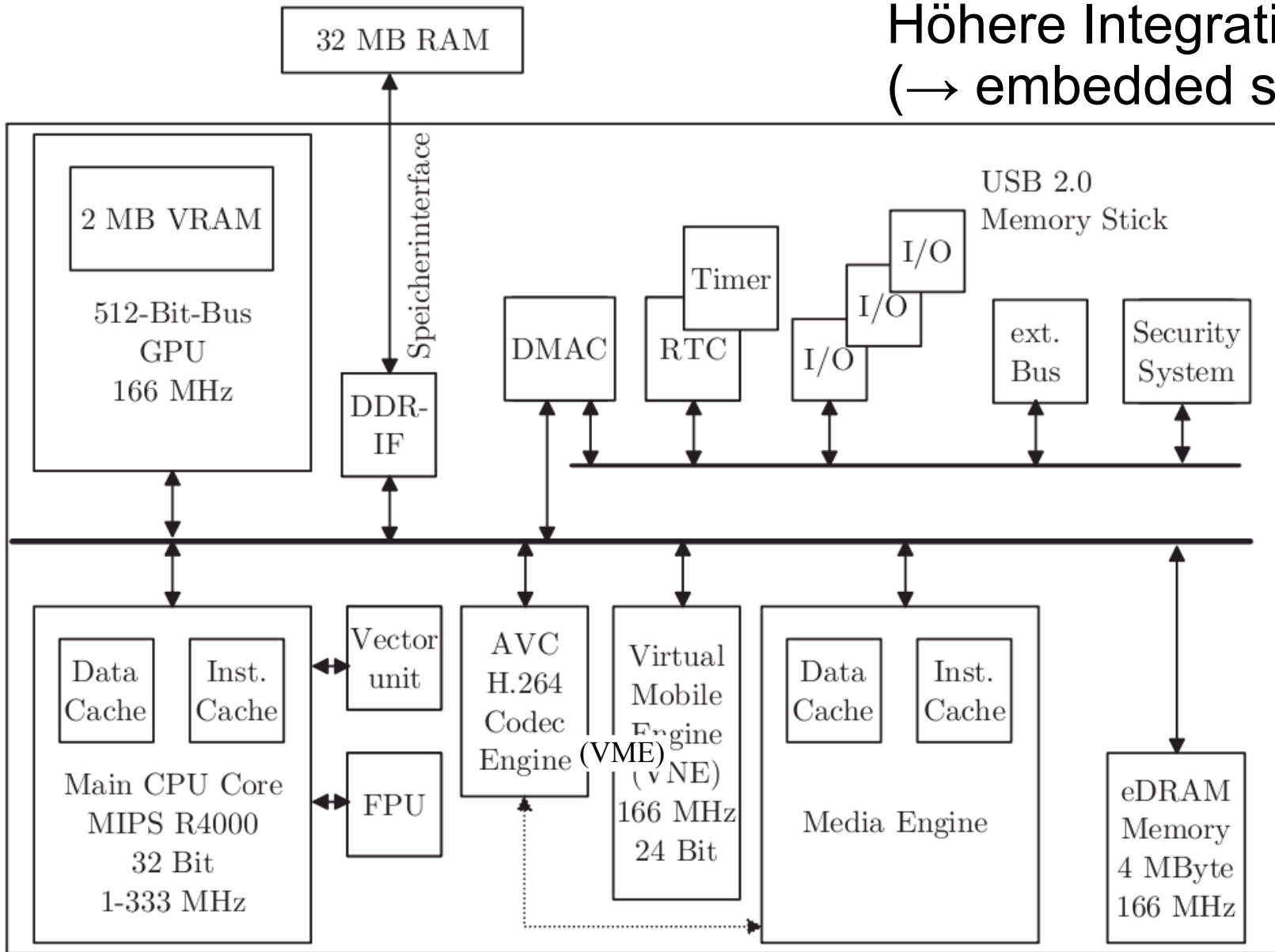
- Es gibt folgende Befehlsarten:
  - Arithmetische und logische Befehle zur Verknüpfung von Daten
  - Transportbefehle zum Verschieben von Daten zwischen diesen Komponenten
  - Bedingte und unbedingte Sprungbefehle, Unterprogrammaufrufe
  - Ein-/Ausgabebefehle zur Kommunikation mit der Peripherie
  - Sonstige Befehle wie Unterbrechen, Warten, Stop etc.

# Beispiel: Aufbau eines PCs



# Beispiel: Aufbau der PSP

Höhere Integrationsdichte  
(→ embedded systems)



Chip		Anzahl Transistoren
Flipflop	Speichert ein Bit	6
Gatter (und/oder)	Verknüpft zwei binäre Werte	4
Addierer	Addiert 64-Bit-breite Worte	> 400
Datenpfad	Komplettes Rechenwerk mit Puffern	> 200.000
Pentium II	Ganzer Prozessor	4,5 Millionen
Pentium 4	Ganzer Prozessor	42 Millionen
Xeon-Dunnington	6-Kern-Prozessor	1,9 Milliarden

