Einführung in die Informatik 1

– Aufbau und Funktionsweise eines Computers –

Sven Kosub

AG Algorithmik/Theorie komplexer Systeme
Universität Konstanz

E 202 | Sven.Kosub@uni-konstanz.de | Sprechstunde: Freitag, 12:30-14:00 Uhr, o.n.V.

Wintersemester 2008/2009

Informatik

Informatik =
Wissenschaft von der systematischen Verarbeitung von Informationen, insbesondere ...

... der automatischen Verarbeitung mit Hilfe von Rechenanlagen
= Computerwissenschaft

zentrale Gegenstände der Informatik:

- Information $\surd$
- Algorithmen ($\simeq$ Systematik der Verarbeitung) demnächst
- Computer ($\simeq$ Rechenanlagen) heute

Computer

Grundbestandteile eines Computers:

- **Hardware**
  - ist fest gegeben
  - kann angefasst werden
  - ist unveränderlich (bis auf Austausch von Komponenten)
- **Software**
  - besteht aus gespeicherten Programmen, die durch Hardware ausgeführt werden
  - ist unsichtbar
  - ist sehr leicht zu ändern (durch Änderung magnetischer, optischer oder elektrischer Zustände)

Computer: Hardware

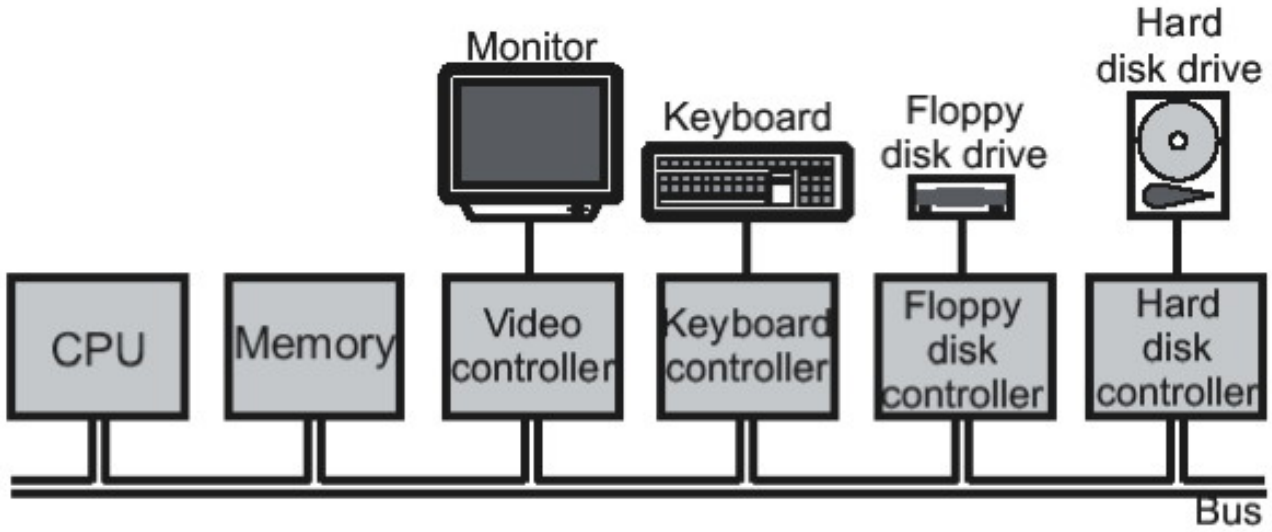
Hardware-Komponenten eines (einfachen) Computers:

- Prozessor (CPU) (*central processing unit*)
- Hauptspeicher (*main memory*)

- Monitor
- Festplatten (*hard disk*)
- Maus
- Tastatur (*keyboard*)
- Laufwerke für CD-ROM/DVD (*drive*)
- Diskettenlaufwerk (*floppy disk drive*)
- Netzwerkkarten (*network board*)
- ...

} Peripherie

Computer: Hardware

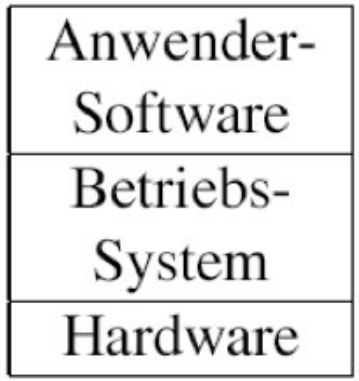


- Datenaustausch zwischen Komponenten über Verbindungskanäle (*channels*)
- Realisierung meist als Bus (von lat. *omnibus*, „für alle“)
- Komponenten (außer CPU) über Controller mit Bus verbunden
- PC: Hauptplatine (*motherboard*) mit CPU, Hauptspeicher, Busse, Controller, Steckplätze (*slots*)

Computer: Software

Software-Komponenten eines Computers:

- Anwendersoftware (*application software*)
- Betriebssystem (*operating system*)

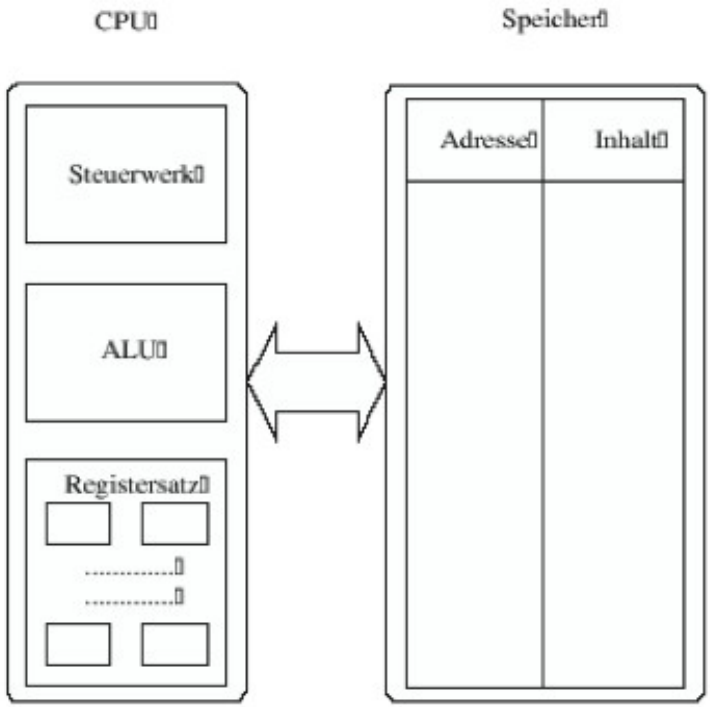


Was sind Programme?

- Sequenzen von Befehlen an Prozessor
- Prozessor arbeitet Befehle nacheinander ab
- wendet Befehle auf Daten im Speicher an

Wo stehen die Programme?

Computer: von Neumann-Architektur



- Programme und Daten stehen im Hauptspeicher
- keine explizite Trennung zwischen Befehlen und Daten
- Befehle und Daten müssen gleich repräsentiert sein

Computer: Speicherorganisation

- 1 Bit: kleinste Speichereinheit mit zwei logischen Zuständen
  - logische Zustände sind 1 und 0 („wahr“und „falsch“)
  - logische Zustände entsprechen physikalischen Zuständen (z.B. liegt Spannung an oder nicht in bestimmtem Schaltkreis)
- 2 Bit: $2^2 = 4$ Zustände darstellbar
- 8 Bit (=1 Byte): $2^8 = 256$ Zustände darstellbar

Bit $n-1$	Bit $n-2$	...	Bit 1	Bit 0	
0	0	...	0	0	Zustand 0
0	0	...	0	1	Zustand 1
0	0	...	1	0	Zustand 2
0	0	...	1	1	Zustand 3
⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	1	Zustand $2^n - 1$

Bit $n-1$ heißt signifikantestes Bit (*most significant bit*)

Bit 0 heißt am wenigsten signifikantes Bit (*least significant bit*)

Computer: Speicherorganisation

Hauptspeicher:

- physikalisch realisiert durch elektronische Bauteile
- logisch organisiert als Aneinanderreihung von Speicherzellen
- Zelle hat Adresse und Inhalt, d.h. über Adresse wird Inhalt ausgelesen oder beschrieben
- wahlfreier Zugriff (*random access memory*, RAM), d.h. Zugriff dauert für alle Zellen gleich lang

Einheiten für die Speichergröße:

- 1 Byte ist kleinste addressierbare Speichereinheit
- 1 Kilobyte (1 KB) $= 2^{10}$ Byte $= 1.024$ Byte
- 1 Megabyte (1 MB) $= 2^{20}$ Byte $= 1.024$ KB
- 1 Gigabyte (1 GB) $= 2^{30}$ Byte $= 1.024$ MB

Beachte: 1kB (1.000 Byte) $\neq$ 1KB (1.024 Byte)

Computer: Speicherorganisation

weitere wichtige Einheiten:

- 1 Wort = 4 Byte = 32 Bit
- 1 Halbwort = 2 Byte = 16 Bit
- 1 Doppelwort = 8 Byte = 64 Bit

Verarbeitungsbreite (in einem Takt verarbeitbare Bits) heutiger Rechner:

- PCs mit Intel Pentium 32 Bit (Itanium-2 64 Bit)
- PowerPC G4: 64 Bit
- RISC-Workstations: 64 Bit

Adressraum eines Prozessors:

- maximale Anzahl adressierbarer Speicherzellen
- 32 Bit Verarbeitungsbreite: 2^{32} Byte $= 4 \cdot 2^{30}$ Byte $= 4$ GB
- 64 Bit Verarbeitungsbreite: 2^{64} Byte $= 2^{34}$ GB $= 16$ EB

Binärcodierung elementarer Datentypen

- Computer können nur Bitmuster speichern
- Binärcode ist Abbildung von Klartext in Bitmuster
- für verschiedene Datentypen (Zahlen, Buchstaben, Befehle) verschiedene Binärcodes
- für Decodierung eines Bitmuster Typ notwendig

Wie codiert man binär:

- (endlich große) natürliche Zahlen?
- (endlich große) ganze Zahlen?
- (endlich große) Gleitkommazahlen?
- Texte?

