



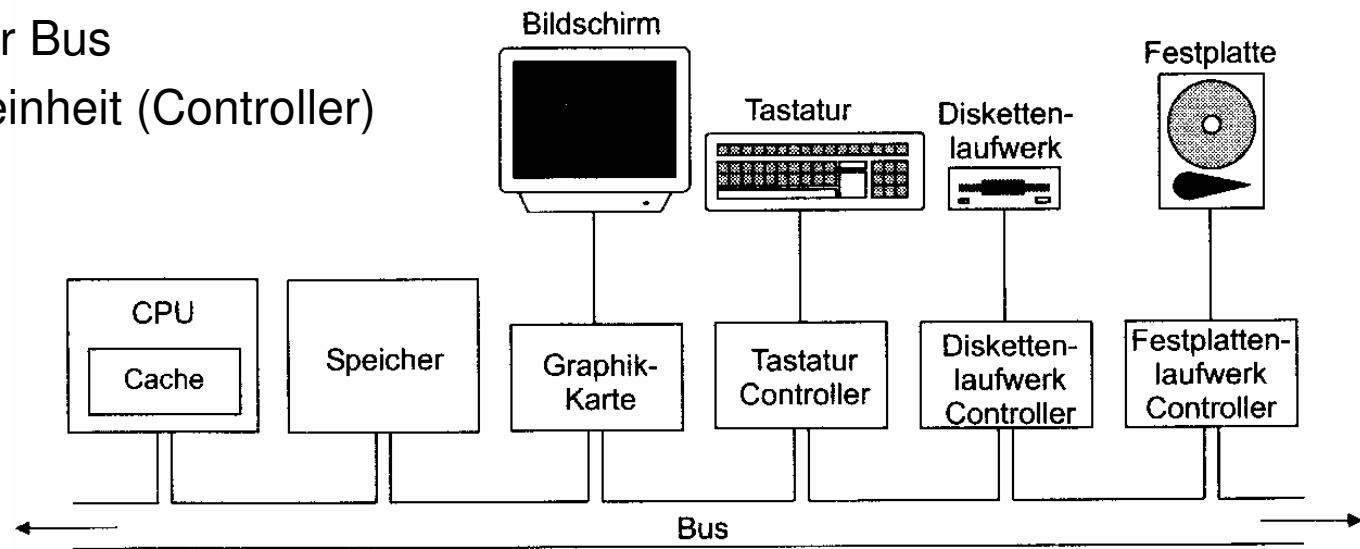
# Aufbau und Funktionsweise eines Computers

Thomas Röfer

Hardware und Software  
von Neumann Architektur  
Schichtenmodell der Software  
Zahlsysteme  
Repräsentation von Daten im Computer

# Hardware

- ▶ **Prozessor (CPU)**
- ▶ **Hauptspeicher**
- ▶ **Peripherie**
  - ▶ Festplatte, Monitor, Tastatur, Maus, CD/DVD Laufwerk/Brenner, Diskettenlaufwerk, Netzwerkkarten
  - ▶ Anbindung über Bus
  - ▶ Eigene Steuereinheit (Controller)





# Software

## ▶ Betriebssystem

- ▶ Isoliert Anwendersoftware von der Hardware
- ▶ Verwaltet Ressourcen des Rechners
- ▶ Firmware/BIOS
- ▶ MSDOS, MS Windows, Linux, MacOS, Solaris, ...
- ▶ PalmOS, MS Windows Mobile, SymbianOS, ...

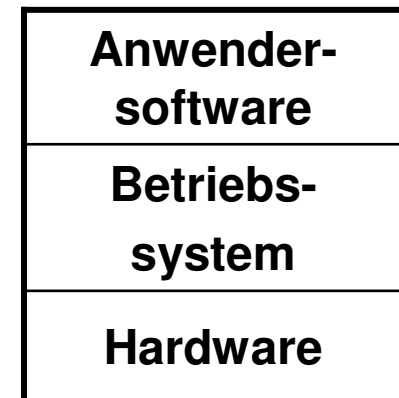
## ▶ Systemsoftware

- ▶ Lösung von Aufgaben im Rechner
- ▶ z.B. Entwicklungswerkzeuge: Compiler, Linker, Debugger