# Landschaft der Prozessoren (Wichtige Begriffe)

---

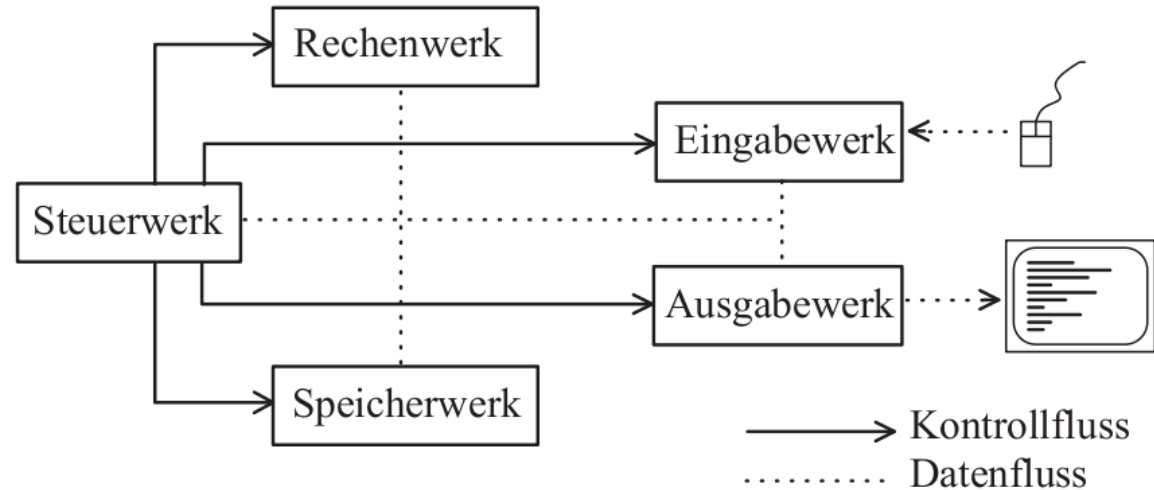
- **Hardwarebeschreibungssprachen:**  
Programmiersprachen zur Beschreibung von Hardware. Können verwendet werden, um Hardware zu generieren. Wichtigste Vertreter: VHDL (Very High Speed Integrated Circuit Hardware Description Language) und Verilog. (siehe HP-Forschungsprogramm PICO: „Program In – Chip Out“)
- **Synthetisierbarer Kern:**  
Prozessorkern, der in einer Hardwarebeschreibungssprache vorliegt (**Softcore**) und von Lizenznehmern in eigene Entwürfe eingebunden werden kann; z. B.: MIPS32 24K, ARM1136J, Intel XScale 80200T.

# Landschaft der Prozessoren (Wichtige Begriffe)

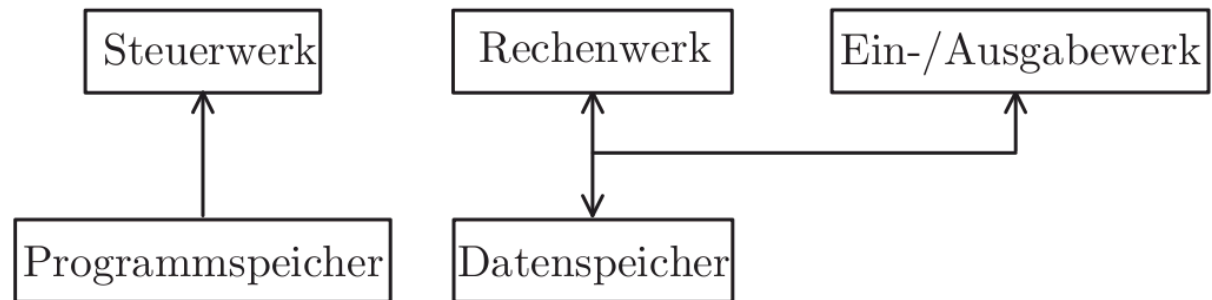
---

- **System on a Chip (SoC):**  
enthält auf einem einzigen Chip neben dem Prozessorkern auch Speicher, Schnittstellen, Timer und ggf. auch Grafik-, DMA- und Interrupt-Controller sowie PCMCIA, Touch-Panel-Interface etc. (z.B. PowerPC 405LP). Daher kein Chipsatz erforderlich
- **Embedded Systeme** (auch: Prozessrechnersysteme):  
kleine Computer, die in bestimmten Produkten eingesetzt werden (also **eingebettet** sind). Sie übernehmen dort Steuerungs-, Kontroll- oder Bedienungsaufgaben.  
Einsatzgebiete: Haushaltsgeräte, Unterhaltungselektronik, Fahrzeuge, ...