Binärcodierung von natürlichen Zahlen

Wie codiert man binär:

- (endlich große) natürliche Zahlen?
- (endlich große) ganze Zahlen?
- (endlich große) Gleitkommazahlen?
- Texte?

Binärcodierung von natürlichen Zahlen

Zahlsysteme zur Basis $\beta \geq 2$ (d.h. mit Ziffern $0, 1, \ldots, \beta - 1$):

- schreibe natürliche Zahl z mit n Stellen als Polynom

$$z = \sum_{i=0}^{n-1} z_i \cdot \beta^i \qquad \text{wobei } 0 \leq z_i < \beta$$

wichtige Zahlsysteme:

- $\beta = 10$: Dezimalsystem
- $\beta = 2$: Binärsystem
- $\beta = 8$: Oktalsystem
- $\beta = 16$: Hexadezimalsystem

- $23_{10} = 10111_2 = 27_8 = 17_{16}$
- $15_{10} = 1111_2 = 17_8$
- $7_{10} = 7_8 = 111_2$
- $11_{10} = 1011_2$

Binärcodierung von natürlichen Zahlen

Rechnen mit Binärdarstellungen:

- wie im Dezimalsystem mit Überträgen schon bei 2 und nicht bei 10
- Addition:

$$1 \cdot 2^i + 1 \cdot 2^i = 1 \cdot 2^{i+1} + 0 \cdot 2^i,$$

  also z.B. $1_2 + 1_2 = 10_2$
- Multiplikation:

$$\left(\sum_{i=0}^{n-1} z_i \cdot 2^i \right) \cdot 2 = \sum_{i=0}^{n-1} z_i \cdot 2^{i+1} = \sum_{i=1}^{n} z_{i-1} \cdot 2^i + 0 \cdot 2^0,$$

  d.h. Multiplikation mit 2 verschiebt Ziffernfolge um 1 Stelle nach links
- Division durch 2 verschiebt Ziffernfolge um 1 Stelle nach rechts, Ziffer an Stelle 0 ist Divisionsrest

Exkurs: Modulo-Arithmetik

Arithmetik modulo einer natürlichen Zahl:

- n sei ganze Zahl und $k \geq 2$ sei natürliche Zahl
- Modulus von n bezüglich k ist eine natürliche Zahl r mit $0 \leq r \leq k-1$, so dass es eine ganze Zahl t gibt mit

$$n = t \cdot k + r$$

- beachte: t und r sind eindeutig für n und k
- definieren Modulo-Funktion:

$$\text{mod}(n, k) = r$$

- alternativ: $n \equiv r \mod k$ (lies: n ist kongruent r modulo k)

- $\text{mod}(25, 9) = 7$ bzw. $25 \equiv 7 \mod 9$, denn: $25 = 2 \cdot 9 + 7$
- $\text{mod}(-25, 9) = 2$ bzw. $-25 \equiv 2 \mod 9$, denn: $-25 = -3 \cdot 9 + 2$
- $\text{mod}(18, 9) = 0$ bzw. $18 \equiv 0 \mod 9$, denn: $18 = 2 \cdot 9 + 0$

Exkurs: Modulo-Arithmetik

- $\mathrm{mod}(25, 9) = 7$ bzw. $25 \equiv 7 \mod 9$, denn: $25 = 2 \cdot 9 + 7$
- $\mathrm{mod}(-25, 9) = 2$ bzw. $-25 \equiv 2 \mod 9$, denn: $-25 = -3 \cdot 9 + 2$
- $\mathrm{mod}(18, 9) = 0$ bzw. $18 \equiv 0 \mod 9$, denn: $18 = 2 \cdot 9 + 0$

								...
-27	-26	**-25**	-24	-23	-22	-21	-20	-19
-18	-17	-16	-15	-14	-13	-12	-11	-10
-9	-8	-7	-6	-5	-4	-3	-2	-1
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	**25**	26
...								

Berechnung der Modulo-Funktion:

$$\mathrm{mod}(n, k) = n - \left\lfloor \frac{n}{k} \right\rfloor \cdot k$$

Exkurs: Modulo-Arithmetik

Rechenregel für Modulo-Funktion:

$$\begin{aligned} \mathrm{mod}(n+m,k) &= \mathrm{mod}(\mathrm{mod}(n,k)+\mathrm{mod}(m,k),k) \\ \mathrm{mod}(n\cdot m,k) &= \mathrm{mod}(\mathrm{mod}(n,k)\cdot\mathrm{mod}(m,k),k) \end{aligned}$$

- Wollen $\mathrm{mod}(11-26,9)$ berechnen:

$$\begin{aligned} \mathrm{mod}(11-26,9) &= \mathrm{mod}(-15,9) = 3 \\ \mathrm{mod}(11-26,9) &= \mathrm{mod}(\mathrm{mod}(11,9)+\mathrm{mod}(-26,9),9) \\ &= \mathrm{mod}(2+1,9) = 3 \end{aligned}$$

- Wollen $\mathrm{mod}(11\cdot(-26),9)$ berechnen:

$$\begin{aligned} \mathrm{mod}(11\cdot(-26),9) &= \mathrm{mod}(\mathrm{mod}(11,9)\cdot\mathrm{mod}(-26,9),9) \\ &= \mathrm{mod}(2\cdot 1,9) = 2 \\ \mathrm{mod}(11\cdot(-26),9) &= \mathrm{mod}(-286,9) = ? \end{aligned}$$

Binärcodierung von natürlichen Zahlen

(binäre) Arithmetik fixer Länge n:

- mathematisch: Arithmetik modulo 2^n
- Binärcode von 2^n benötigt $n+1$ Stellen, denn:

$$2^n = 1\underbrace{0\ldots0}_{n\text{-mal}}{}_2$$

- auf n Stellen genau ist 2^n gleich 0, mathematisch: $2^n \equiv 0 \mod 2^n$
- d.h. bei Übertrag an Stelle $n+1$ wird Übertrag ignoriert
- beachte: Rechnung nur modulo 2^n korrekt

Für $n = 4$ gilt $01000_2 + 01001_2 \equiv 10001_2 \mod 10000_2$,

Binärcodierung von natürlichen Zahlen

Konversion:

- Umwandlung von Zahldarstellungen aus einem Zahlsystem in ein anderes
- Computer: Binärsystem $\longleftrightarrow$ Monitor (Mensch): Dezimalsystem

Im Folgenden:

- Dezimal-Binär-Konversion
- Binär-Dezimal-Konversion

Binärcodierung von natürlichen Zahlen

Dezimal-Binär-Konversion

- Zahl in Dezimaldarstellung als Ziffernfolge $z_{n-1} \dots z_1 z_0$ gegeben
- verwenden Horner-Schema:

$$
\begin{aligned}
z &= \sum_{i=0}^{n-1} z_i \cdot 10^i \\
&= z_{n-1} \cdot 10^{n-1} + z_{n-2} \cdot 10^{n-2} + z_{n-3} \cdot 10^{n-3} + \dots + z_1 \cdot 10 + z_0 \\
&= (z_{n-1} \cdot 10 + z_{n-2}) \cdot 10^{n-2} + z_{n-3} \cdot 10^{n-3} + \dots + z_1 \cdot 10 + z_0 \\
&= ((z_{n-1} \cdot 10 + z_{n-2}) \cdot 10 + z_{n-3}) \cdot 10^{n-3} + \dots + z_1 \cdot 10 + z_0 \\
&\vdots \\
&= (\dots((z_{n-1} \cdot 10 + z_{n-2}) \cdot 10 + z_{n-3}) \cdot 10 + \dots + z_1) \cdot 10 + z_0 \\
&= (\dots((d_{n-1} \cdot 1010_2 + d_{n-2}) \cdot 1010_2 + d_{n-3}) \cdot 1010_2 + \dots + d_1) \\
&\qquad \cdot 1010_2 + d_0
\end{aligned}
$$

wobei d_i Binärdarstellung der Ziffern z_i

Binärcodierung von natürlichen Zahlen

Konvertiere 2008_{10} in Binärdarstellung:

$$\begin{aligned}
2008_{10} &= 2 \cdot 1000 + 0 \cdot 100 + 0 \cdot 10 + 8 \cdot 1 \\
&= ((2 \cdot 10 + 0) \cdot 10 + 0) \cdot 10 + 8 \\
&= ((10_2 \cdot 1010_2 + 0_2) \cdot 1010_2 + 0_2) \cdot 1010_2 + 1000_2 \\
&= (10100_2 \cdot 1010_2 + 0_2) \cdot 1010_2 + 1000_2 \\
&= 11001000_2 \cdot 1010_2 + 1000_2 \\
&= 11111011000_2
\end{aligned}$$

Gegenprobe: $11111011000_2 = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 = 2008_{10}$

algorithmischer Ablauf des Horner-Schema:

- setze $h_n =_{\text{def}} 0$
- setze $h_{i-1} =_{\text{def}} h_i \cdot 1010_2 + d_{i-1}$ für $i = n, n-1, \ldots, 1$
- h_0 ist die Binärdarstellung

Binärcodierung von natürlichen Zahlen

Binär-Dezimal-Konversion

- Zahl D in Binärdarstellung gegeben
- verwenden inverses Horner-Schema:

$$
\begin{aligned}
D_0 &=_{\text{def}} D \\
d_0 &=_{\text{def}} D_0 \mod 1010_2 \\
D_1 &=_{\text{def}} D_0/1010_2 \qquad \text{(ganzzahlige Division)} \\
d_1 &=_{\text{def}} D_1 \mod 1010_2 \\
&\vdots \\
D_i &=_{\text{def}} D_{i-1}/1010_2 \\
d_i &=_{\text{def}} D_i \mod 1010_2
\end{aligned}
$$

- Abbruch wenn $D_i = 0$
- wandle d_i in Dezimalziffern z_i um
- beachte: Ziffernfolge muss noch umgedreht werden

Binärcodierung von ganzen Zahlen

Wie codiert man binär:

- (endlich große) natürliche Zahlen?
- (endlich große) ganze Zahlen?
- (endlich große) Gleitkommazahlen?
- Texte?