## ▶ Anwendersoftware

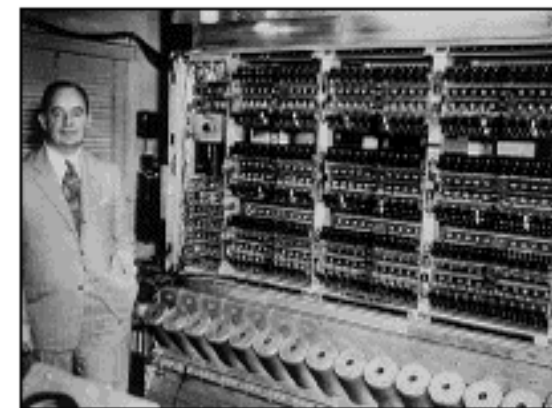
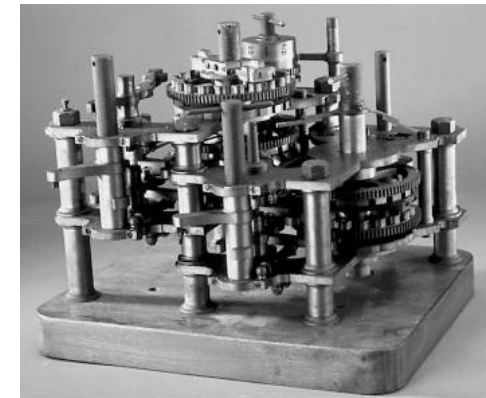
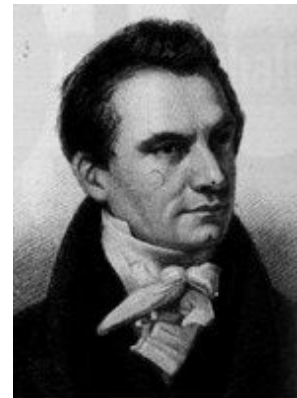
- ▶ Lösung von Problemen der externen Welt der Anwender
- ▶ z.B. Textverarbeitung, Tabellenkalkulation, ..., Spiele