- Von-Neumann-Architektur



- Harvard-Architektur

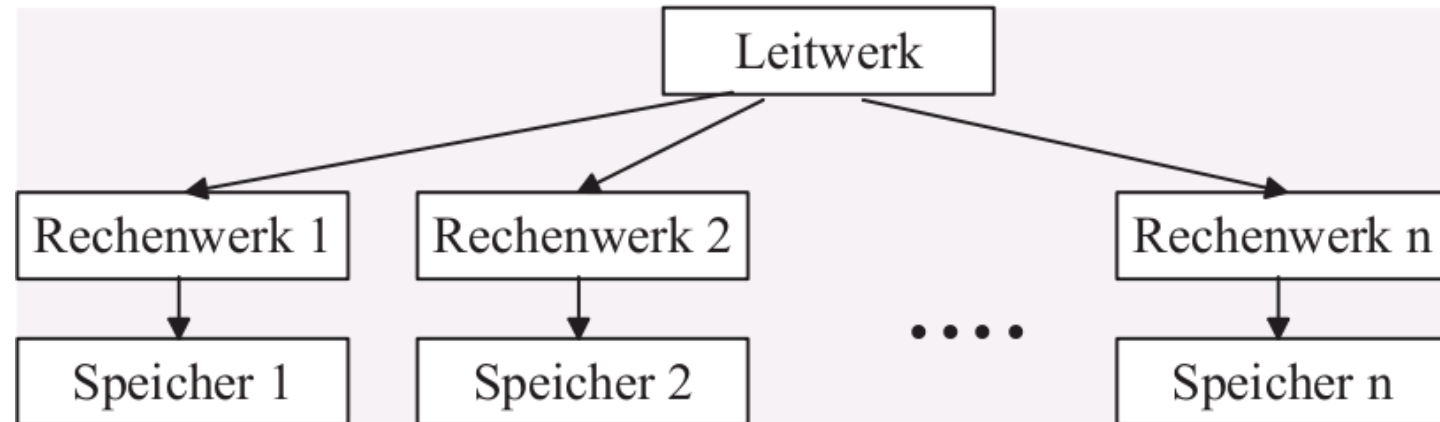
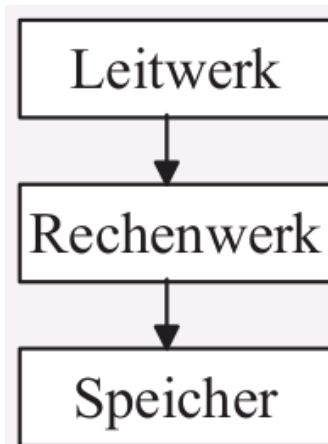


Getrennte Speicher für Programm und Daten. Anzutreffen bei manchen Signalprozessoren. Als modifizierte Harvard-Architektur in praktisch allen modernen Prozessoren mit getrennten Level-1-Caches für Code und Daten verwendet



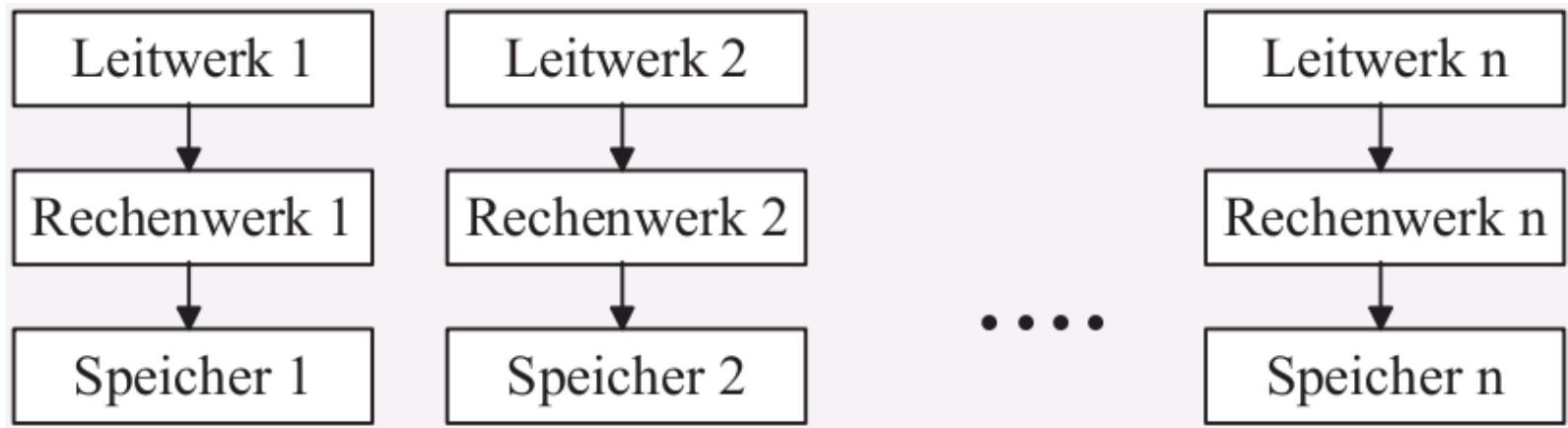
# Klassifikation nach Flynn: SISD, SIMD

- Flynn (1966). Heute nicht mehr ganz tragfähig, aber die Begriffe werden noch verwendet.
- Single Instruction, Single Data (**SISD**):
- Single Instruction, Multiple Data (**SIMD**):