Binärcodierung von ganzen Zahlen

Darstellung negativer Zahlen (mit n Stellen):

- $\overline{z} =_{\text{def}} 2^n - z$ ist Zweierkomplement von z
- es gilt $-z \equiv 2^n - z \equiv \overline{z} \mod 2^n$
- Binärzahlen von 0 bis $2^{n-1} - 1$ stehen für

$$\{0, \ldots, 2^{n-1} - 1\}$$

- Binärzahlen von 2^{n-1} bis $2^n - 1$ stehen für

$$\{-(2^n - 2^{n-1}), \ldots, -(2^n - (2^n - 1))\}$$
$$= \{-2^{n-1}, \ldots, -1\}$$

- Verträglichkeit mit Arithmetik:

$$x - y \equiv x + (2^n - y) \equiv x + \overline{y} \mod 2^n$$
$$\overline{0} \equiv 2^n - 0 \equiv 0 \mod 2^n$$

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

Binärcodierung von ganzen Zahlen

kürzere Darstellung von Binärzahlen:

- Oktalzahlen: Basis $\beta = 8 = 2^3$, d.h. Blöcke der Größe 3 werden zu Ziffern aus $\{0, 1, \ldots, 7\}$ zusammengefasst

$$2008_{10} = 11111011000_2 = \underbrace{011}_{3}\underbrace{111}_{7}\underbrace{011}_{3}\underbrace{000}_{0} = 3730_8$$

- Hexadezimalzahlen: Basis $\beta = 16 = 2^4$, d.h. Blöcke der Größe 4 werden zu Ziffern aus $\{0, 1, \ldots, 9, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}\}$ zusammengefasst

$$2008_{10} = 11111011000_2 = \underbrace{0111}_{7}\underbrace{1101}_{\mathrm{d}}\underbrace{1000}_{8} = 7\mathrm{d}8_{16}$$

Binärcodierung von Gleitkommazahlen

Wie codiert man binär:

- (endlich große) natürliche Zahlen?
- (endlich große) ganze Zahlen?
- (endlich große) Gleitkommazahlen?
- Texte?

Binärcodierung von Gleitkommazahlen

Gleitkommazahlen (*floating point numbers*):

- endliche Approximationen reeller Zahlen
- Repräsentierung mittels Vorzeichen, Mantisse und Exponent:

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

- Normalisierung der Mantisse auf eine Stelle vor dem Komma

Umsetzung (nach IEEE-Standard):

- einfache Genauigkeit (*single precision*) mit Datentyp *float* (32 Bit)
- doppelte Genauigkeit (*double precision*) mit Datentyp *double* (64 Bit)

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

Binärcodierung von Gleitkommazahlen

- normierte Mantisse $\neq 0$ beginnt immer mit 1, d.h. 1 wird nicht gespeichert
- 23 Bit (bzw. 52 Bit) repräsentieren die Nachkommazahlen
- Exponent e ohne Vorzeichen; dafür abhängig vom Datentyp Verschiebung b, d.h. e wird stets als $e - b$ interpretiert
- besondere Zahlen:

v	Exponent	Mantisse	Zahlwert
0	0	0	0
1	0	0	0
0	11...1	0	$+\infty$
1	11...1	0	$-\infty$
0	11...1	$\neq 0$	`NaN` (*not a number*)

Binärcodierung von Gleitkommazahlen

Zahlbereiche der Datentypen:

- **float**
  - Verschiebung $b = 127$, Exponententeil zwischen 2^{-126} bis 2^{127}
    dezimal: ca. 10^{-38} bis 10^{38}
  - Mantissen mit Abstand 2^{-23}
    dezimal: ca. $1,19209289551 \cdot 10^{-7}$
    (auf 7 Stellen genau)

- **double**
  - Verschiebung $b = 1023$, Exponententeil zwischen 2^{-1022} bis 2^{1023}
    dezimal: ca. $2 \cdot 10^{-307}$ bis 10^{308}
  - Mantissen mit Abstand 2^{-52}
    dezimal: ca. $2,22044604925 \cdot 10^{-16}$
    (auf 16 Stellen genau)

Binärcodierung von Gleitkommazahlen

Rechnen mit Gleitkommazahlen:

- Addition: für gleiche Exponenten werden Mantissen addiert (und normalisiert)
- Multiplikation: Multipliziere Mantissen und addiere Exponenten (und normalisiere)
- Rundungsfehler bei sehr kleinen und sehr großen Zahlen

Warnung: Für bestimmte x ist $10^x + 1 - 1 \neq 10^x$

- verschiedene Verfahren für die gleiche Größe mit unterschiedlichen Wirkungen:
  - stabil: kleine Fehler bei Eingangsgrößen mit kleinen Fehlern bei Ausgangsgrößen
  - instabil: kleine Fehler bei Eingangsgrößen mit großen Fehlern bei Ausgangsgrößen

Binärcodierung von Gleitkommazahlen

Wie ist Mantissenwert einer float-Zahl zu interpretieren?

Binäre Expansion einer reellen Zahl $0 \leq z \leq 1$:

- (möglicherweise unendliche) Folge $0, b_1 b_2 \ldots_2$ von Binärziffern b_i mit

$$z = \sum_{i=1}^{\infty} 2^{-b_i}$$

- nicht immer eindeutig, z. B. $\frac{1}{2} = 0,1_2$ und $\frac{1}{2} = 0,011111\ldots_2$
- Konversion von endlichem Dezimalbruch in unendliche binäre Expansion möglich, z. B.

$$0,1_{10} = 0,0001100110011\overline{0011}_2$$

Binärcodierung von Gleitkommazahlen

Zur Probe sei $0,0001100110011\overline{0011}_2$ binäre Expansion einer reellen Zahl z:

$$\begin{aligned} z &= 2^{-4}+2^{-5}+2^{-8}+2^{-9}+2^{-12}+2^{-13}+2^{-16}+2^{-17}+\cdots \\ &= \sum_{i=1}^{\infty}\left(2^{-4i}+2^{-4i-1}\right) = \sum_{i=1}^{\infty}\left(2^{-4}\right)^i+\frac{1}{2}\sum_{i=1}^{\infty}\left(2^{-4}\right)^i = \frac{3}{2}\sum_{i=1}^{\infty}\left(2^{-4}\right)^i \\ &= \frac{3}{2}\cdot\left(\frac{1}{1-2^{-4}}-1\right)=\frac{3}{2}\cdot\left(\frac{2^4}{2^4-1}-1\right)=\frac{3}{2}\cdot\frac{2^4-(2^4-1)}{2^4-1}=\frac{3}{2}\cdot\frac{1}{15}=\frac{1}{10} \end{aligned}$$

Darstellung von $0,1_{10}$ als Gleitkommazahl vom Typ float:

- normalisierte Gleitkommazahl: $1,10011001100110011\overline{0011}_2\cdot 2^{-4}$
- Vorzeichen $v=0$
- Exponent $e=123_{10}=01111011_2$ (wegen $e-127=-4$)
- Mantisse kann auf- oder abgerundet werden (nur 23 Bit):

$$\begin{aligned} 0\,01111011\,10011001100110011001100_{\text{float}} &\approx 0,099999994039 54_{10} \\ 0\,01111011\,10011001100110011001101_{\text{float}} &\approx 0,100000001490 12_{10} \end{aligned}$$

- absoluter Fehler beträgt $2^{-23}\cdot 2^{-4}=2^{-27}\approx 7,45\cdot 10^{-9}$

Binärcodierung von Gleitkommazahlen

Fallstudie: The Patriot Missile Failure

- 25.02.1991: irakische Scud trifft amerikanische Raketenbasis in Dhahran (Saudi-Arabien)
- 28 getötete Soldaten und ca. 100 Verletzte
- Flugabwehrrakete vom Typ Patriot verfehlte Scud

Ursache: Fehlbehandlung von Rundungsfehlern durch Software

- interner Zähler für Zeit seit Aktivierung (1 Takt = 10 s)
- zur Berechnung physikalischer Parameter Zähler mit $\frac{1}{10}$ multipliziert
- Realisierung mittels 23-Bit-Mantisse für Gleitkommaoperationen
- Konversion von $0,1_{10}$ in float-Zahl mit Fehler $\approx 7 \cdot 10^{-9}$
- Rakete ca. 100 h aktiv, d.h. Fehler ca. $7 \cdot 10^{-8} \cdot 100 \cdot 60 \cdot 60 \cdot 10\text{s} = 0,0252\text{s}$
- Scud-Rakete mit ca. 1.676 $\text{m}\cdot\text{s}^{-1}$, d.h. in 0,0252s ungefähr 42 m

Quelle: GAO/IMTEC-92-26

Binärcodierung von Texten

Wie codiert man binär:

- (endlich große) natürliche Zahlen?
- (endlich große) ganze Zahlen?
- (endlich große) Gleitkommazahlen?
- Texte?