## ▶ Zuordnung zunehmend schwierig



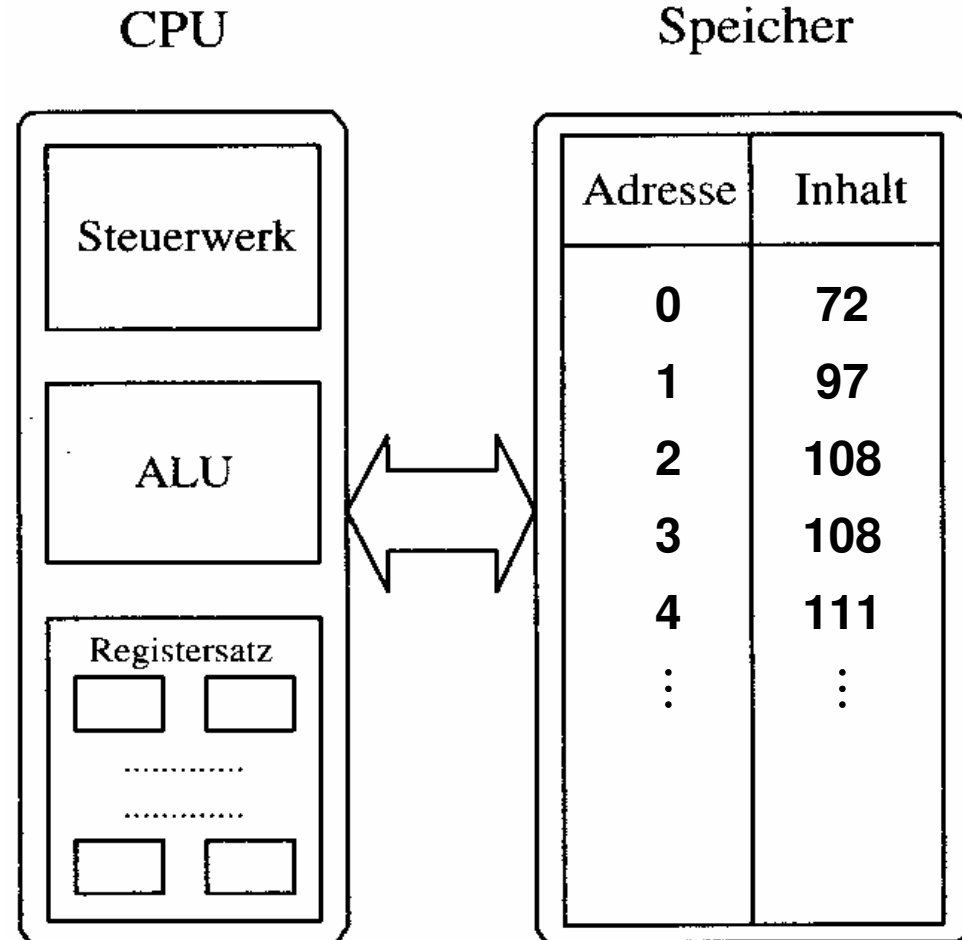
# Grundlagen

- ▶ **Analytical Engine**
  - ▶ Charles Babbage (1834)
  - ▶ Prozessoreinheit
  - ▶ Speichereinheit
  - ▶ Programmsteuerung
- ▶ **Programme**
  - ▶ sind Sequenzen von Befehlen,
  - ▶ werden nacheinander vom Prozessor verarbeitet,
  - ▶ werden auf Daten im Speicher angewendet
- ▶ **von Neumann Architektur**
  - ▶ 1950
  - ▶ elektronischer Rechner



# von Neumann Architektur

- ▶ **Prozessor (CPU)**
  - ▶ Steuerwerk
  - ▶ Arithmetisch-logische Einheit (ALU)
  - ▶ Registersatz
- ▶ **Speicher**
  - ▶ Wahlfreier Zugriff (RAM)
  - ▶ Jede Speicherzelle hat
    - ▶ *eine Adresse*
    - ▶ *einen Inhalt*
  - ▶ Beispiel: ein PC mit 512 MB hat 536.870.912 Speicherzellen
- ▶ **Programm und Daten im selben Speicher**





# Bits und Bytes

- ▶ **1 Bit kann 2 Zustände einnehmen: 0 oder 1**
- ▶ **Mehrere Bits:  $2^{\text{Anzahl}}$  Zustände**
- ▶ **Bei n Bits**
  - ▶ Bit 0 = least significant bit (LSB)
  - ▶ Bit n-1 = most significant bit (MSB)
- ▶ **1 Byte = 8 Bits**
  - ▶  $2^8$  Zustände = 256 Zustände = 0 ... 255
  - ▶ Speicherzellen sind 1 Byte groß
- ▶ **1 Nibble = 4 Bits**
- ▶ **1 Wort**
  - ▶ Anzahl Bits ergibt sich aus der Breite des Datenbusses
  - ▶ Aktuell: IA32: 32 Bit, AMD64/EM64T: 64 Bit
  - ▶ Adresse eines Wortes ist immer die Adresse des ersten Bytes des Wortes
  - ▶ Das LSB kann Teil des ersten Bytes eines Wortes sein (little endian) oder Teil des letzten Bytes (big endian)

Bit 2	Bit 1	Bit 0	
0	0	0	Zustand 0
0	0	1	Zustand 1
0	1	0	Zustand 2
0	1	1	Zustand 3
1	0	0	Zustand 4
1	0	1	Zustand 5
1	1	0	Zustand 6
1	1	1	Zustand 7

Adresse	Little end.	Big endian
n	Bits 0-7	Bits 24-31
n+1	Bits 8-15	Bits 16-23
n+2	Bits 16-23	Bits 8-15
n+3	Bits 24-31	Bits 0-7



# Bits und Bytes

- ▶ **1 Bit kann 2 Zustände einnehmen: 0 oder 1**
- ▶ **Mehrere Bits:  $2^{\text{Anzahl}}$  Zustände**
- ▶ **Bei n Bits**
  - ▶ Bit 0 = least significant bit (LSB)
  - ▶ Bit n-1 = most significant bit (MSB)
- ▶ **1 Byte = 8 Bits**
  - ▶  $2^8$  Zustände = 256 Zustände
  - ▶ Speicher
- ▶ **1 Wort**
  - ▶ **Beispiel: „3“ als Wort**
  - ▶ Little endian: 03 00 00 00
  - ▶ Big endian: 00 00 00 03
- ▶ **Daten**
  - ▶ Aktuell: IA32: 32 Bit, IA64: 64 Bit
  - ▶ Adresse eines Wortes ist immer die Adresse des ersten Bytes des Wortes
  - ▶ Das LSB kann Teil des ersten Bytes eines Wortes sein (little endian) oder Teil des letzten Bytes (big endian)


Bit 2	Bit 1	Bit 0	
0	0	0	Zustand 0
0	0	1	Zustand 1
0	1	0	Zustand 2
0	1	1	Zustand 3
1	0	0	Zustand 4
1	0	1	Zustand 5
1	1	0	Zustand 6
1	1	1	Zustand 7

Adresse	Little end.	Big endian
n	Bits 0-7	Bits 24-31
n+1	Bits 8-15	Bits 16-23
n+2	Bits 16-23	Bits 8-15
n+3	Bits 24-31	Bits 0-7