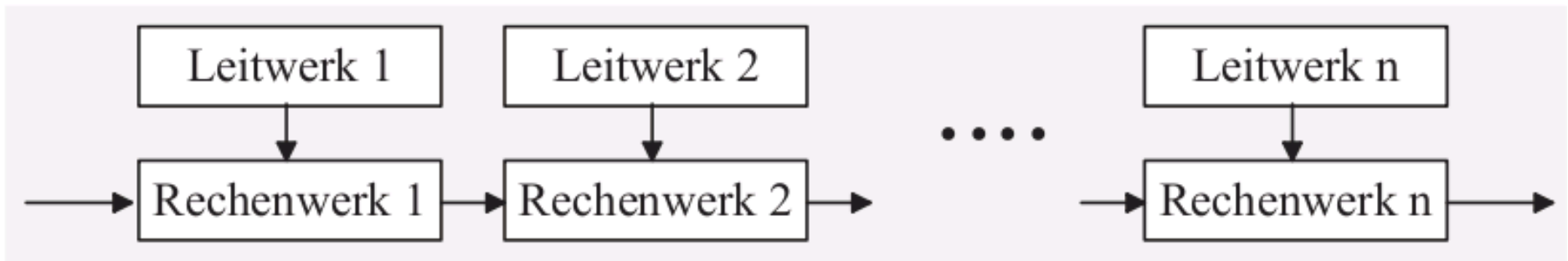


# Klassifikation nach Flynn: MIMD, MISD

- Multiple Instruction, Multiple Data (**MIMD**):



- Multiple Instruction, Single Data (**MISD**):



## **2. ISA: Instruction Set Architecture (Befehlssatzarchitektur)**

# Befehlssatzarchitektur (Instruction Set Architecture, ISA)

---

Beschreibung umfasst:

- Maschinenbefehlssatz
- Registerstruktur
- Adressierungsarten
- Interruptbehandlung

Klassisch: Unterscheidung in  
Ein-, Zwei- und Drei-*Adressmaschinen*

Heute üblicher: unterscheiden nach  
Ein-, Zwei- und Drei-*Adressbefehlen*

- Register: die schnellsten speichernden Elemente eines Prozessors
- meist allgemein verwendbare Register (General Purpose Registers, GPR) und Spezialregister.
- Gesamtheit aus Befehlssatz und verfügbaren Registern heißt **Programmiermodell**