Binärcodierung von Texten

- Text ist Folge von Symbolen
- Symbole einzeln codieren
- n Bit codieren $\leq 2^n$ Buchstaben
- 26 Buchstaben mit 5 Bit ($2^4 < 26 < 2^5$)
- ISO 7 Bit (ASCII)
- ISO 8 Bit (erweitertes ASCII)
- ISO 16 Bit (UTF-16, Unicode)

oct	hex	0 0	20 10	40 20	60 30	100 40	120 50	140 60	160 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

- „Dieser Text ist geheim“ entspricht in ASCII-Codierung 44 69 65 73 65 72 20 54 65 78 74 20 69 73 74 20 67 65 68 65 69 6d
- 2008 entspricht in ASCII-Codierung 32 30 30 38

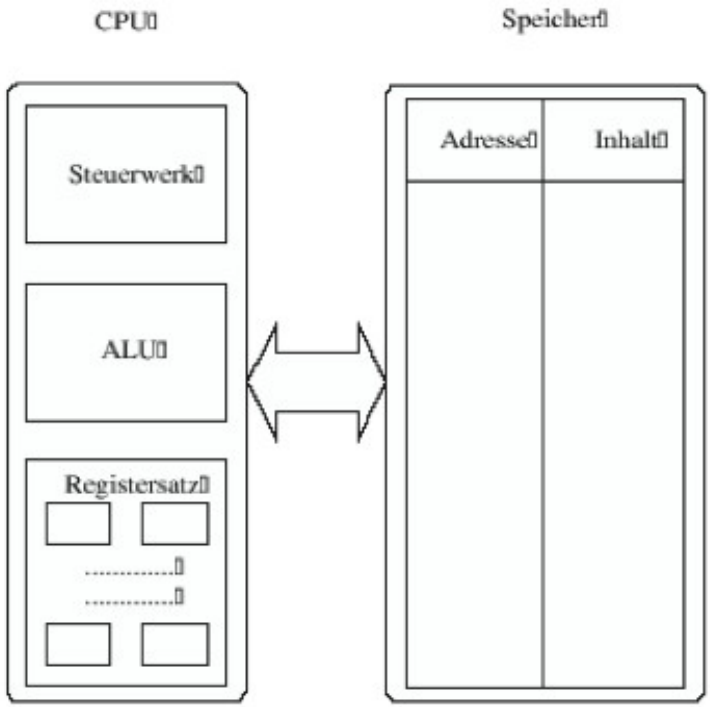
Computer: von Neumann-Architektur

Erinnerung: Programme und Daten im Hauptspeicher

Programme als Daten:

- Quelltext (*source code*):
  - Text in einer Programmiersprache wie z.B. Java
  - Binärcodierung als Folge von Symbolen
- Objektcode (*object code*):
  - Befehlsfolge in spezifischer Sprache eines Prozessortypes
  - Binärcodierung abhängig vom Prozessortyp

Computer: Prozessor und Programmausführung



Prozessor (CPU) besteht aus:

- Steuerwerk
- arithmetisch-logische Einheit (ALU)
- Register

Computer: Prozessor und Programmausführung

- Steuerwerk holt Befehle aus Speicher und interpretiert sie
- Befehle liegen in Maschinensprache vor (Objektcode)
- Verwendung spezieller Register: Befehlszähler und Befehlsregister
- Befehlszähler enthält Adresse der Speicherzelle, in der ein Befehl steht
- Befehlsregister enthält Objektcode eines Befehls

Beispiele für Befehlstypen (kein Objektcode!):

- `LOAD`: Lade Datum oder Inhalt einer Speicherzelle in ein Register
- `STORE`: Schreibe Datum aus Register in Speicherzelle
- Verknüpfung von Registern (z.B. `ADD`), Ergebnis wieder in Register
- `JUMP`: Springe zu einer Befehlsadresse im Speicher
- `CONDITIONAL JUMP`: Springe zu einer Befehlsadresse im Speicher, falls Inhalt eines bestimmten Registers gleich 0 ist

Computer: Fundamentaler Instruktionszyklus

1. Fetch: Hole Befehl, dessen Adresse im Befehlszähler steht, aus dem Speicher in das Befehlsregister
2. Increment: Erhöhe Befehlszähler um 1, damit er auf nächste Instruktion zeigt
3. Decode: Decodiere Instruktion, damit klar wird, was zu tun ist
4. Fetch operands: Falls nötig, hole Operanden aus den im Befehl angegebenen Speicherzellen
5. Execute: Führe den Befehl aus (ggf. mit Hilfe der ALU); bei einem Sprung (`JUMP`/`CONDITIONAL JUMP`) wird neuer Wert in das Befehlsregister geschrieben
6. Loop: Gehe wieder zum ersten Schritt

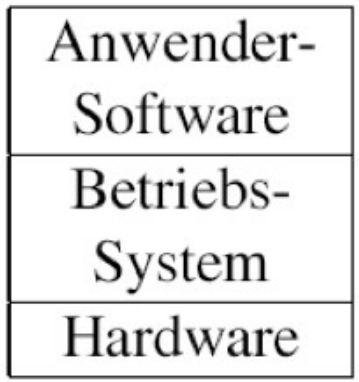
Für einfache Befehle pro Schritt ein Takt, für komplexe i.A. mehrere Takte

Computer: Software

Erinnerung:

Software-Komponenten eines Computers:

- Anwendersoftware (*application software*)
- Betriebssystem (*operating system*)



Im Folgenden: Verfeinerung der Komponentenstruktur

Computer: Software

Architektonik hochorganisierter (qualitativ komplexer) Systeme:

- Abstraktionsschichten (Hierarchien)
- Modularisierung (Heterarchien)
- (informationelle) Verbindung zwischen Ebenen bzw. Modulen über Schnittstelle (*interface*)

Informationsverarbeitung in Computersystemen: Abstraktionsschichten

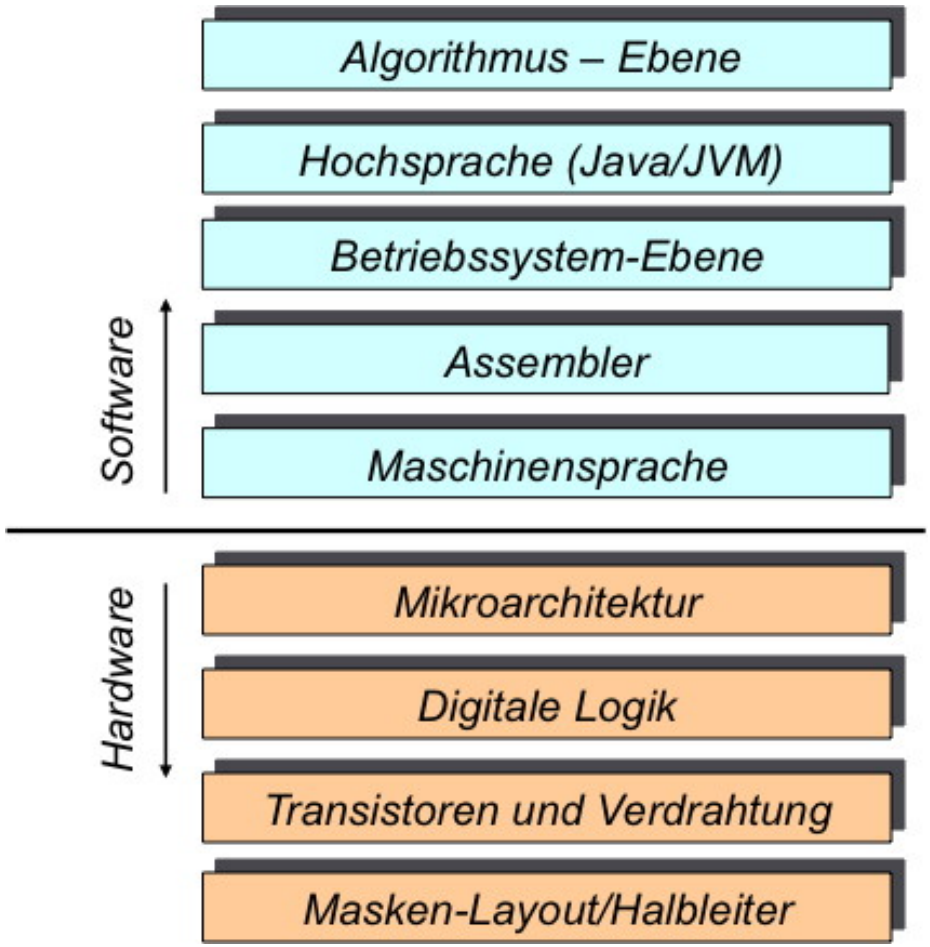
- jede Schicht definiert „Maschine“
- Maschine unterstützt Schnittstelle für darüber liegende Maschine
- Maschine benutzt Schnittstelle der darunter liegenden Maschine
- unterste Schichten in Hardware realisiert
- virtuelle Maschine: in Software realisierte Schicht