## Zustände und ihre Interpretationen

- ▶ **Bits im Speicher repräsentieren Zustände**
- ▶ **Ihre Bedeutung hängt davon ab, wie sie interpretiert werden**
  - ▶ 4 Bytes (z.B. 65 117 116 111) können z.B.
    - ▶ *als ganze Zahl gedeutet werden (integer, z.B. 1869903169)*
    - ▶ *als kurze Gleitkommazahl (float, z.B. 7.56561e+028)*
    - ▶ *als Folge von vier Zeichen (string, z.B. „Auto“)*
    - ▶ *als Folge von Befehlen für den Prozessor*

<i>INC</i>	<i>CX</i>
<i>JNZ</i>	<i>+116</i>
<i>DB</i>	<i>111</i>
    - ▶ *als Farbe (vom Grafikchip, z.B. )*
- ▶ **In Programmiersprachen wird die Interpretation durch *Datentypen* festgelegt**

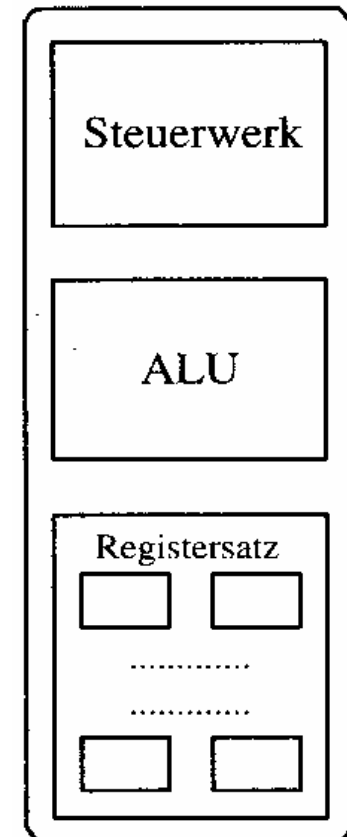
# Instruktionszyklus einer CPU

## ▶ Programmausführung

- ▶ Programme sind in Maschinensprache kodiert
- ▶ Prozessortyp-abhängig (z.B. PowerPC vs. IA32)
- ▶ Spezielles Register: Befehlszähler

## ▶ Fundamentaler Instruktionszyklus einer CPU

1. **Fetch:** Hole den Befehl, dessen Adresse im Befehlszähler steht, aus dem Speicher in das Instruktionsregister.
2. **Increment:** Inkrementiere den Befehlszähler, damit er auf die nächste auszuführende Instruktion verweist.
3. **Decode:** Dekodiere die Instruktion, damit klar wird, was zu tun ist.
4. **Fetch Operands:** Falls nötig, hole die Operanden aus den im Befehl bezeichneten Stellen im Speicher.
5. **Execute:** Führe die Instruktion aus, ggf. durch die ALU. (Bei einem Sprung wird hier ein neuer Wert in den Befehlszähler geschrieben.)
6. **Loop:** Gehe zu Schritt 1 □