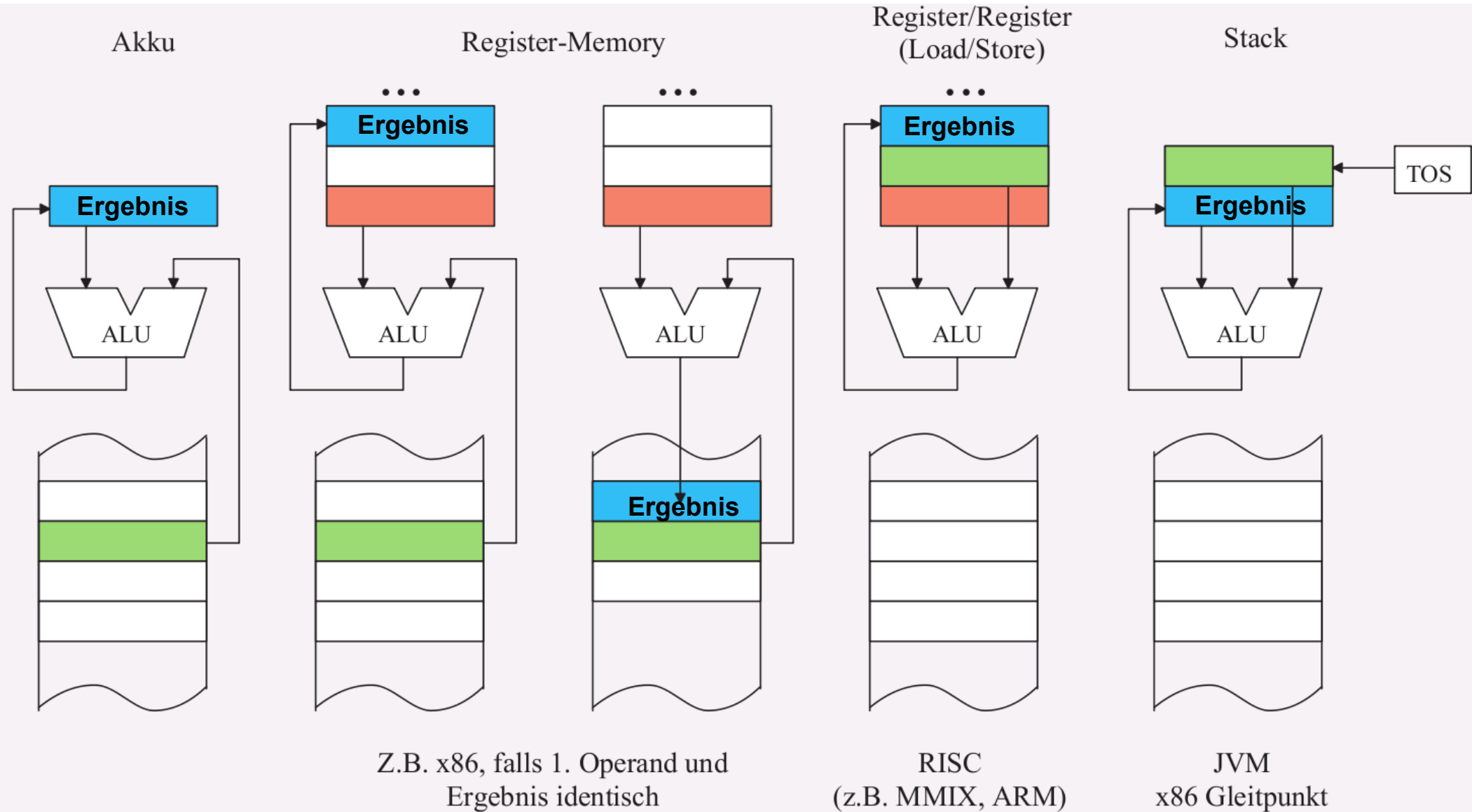
## Typische Spezialregister

- Befehlszähler
- Stackpointer
- Statusregister (kann z. B. anzeigen, ob bei der letzten Operation ein Überlauf aufgetreten ist, oder ob das Ergebnis negativ war etc.)
- Indexregister (für Adressrechnungen)

- ISAs unterscheiden nach Zugriff auf Register und Speicherinhalte
- Aufgabe: Werte aus zwei Speicherzellen addieren und in dritter Zelle speichern

$C := A + B;$

# Operanden und Ergebnis (2/6)





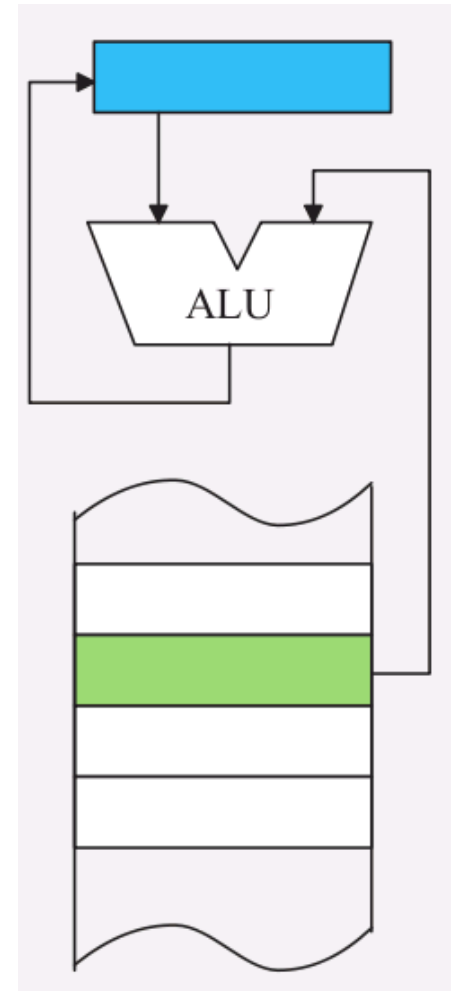
$C := A + B$

mit **Akku**:

```
LOAD  A
ADD   B
STORE C
```

nutzt implizit den Akku:

- LOAD A = lade A in Akku
- ADD B = addiere B zum Wert in Akku (Ergebnis im Akku)
- STORE C = schreibe Akku-Inhalt nach C

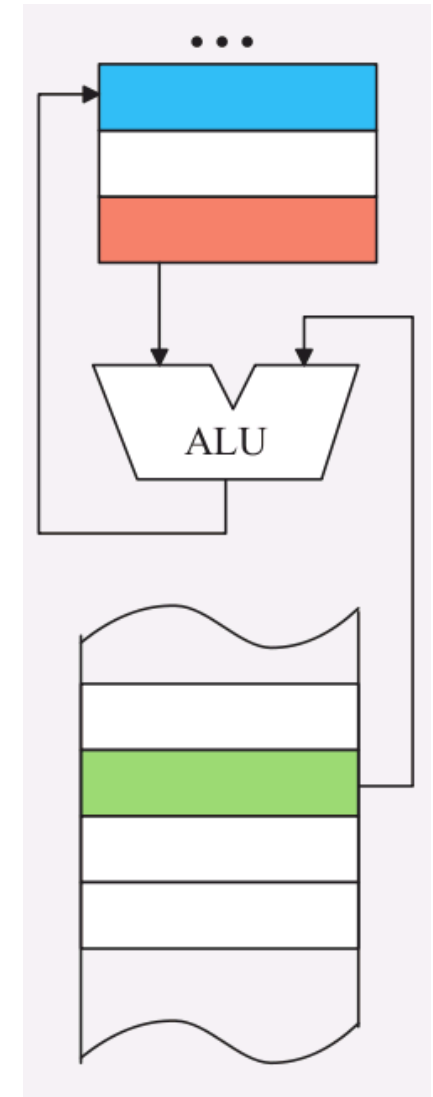


$C := A + B$

mit **Register-Memory**:

```
LOAD  R1, A
ADD   R3, R1, B
STORE R3, C
```

nennt alle Register explizit (R1, R3)  
Argumente können Register oder  
Speicherstellen sein (z. B. R1, B)