Computer: Software

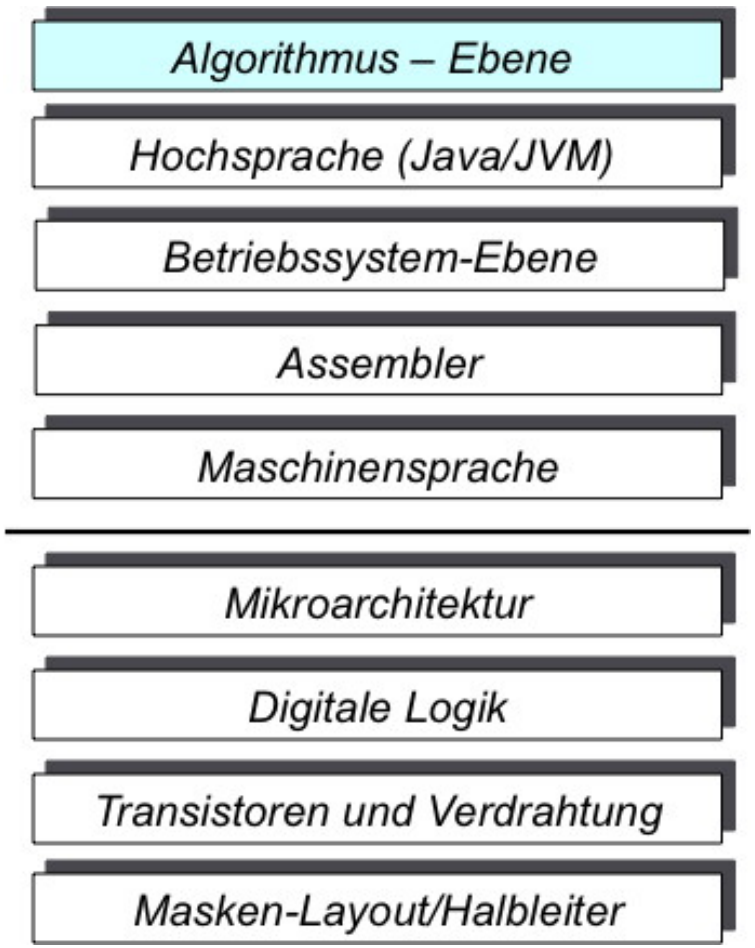
Grundformen des Übergangs zwischen Schicht n und Schicht $n-1$:

- **Interpretation**
  - Befehlsfolge (Programm) auf Schicht n wird auf Schicht $n-1$ als Datenfolge angesehen
  - Schicht $n-1$ interpretiert Daten nacheinander und führt die entsprechende Aktionen (Befehlssequenzen) aus
- **Übersetzung** (*compilation*)
  - Befehlsfolge (Programm) auf Schicht n wird zunächst vollständig von Compiler in (funktions-)äquivalente Befehlsfolge (Programm) auf Schicht $n-1$ übersetzt
  - Schicht $n-1$ führt Programm nach Übersetzung aus

Abstraktionsschichten: Überblick



Abstraktionsschichten: Algorithmus



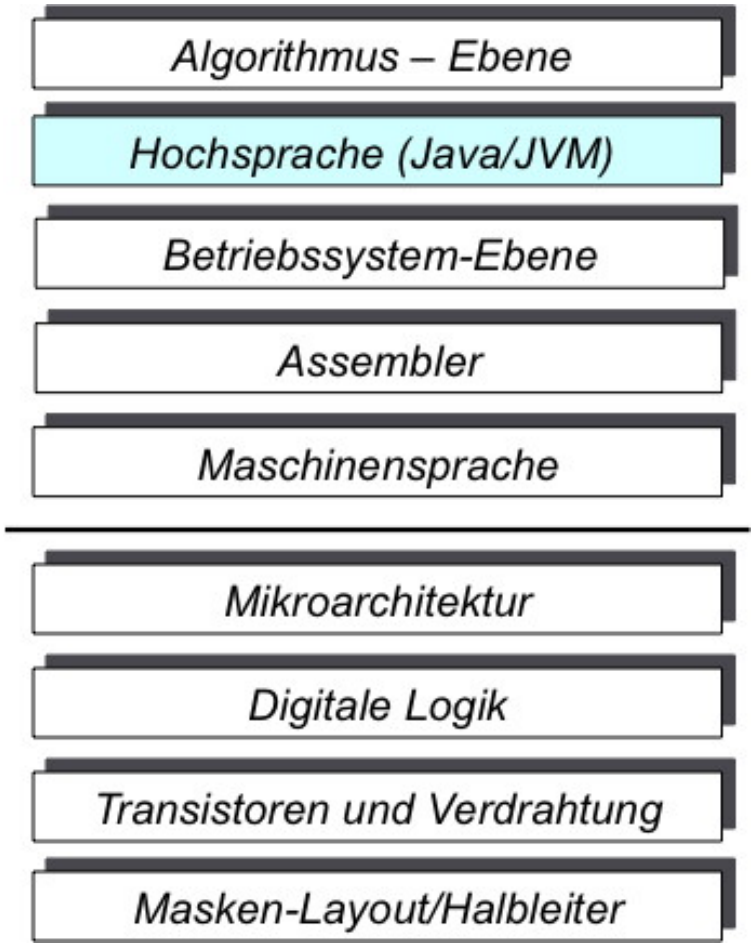
Skalarprodukt (Mathematik)

$$\vec{a}^T \cdot \vec{b}$$

$$= \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

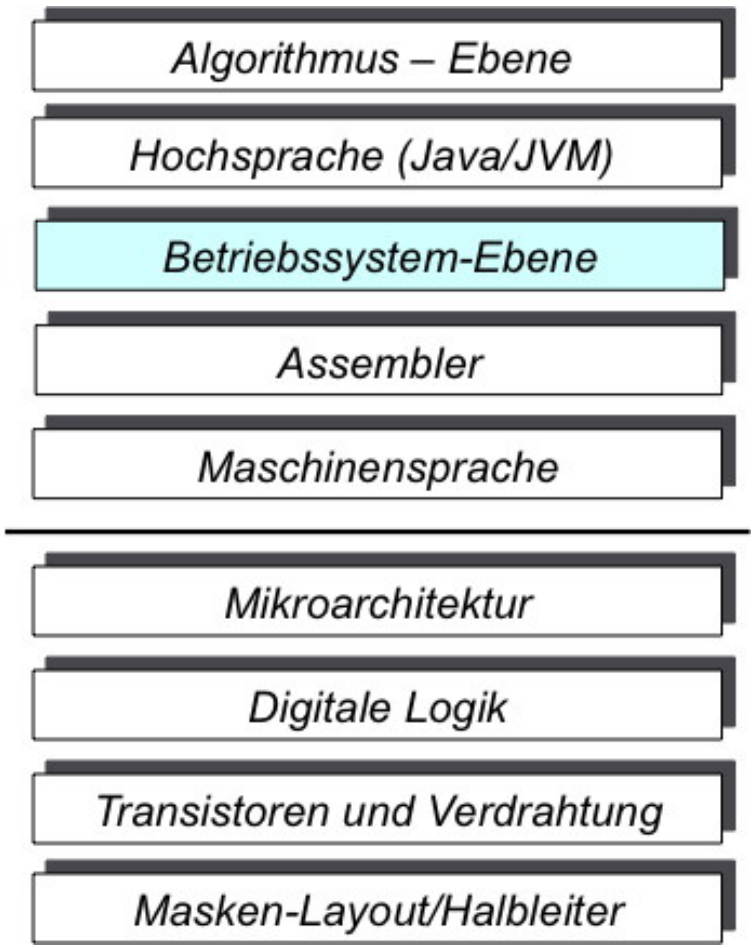
$$= \sum_{i=1}^{4} a_i \cdot b_i$$

Abstraktionsschichten: Hochsprache



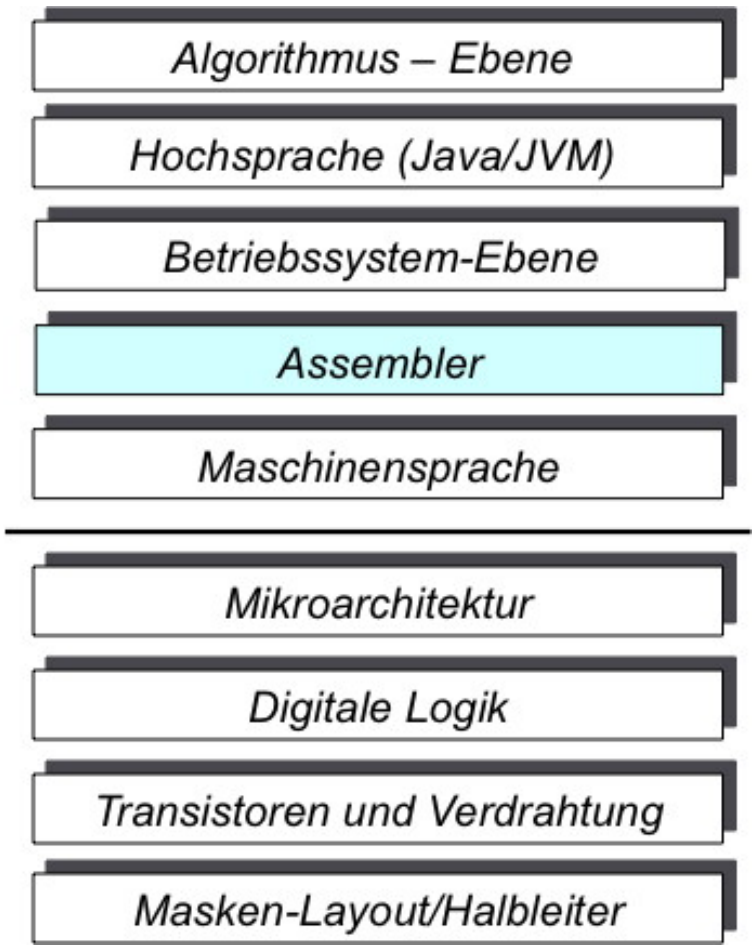

```
public static void main()
{
    int w, x;
    int n = 4;          /* Dimension */
    int[] a = {1,2,3,4};
    int[] b = {10,20,30,40};

    w = n;
    x = 0;
    while(w>0){
       w = w - 1;
       x = x + a[w] * b[w];
    }
          /* in x steht der Wert des
                Skalarproduktes */
}
```