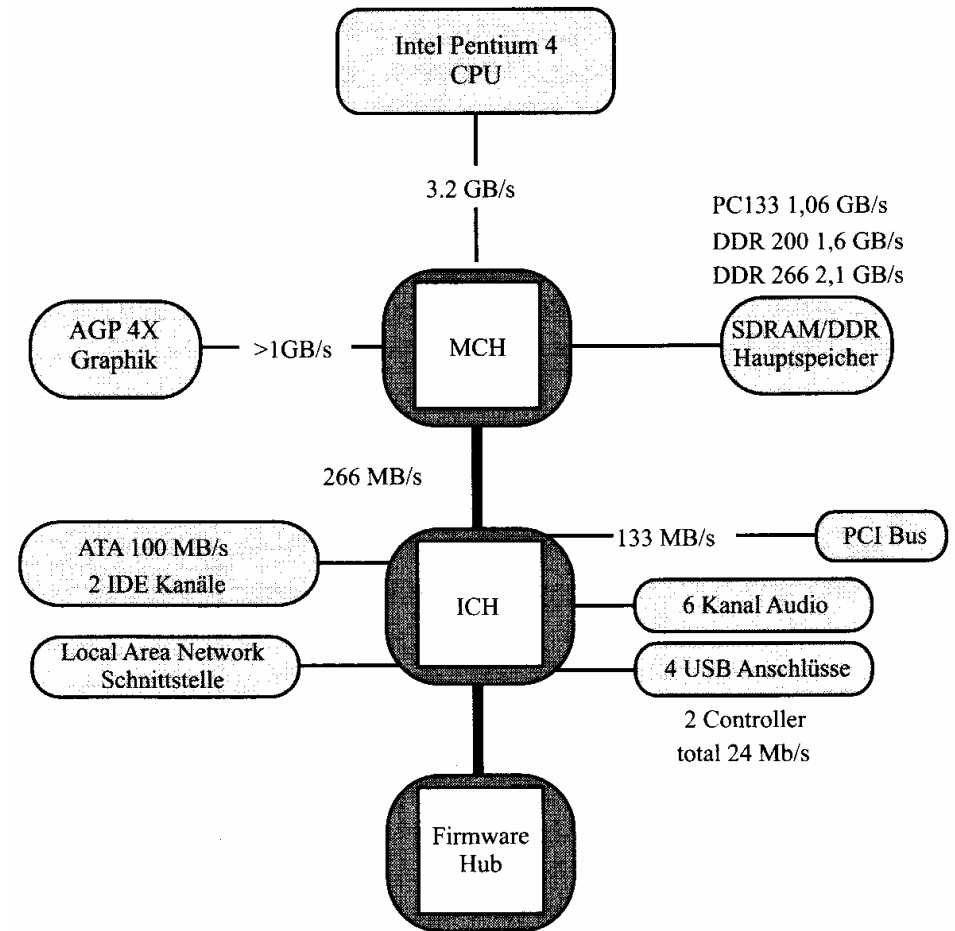
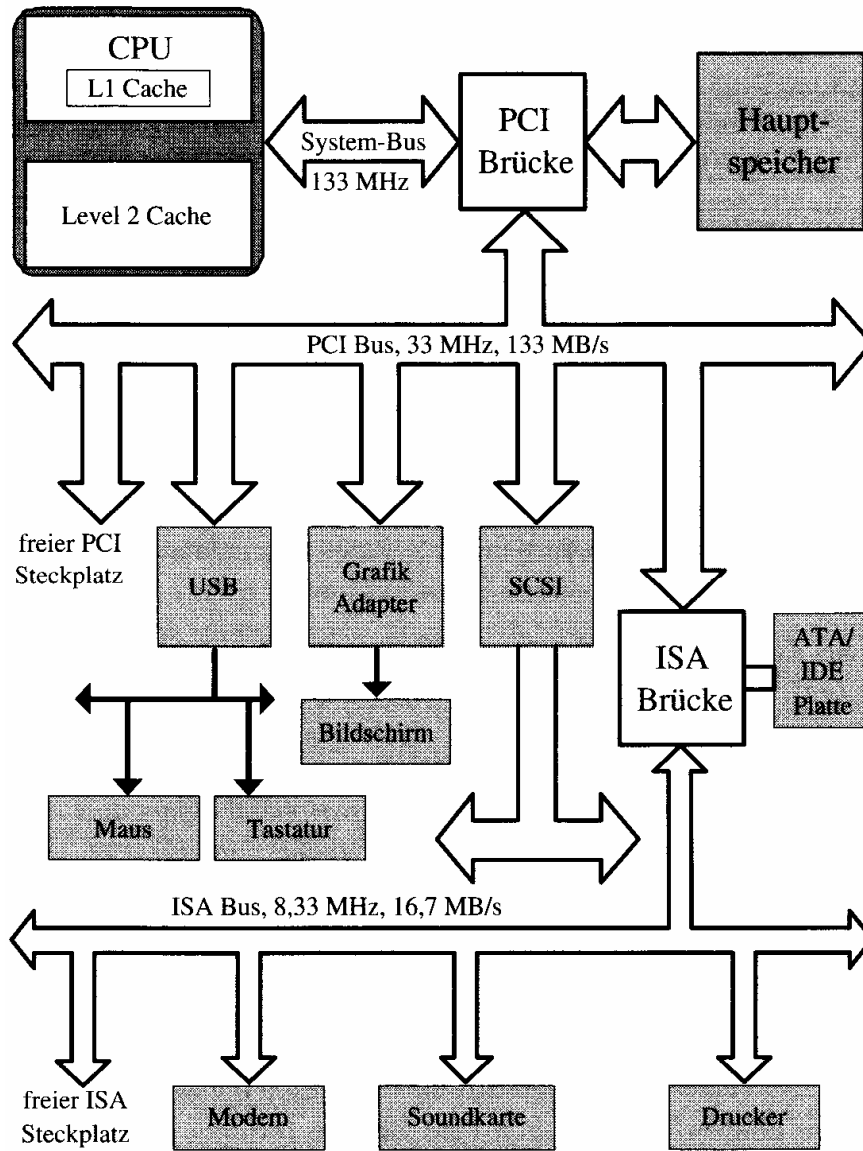
# Programmausführung