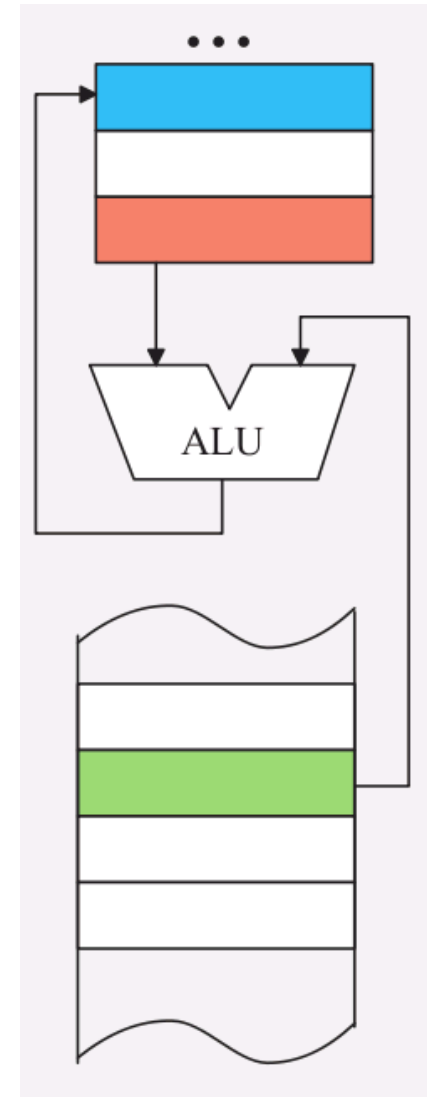


$C := A + B$

mit **Register-Register (Load/Store)**:

```
LOAD  R1, A
LOAD  R2, B
ADD   R3, R1, R2
STORE R3, C
```

- nennt alle Register explizit (R1, R3)
- Argumente für Operationen können nur Register sein (R1, R2)
- Zugriff auf Speicher nur durch LOAD, STORE



$C := A + B$

mit **Stack**:

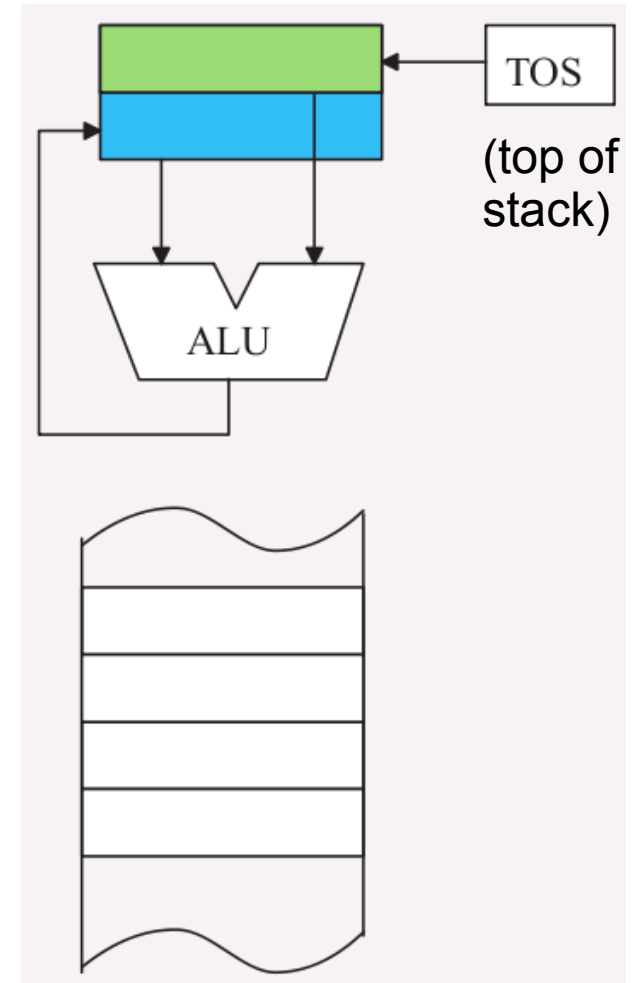
PUSH A

PUSH B

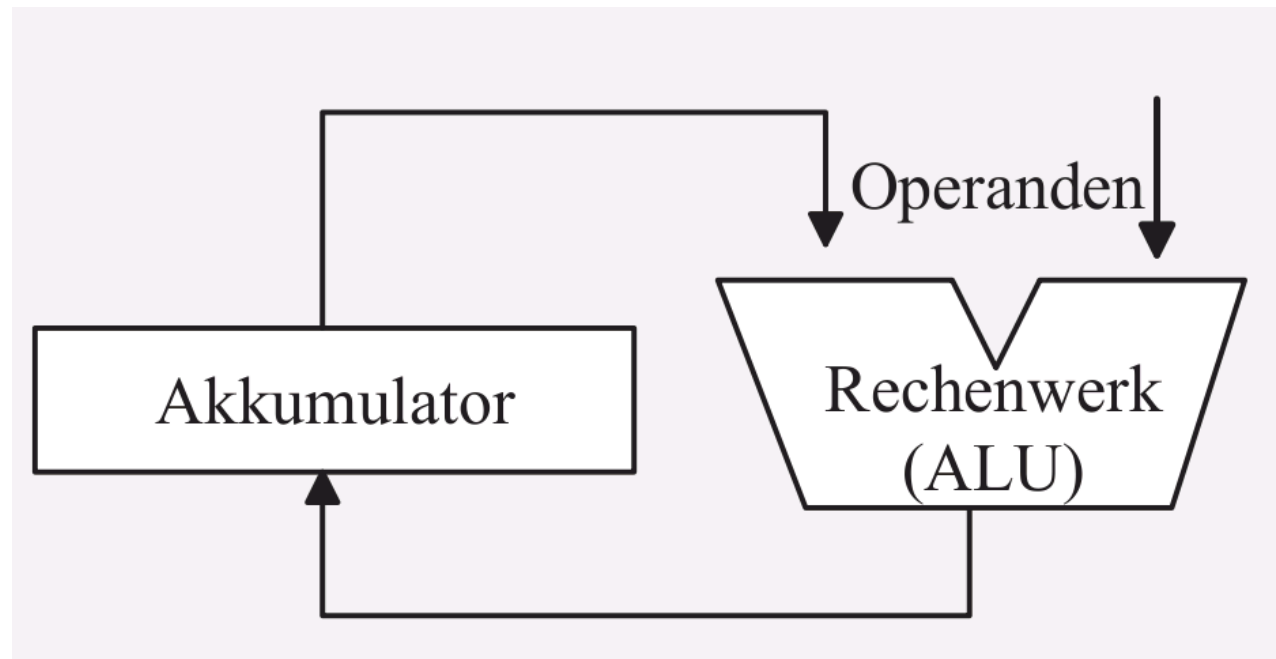
ADD

POP C

- keine Register
- Argumente auf Stack, dann Operation
- Operation implizit (oberste Werte auf Stack)
- Zugriff auf Speicher nur durch PUSH, POP



- Maschinen, deren Befehle nur einen Operanden haben, heißen **Ein-Adress-Maschinen**.
- Akku: spezielles Register, impliziter linker Operand und Zielregister für das Ergebnis („Akkumulatormaschinen“)
- rechter Operand aus Speicher



- **Vorteile**

- Ausführung der einzelnen Befehle wegen der einfachen Hardware sehr schnell
- geringer Speicherbedarf für einen Befehl
- Fast jeder Befehl hat einen Operanden (außer z. B. NOP, INC, DEC) → einheitliche Befehlslänge, einfaches Aktualisieren des Program Counters

- **Nachteile**

- Programmierung in diesem Format erfordert Übung – vor allem das Auswerten von mathematischen Formeln
- Programme wegen des häufig erforderlichen Zwischenspeicherns von Hilfsgrößen etwas „länglich“

- Berechnung der Formel

$$y = \frac{x_1 + x_2 x_3}{x_1 - x_2}$$

mit MMIX (als Ein-Adress-Maschine; \$1=Akku)

01	x1	OCTA	3
02	x2	OCTA	7
03	x3	OCTA	11
04	Accu	IS	\$1
05	Main	LDO	Accu, x2
06		LDO	\$2, x3
07		MUL	Accu, Accu, \$2
08		LDO	\$2, x1
09		ADD	Accu, Accu, \$2

10	STO	Accu, x3
11	LDO	Accu, x1
12	LDO	\$2, x2
13	SUB	Accu, Accu, \$2
14	STO	Accu, x1
15	LDO	Accu, x3
16	LDO	\$2, x1
17	DIV	Accu, Accu, \$2
18	TRAP	0, Halt, 0

- Gleiches Programm in „echter“ Akku-Syntax:

01	x1	OCTA	3	
02	x2	OCTA	7	
03	x3	OCTA	11	
04		LOAD	x2	Akku enthält x2
05		MUL	x3	Akku enthält $x2 * x3$
06		ADD	x1	Akku enthält $x2 * x3 + x1$
07		STOR	x3	$x3 \leftarrow x2 * x3 + x1$
08		LOAD	x1	Akku enthält x1
09		SUB	x2	Akku enthält $x1 - x2$
10		STOR	x1	$x1 \leftarrow x1 - x2$
11		LOAD	x3	Akku enthält orig. $x2 * x3 + x1$
12		DIV	x1	Akku enthält Ergebnis
13		TRAP	Halt...	



- Befehle mit zwei Operanden (Register oder Speicheradressen)

- linker Operand ist implizit Ziel:

CMD  $x1, x2$  bedeutet  $x1 \leftarrow x1 \otimes x2$

- z. B. Addition

ADD  $\$1, \$2$  bedeutet  $\$1 \leftarrow \$1 + \$2$

- in MMIX-Syntax:

ADD  $\$1, \$1, \$2$

- Wieder mit MMIX-Befehlen

01	x1	OCTA	3	
02	x2	OCTA	7	
03	x3	OCTA	11	
04	Main	LDO	\$1, x1	Startwerte in \$1,
05		LDO	\$2, x2	\$2,
06		LDO	\$3, x3	\$3.
07		MUL	\$3, \$3, \$2	Produkt $x2 * x3$ in \$3
08		ADD	\$3, \$3, \$1	$x1 + x2 * x3$ in \$3
09		SUB	\$1, \$1, \$2	$x1 - x2$ in \$1
10		DIV	\$3, \$3, \$1	Bruch in \$3, fertig

- und mit „echter“ Zwei-Adress-Syntax

```

01  x1    OCTA  3
02  x2    OCTA  7
03  x3    OCTA 11
04  Main  LDO   $1, x1
05      LDO   $2, x2
06      LDO   $3, x3
07      MUL   $3, $2
08      ADD   $3, $1
09      SUB   $1, $2
10      DIV   $3, $1