Abstraktionsschichten: Betriebssysteme



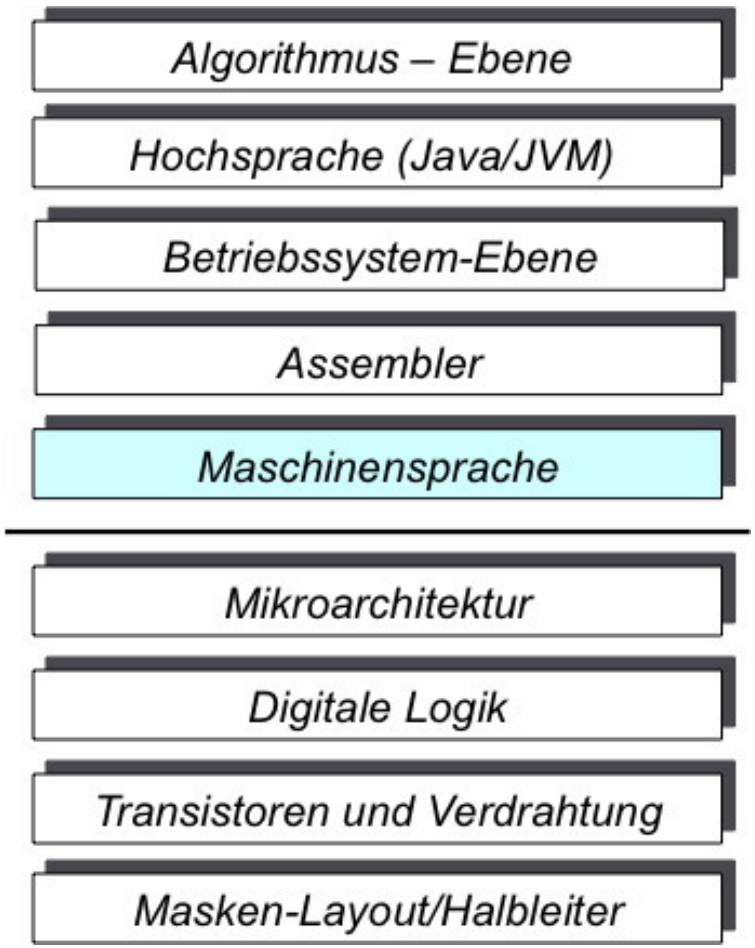
- Verwaltung und Zuweisung von Ressourcen
- Laden des Programms
- Speicherverwaltung
- Zugriff auf Festplatte, Drucker, etc.
- Multi-Tasking: Zuteilung der CPU
- ...

Abstraktionsschichten: Assembler



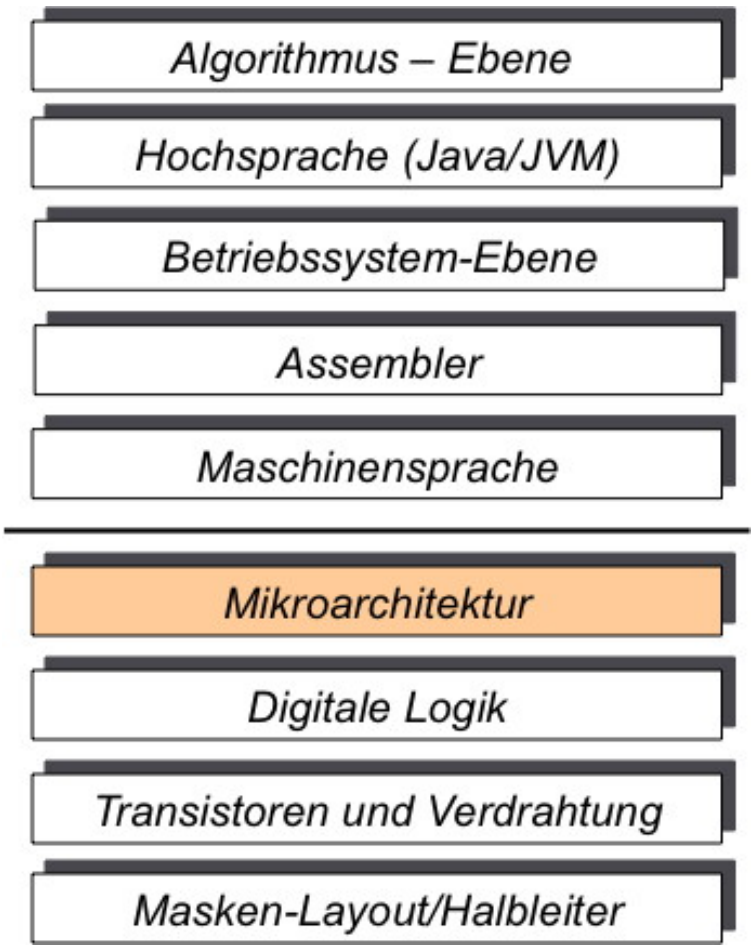

```
            LOC     Data_Segment
            GREG
N           OCTA    4               Vektordimension
ADR_A1      OCTA    1               a1
            OCTA    2               a2
            OCTA    3               a3
            OCTA    4               a4
ADR_B1      OCTA    10              b1
            OCTA    20              b2
            OCTA    30              b3
            OCTA    40              b4
u           IS      $1              Parameter 1
v           IS      $2              Parameter 2
w           IS      $3              Parameter 3
x           IS      $4              Ergebnis
y           IS      $5              Zwischenberechnung
z           IS      $6              Zwischenberechnung
            LOC     #100
            GREG
Main        SETL    x,0             Initialisierung
            LDA     u,ADR_A1        Ref. a1 in u
            LDA     v,ADR_B1        Ref. b1 in v
            LDO     w,N             Lade Vektordimension
            MUL     w,w,8           8 Byte Doppelwortlänge
Start       BZ      w,Ende          wenn fertig Ende
            SUB     w,w,8           w=w-8
            LDO     y,u,w           y=⟨u+w⟩
            LDO     z,v,w           z=⟨v+w⟩
            MUL     y,y,z           y=⟨u+w⟩ * ⟨v+w⟩
            ADD     x,x,y           x=x + ⟨u+w⟩ * ⟨v+w⟩
            JMP     Start
Ende        TRAP    0,Halt,0        Ergebnis in x
```