- ▶ **Typische Befehle**
  - ▶ laden (Speicher → Register)
  - ▶ schreiben (Register → Speicher)
  - ▶ verknüpfen von Registern (z.B. addieren)
  - ▶ (bedingter) Sprung
- ▶ **Takt**
  - ▶ Befehle werden getaktet ausgeführt
  - ▶ Anzahl Takte/Befehl hängt vom Befehl ab
  - ▶ Beschleunigung durch Fließbandverarbeitung (Pipelining)
- ▶ **von Neumannscher Flaschenhals (bottleneck)**
  - ▶ Prozessor (viel) schneller als Speicher
  - ▶ Dadurch muss Prozessor oft auf Speicher warten
  - ▶ Gebräuchliche Lösungsansätze
    - ▶ *viele CPU-Register*
    - ▶ *schnelle Zwischenspeicher (Cache)*

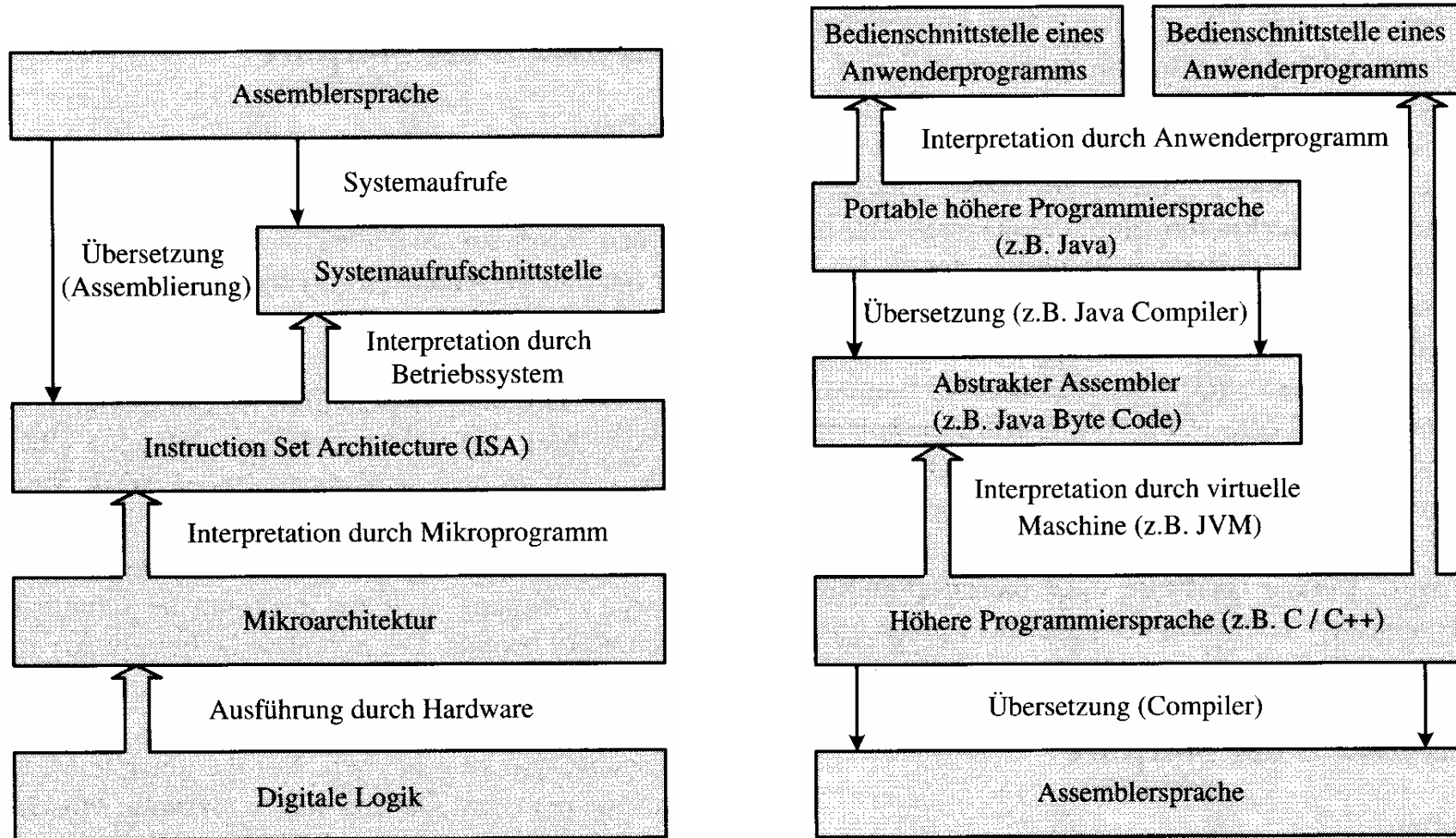


## Moore's Gesetz

- ▶ **Gordon Moore (Intel):**
  - ▶ Die Anzahl der Transistoren pro Chip verdoppelt sich alle 18 Monate
- ▶ **Folgen**
  - ▶ Verdopplung der Speichergröße alle 18 Monate
  - ▶ Verdopplung der Geschwindigkeit alle 18 Monate
- ▶ **Beispiel**
  - ▶ 1981: IBM PC, 4,77 MHz, 64 KB RAM
  - ▶ 2005: Intel P4, 3,8 GHz (nur  $2^{9,6} \cdot 4,77\text{MHz}$ , aber P4 braucht weniger Takte pro Instruktion), 1 GB RAM ( $2^{14} \cdot 64\text{ KB}$ )



# Schichtenmodell der Software





# Betriebssystem

- ▶ **Verwaltet Ressourcen eines Rechners**
  - ▶ Systemaufrufchnittstellen (oft APIs – Application Programming Interfaces)
- ▶ **Verwaltet Prozesse und ihre Ressourcen**
  - ▶ Prozessorzeit (oft in Zeitscheiben)