```

```

01  x1    OCTA  3
02  x2    OCTA  7
03  x3    OCTA 11
04  Main  LDO   $1, x1
05      LDO   $2, x2
06      LDO   $3, x3
07      MUL   $3, $3, $2
08      ADD   $3, $3, $1
09      SUB   $1, $1, $2
10      DIV   $3, $3, $1

```

- Format von MMIX-Befehlen
- ein Ziel (erster Operand)
- zwei Quellen (2. und 3. Operand)

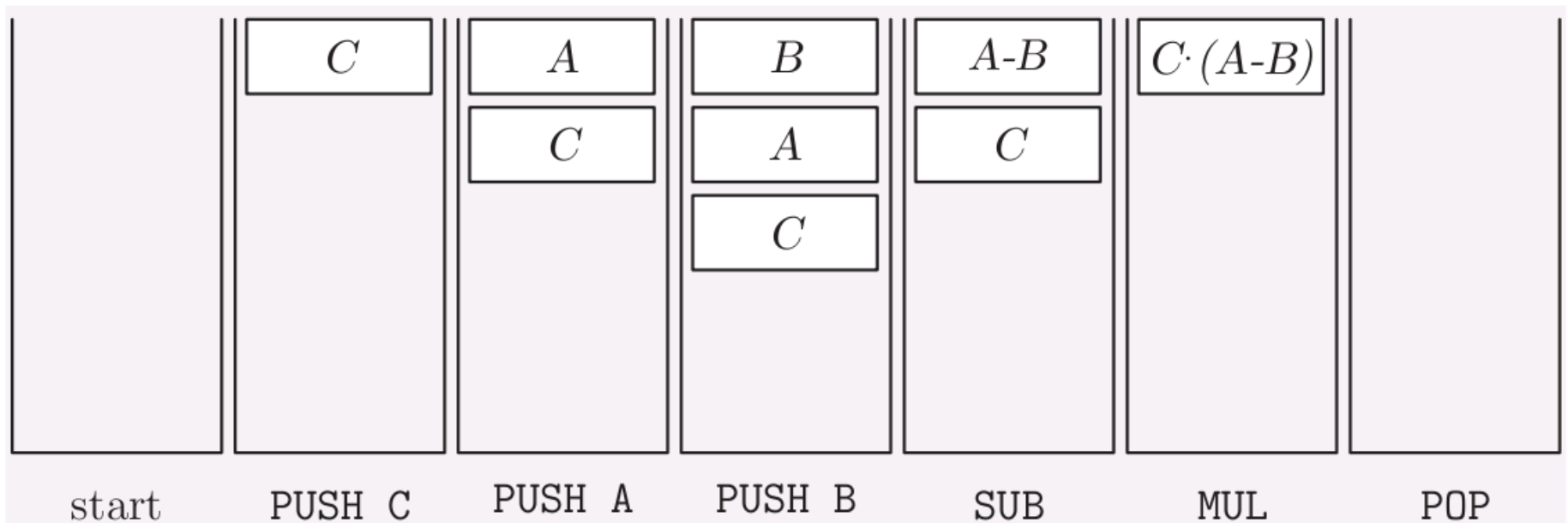
## Beispiel

01	x1	OCTA	3
02	x2	OCTA	7
03	x3	OCTA	11
04	Main	LDO	\$1, x1
05		LDO	\$2, x2
06		LDO	\$3, x3
07		<b>SUB</b>	<b>\$6, \$1, \$2</b>
08		<b>MUL</b>	<b>\$7, \$2, \$3</b>
09		<b>ADD</b>	<b>\$7, \$7, \$1</b>
10		<b>DIV</b>	<b>\$7, \$7, \$6</b>

- Vorteile
  - bequeme Programmierung
  - kurze Programme (Anzahl der Befehle)
- Nachteil
  - Eine Speicheradresse ist 64 Bit lang
  - drei Operanden (zunächst Register oder Speicheradresse): enorm große Befehlsbreite  
→ darum keine Speicheradressen als Operanden

# Null-Adress- (Stack-) Maschine

- Operanden der ALU immer oben auf Stack
- keine normalen Register
- Anwendung: Java Byte Code
- Beispiel: Berechne  $(a-b)*c$



- flexibel
  - Befehlslänge hängt vom Befehl ab
  - Auslesen komplizierter (erstes Byte entscheidet über Länge), keine Ausrichtung an Wortgrenzen
  - CISC
- fest
  - Alle Befehle gleich lang
  - Leichtes Auslesen
  - Einschränkung: Operanden nur in Registern (nicht genug Platz für Adressangaben)
  - RISC

$c := a+b; d := a-b; e := c*d$  – gesucht: nur  $e = (a+b)(a-b)$

Zwei Varianten

- Register-Memory

ADD c , a , b	SUB d , a , b	MUL e , c , d	
1 4 4 4	1 4 4 4	1 4 4 4	

**S=39**

- Register-Register (Load/Store)

LD R1 , a	LD R2 , b	ADD R3 , R1 , R2	SUB R4 , R1 , R2
1 4	1 4	2	2

MUL R5 , R3 , R4	STO R5 , e
2	1 4

**S=21**