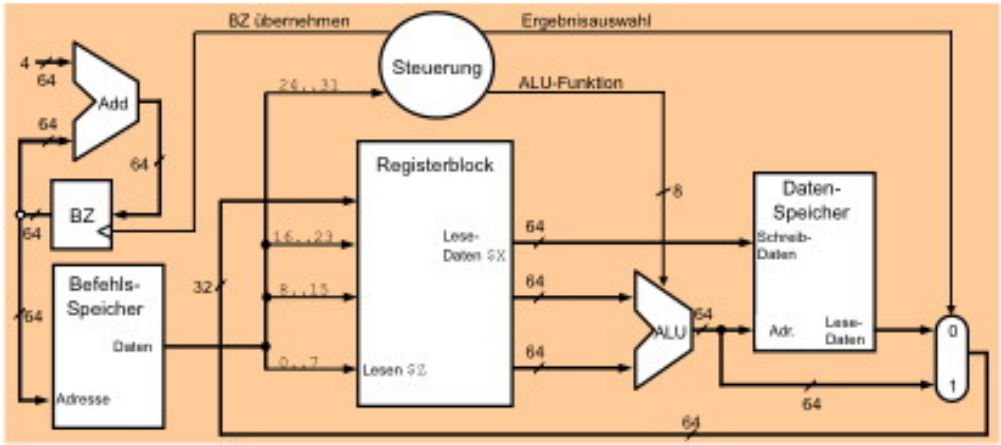
Abstraktionsschichten: Maschinensprache



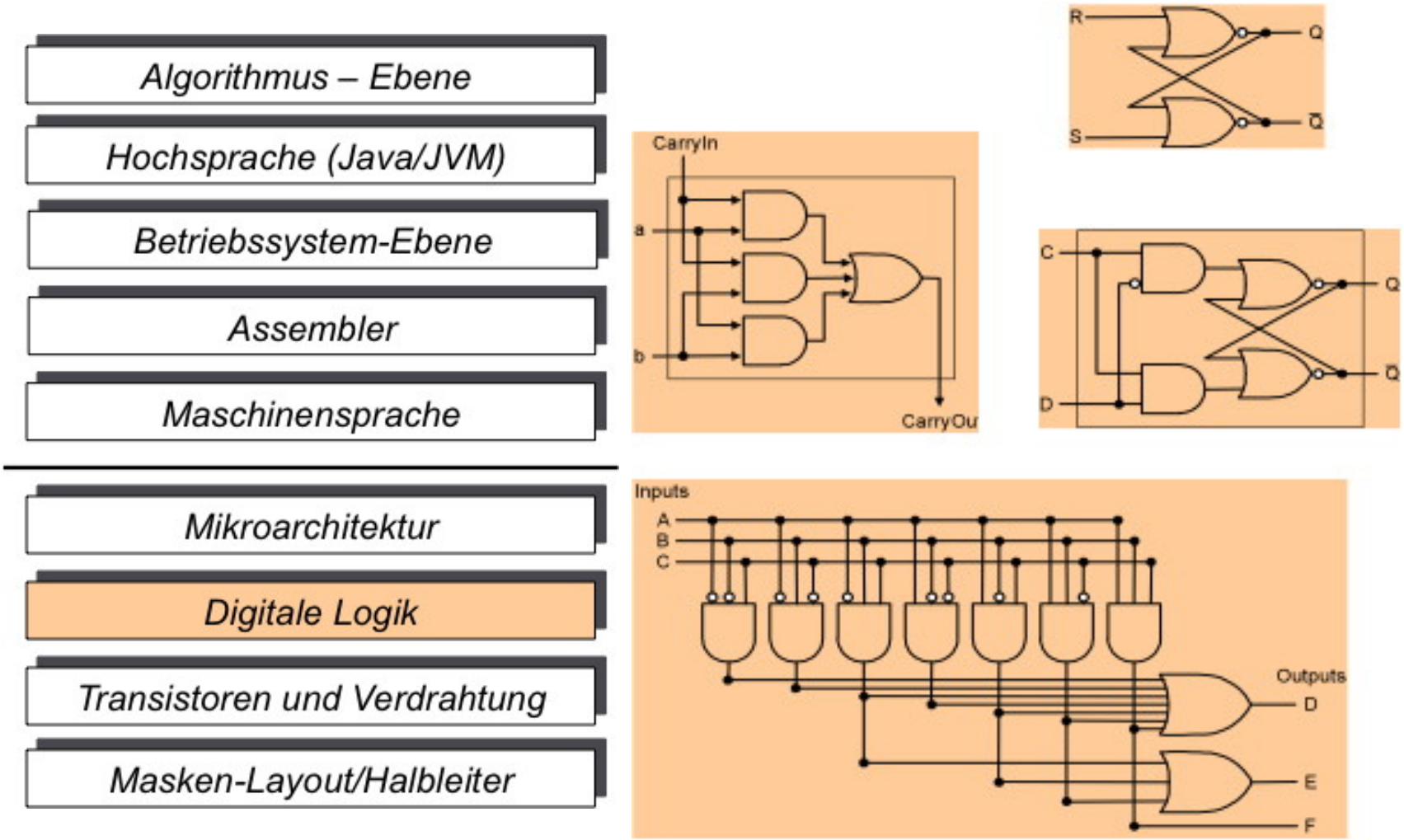

```
0x0000000000000100:    e3040000
0x0000000000000104:    2301fe08
0x0000000000000108:    2302fe28
0x000000000000010c:    8d03fe00
0x0000000000000110:    19030308
0x0000000000000114:    42030007
0x0000000000000118:    25030308
0x000000000000011c:    8c050103
0x0000000000000120:    8c060203
0x0000000000000124:    18050506
0x0000000000000128:    20040405
0x000000000000012c:    f1fffffa
0x0000000000000130:    00000000
.....                  ...
0x2000000000000000:    00000004
0x2000000000000004:    00000001
0x2000000000000008:    00000002
0x200000000000000c:    00000003
0x2000000000000010:    00000004
0x2000000000000014:    0000000a
0x2000000000000018:    00000014
0x200000000000001c:    0000001e
0x2000000000000020:    00000028
```