  - ▶ Speicher
  - ▶ Dateien
  - ▶ Kommunikationspfade
- ▶ **Enthält Gerätetreiber**
  - ▶ bzw. bietet Schnittstellen für Gerätetreiber
- ▶ **Bietet Ein- und Ausgabeströme**
  - ▶ Dateisystem
  - ▶ TCP/IP



# Zahlsysteme

- ▶ Endlich ( $\beta$ ) viele Ziffern (z.B. 0 ... 9 beim Dezimalsystem,  $\beta = 10$ )
- ▶  $\beta$  wird auch Radix genannt
- ▶ Schreibung einer Zahl:  $z_{n-1} \dots z_0$ ,  $0 \leq z_i < \beta$

▶ Wert der Zahl:  $z = \sum_{i=0}^{n-1} z_i \beta^i$

▶ Beispiel

- ▶  $42_{10} = 4_{10} \cdot 10_{10}^1 + 2_{10} \cdot 10_{10}^0$
- ▶  $= 100_2 \cdot 1010_2^1 + 10_2 \cdot 1010_2^0$
- ▶  $= 101000_2 + 10_2$
- ▶  $= 101010_2$
- ▶  $101010_2 = 1_2 \cdot 10_2^5 + 0_2 \cdot 10_2^4 + 1_2 \cdot 10_2^3 + 0_2 \cdot 10_2^2 + 1_2 \cdot 10_2^1 + 0_2 \cdot 10_2^0$
- ▶  $= 1_{10} \cdot 2_{10}^5 + 0_{10} \cdot 2_{10}^4 + 1_{10} \cdot 2_{10}^3 + 0_{10} \cdot 2_{10}^2 + 1_{10} \cdot 2_{10}^1 + 0_{10} \cdot 2_{10}^0$

Dezimal	Dual
0	...0000
1	...0001
2	...0010
3	...0011
4	...0100
5	...0101
6	...0110
7	...0111
8	...1000
9	...1001



# Dezimal → Dualsystem

## ▶ Algorithmus

1. Ist die umzurechnende Zahl gerade, schreibe ein 0 sonst eine 1.
2. Teile die umzurechnende Zahl durch 2 (mit Abrunden).
3. Ist die umzurechnende Zahl nicht Null, weiter mit 1.
  - ▶ Die Ziffern 0 und 1 müssen von rechts nach links geschrieben werden.

## ▶ Beispiel

$$\begin{aligned} & \text{▶ } 42_{10} = 42_{10} \cdot 1_2 \\ & \text{▶ } = 21_{10} \cdot 10_2 + 0_2 \\ & \text{▶ } = 10_{10} \cdot 100_2 + 10_2 \\ & \text{▶ } = 5_{10} \cdot 1000_2 + 010_2 \\ & \text{▶ } = 2_{10} \cdot 10000_2 + 1010_2 \\ & \text{▶ } = 1_{10} \cdot 100000_2 + 01010_2 \\ & \text{▶ } = 101010_2 \end{aligned}$$



## Negative Zahlen (Möglichkeiten)

### ▶ Vorzeichenbit

- ▶ MSB für Vorzeichen reservieren (0 → positiv, 1 → negativ)
- ▶  $-3_{10} = -0011_2 \equiv 1011_2$

### ▶ Einerkomplement

- ▶ Alle Bits invertieren
- ▶  $-3_{10} \equiv 1100_2$

### ▶ Zweierkomplement

- ▶ Alle Bits invertieren und eins dazuzählen
- ▶  $-3_{10} \equiv 1101_2$
- ▶ Entspricht:  $16_{10} - 3_{10} = 13_{10} = 1101_2$
- ▶ Vorteil:  $5_{10} - 3_{10} = 5_{10} + -3_{10}$ 
  - ▶  $\equiv 0101_2 + 1101_2$
  - ▶  $= \underline{10010}_2$
- ▶ Grund:  $\equiv 5_{10} + (16_{10} - 3_{10}) = 16_{10} + (5_{10} - 3_{10})$

### ▶ Modulo-Arithmetik

- ▶  $(5_{10} - 3_{10}) \bmod 2_{10}^4$
- ▶  $= (5_{10} + -3_{10}) \bmod 2_{10}^4$
- ▶  $= (0101_2 + 1101_2) \bmod 10000_2$
- ▶  $= 10010_2 \bmod 10000_2$
- ▶  $= 0010_2$



# Weitere Zahlssysteme

## ▶ Hexadezimalsystem

- ▶ Radix  $\beta = 16$ , Ziffern 0 ... 9, A, B, C, D, E, F
- ▶ Eine Hexadezimalziffer entspricht 4 Bit
- ▶ ein Byte kann durch 2 Ziffern dargestellt werden

## ▶ Oktalsystem

- ▶ Radix  $\beta = 8$ , Ziffern 0 ... 7
- ▶ Eine Oktalziffer entspricht 3 Bit
- ▶ Wird eher selten verwendet

Dezimal	Zweierkomplement
+8	nicht darstellbar
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000



# Zeichensätze

- ▶ EBCDIC (IBM)
- ▶ ASCII (ISO 7 Bit)
- ▶ ISO Latin 1 (ISO 8859-1)
- ▶ Unicode (16 Bit)
- ▶ UTF-8

oct	hex	0	20	40	60	100	120	140	160
		0	10	20	30	40	50	60	70
0	0	nul	dle		0	@	P	'	p
1	1	soh	dc1	!	1	A	Q	a	q
2	2	stx	dc2	“	2	B	R	b	r
3	3	etx	dc3	#	3	C	S	c	s
4	4	eot	dc4	\$	4	D	T	d	t
5	5	enq	nak	%	5	E	U	e	u
6	6	ack	syn	&	6	F	V	f	v
7	7	bel	etb	'	7	G	W	g	w
10	8	bs	can	(	8	H	X	h	x
11	9	ht	em	)	9	I	Y	i	y
12	A	lf	sub	*	:	J	Z	j	z
13	B	vt	esc	+	;	K	[	k	{
14	C	ff	fs	,	<	L	\	l	
15	D	cr	gs	-	=	M	]	m	}
16	E	so	rs	.	>	N	^	n	~
17	F	si	us	/	?	O	_	o	del



# Gleitkommazahlen (IEEE 754)

## ▶ Darstellung

- ▶ Mantisse und Exponent

$$z = (-1)^v \cdot \text{Mantisse} \cdot 2^{\text{Exponent}}$$

## ▶ Darstellungsbereiche

- ▶ float:  $\pm 1.40239846E-45$  bis  $\pm 3.40282347E+38$
- ▶ double:  $\pm 4.94065645841246544E-324$  bis  $\pm 1.79769313486231570E+308$

## ▶ Vorteile

- ▶ Erlauben Nachkommastellen
- ▶ Decken sehr großen Zahlenbereich ab

## ▶ Nachteile

- ▶ Abdeckung ist lückenhaft
- ▶ Rundungsfehler

## ▶ Beispiel

- ▶  $1 + 10^{-20} - 1 = 0$

float

v	30 - Exponent - 23	22 - Mantisse - 0
1 Bit	8 Bit	23 Bit

double

v	62 - Exponent - 52	51 - Mantisse - 0
1 Bit	11 Bit	52 Bit