Abstraktionsschichten: Mikroarchitektur

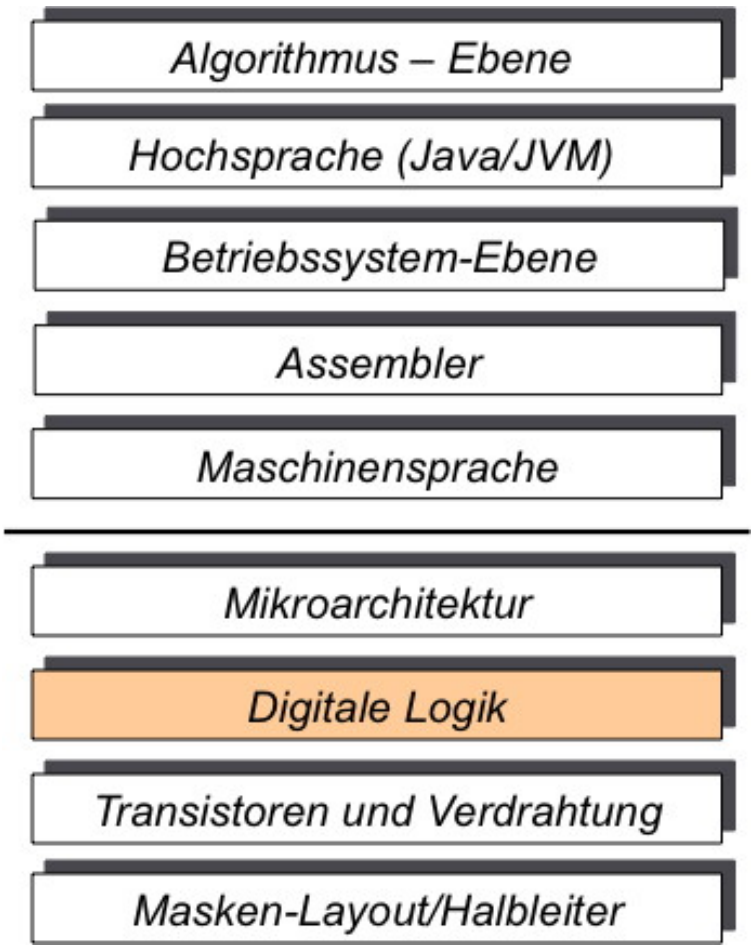


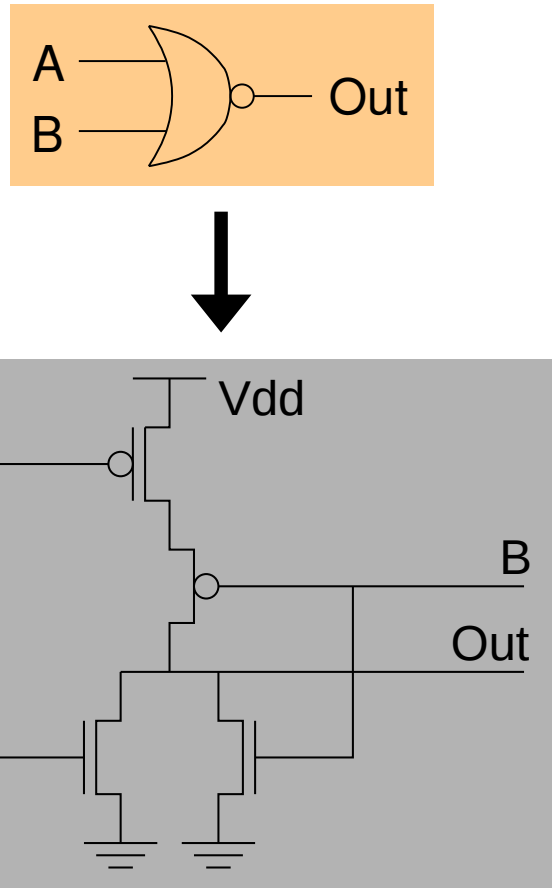


Abstraktionsschichten: Digitale Logik

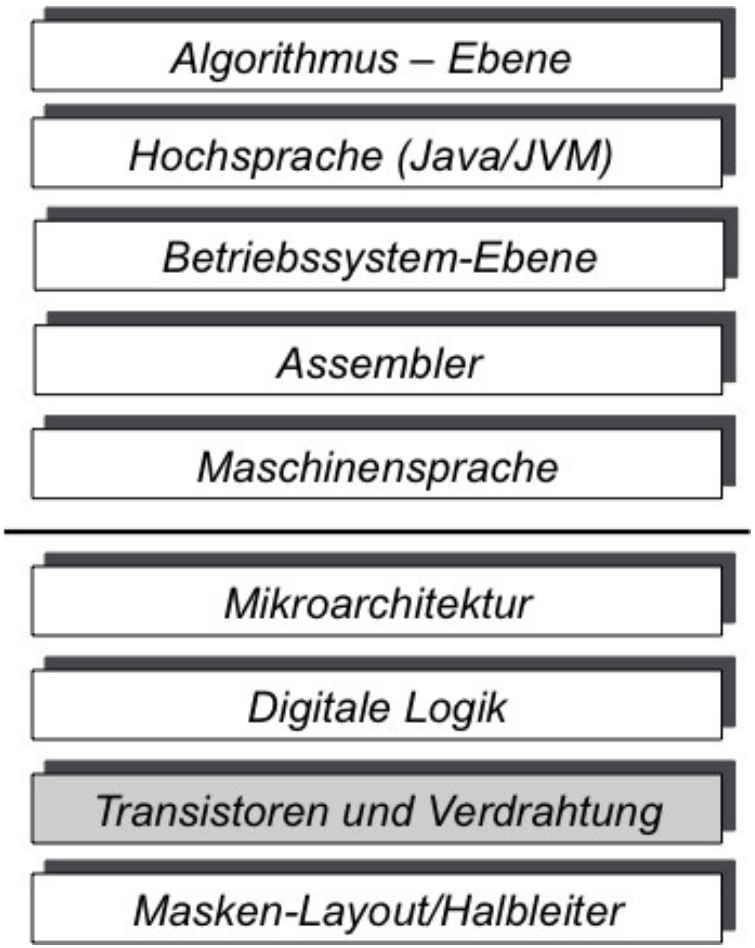


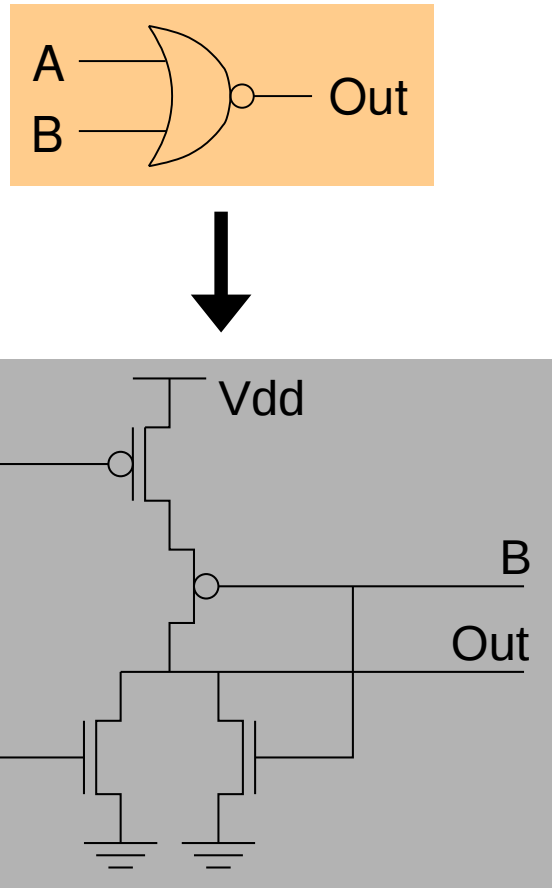
Abstraktionsschichten: Digitale Logik



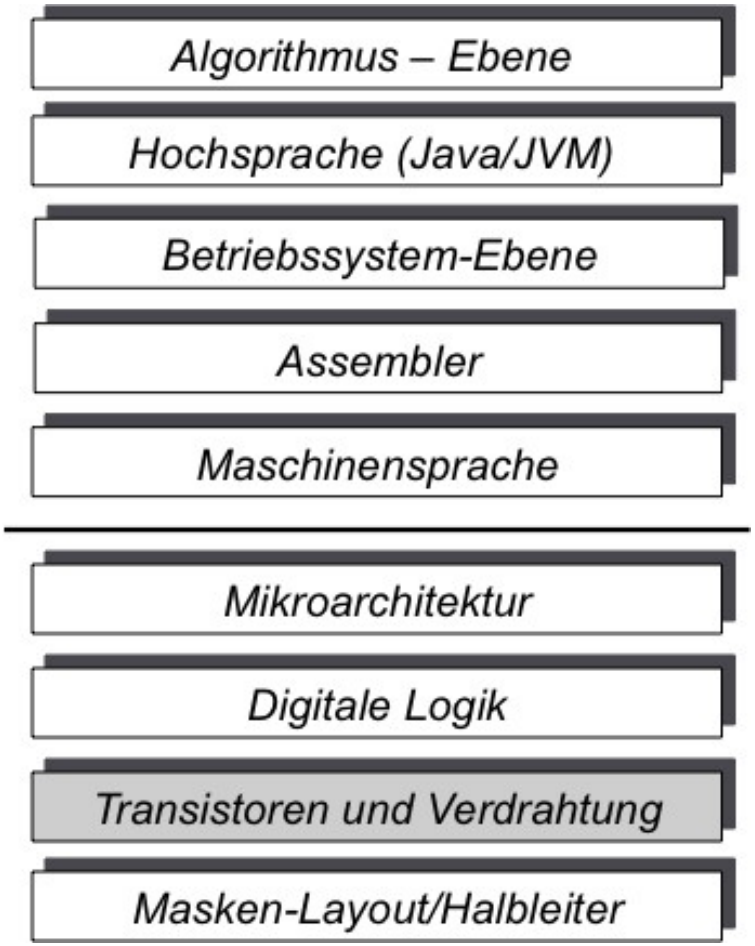


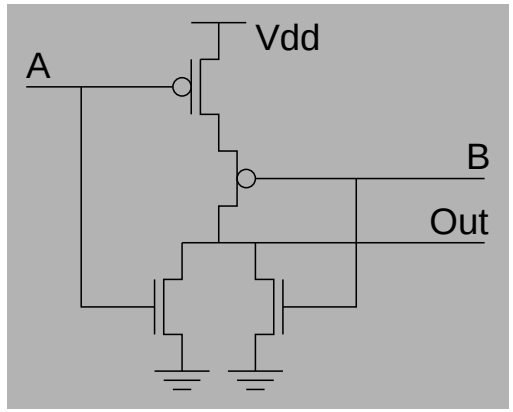
Abstraktionsschichten: Transistoren und Verdrahtung

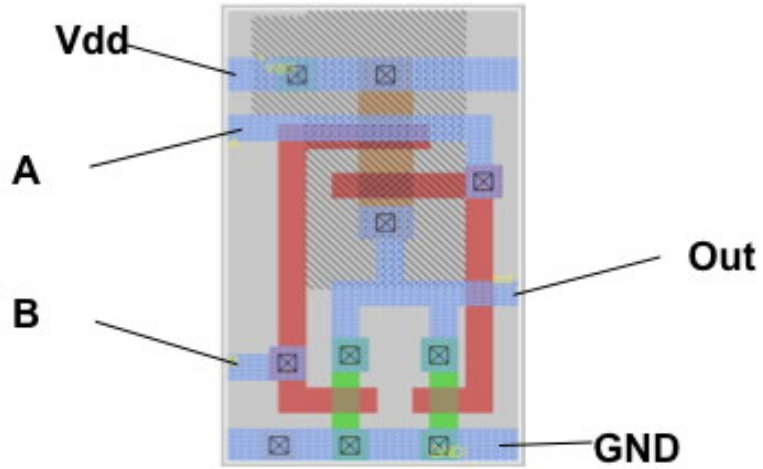




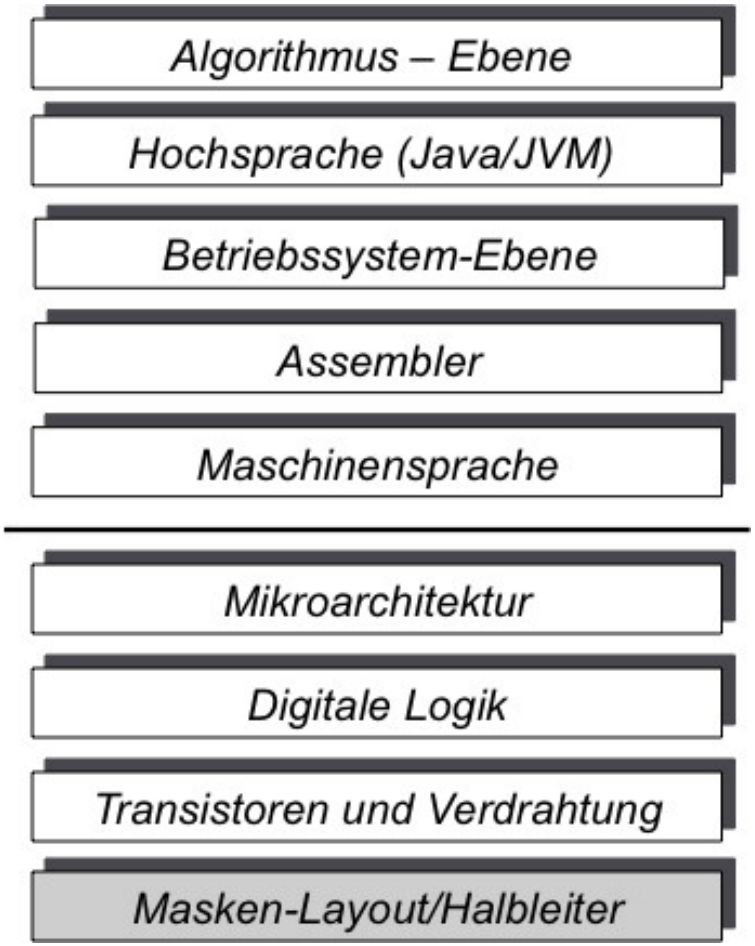
Abstraktionsschichten: Transistoren und Verdrahtung

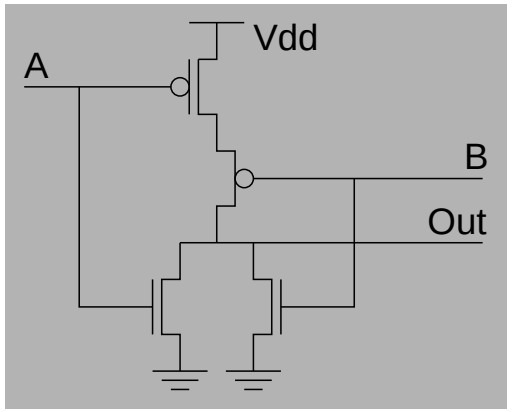


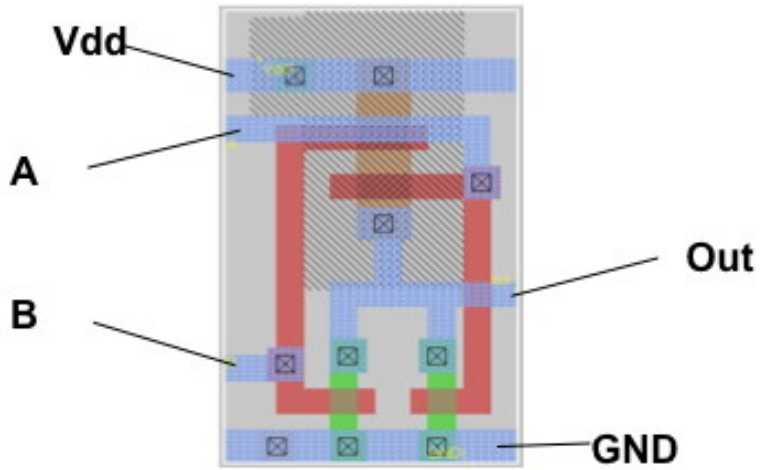




Abstraktionsschichten: Masken-Layout/Halbleiter







Abstraktionsschichten: Hochsprache

Frage: Werden Java-Programme übersetzt oder interpretiert?

Antwort: Beides!

- Java unterteilt Hochsprachenschicht in zwei Schichten
- Java-Programme werden in speziellen Byte-Code übersetzt
- Byte-Code wird von JVM (*Java virtual machine*) interpretiert
- JVM und Systembibliotheken pro Rechnertyp einmal entwickelt

Vorteil:

- Portabilität der Software (Plattformunabhängigkeit, Ladbarkeit über Internet, eingebettete Systeme)

Nachteil:

- Effizienzverlust durch Interpretation (partiell behandelbar: JIT)