

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: Dienstag, 16:30-18:00 Uhr, o.n.V.

Wintersemester 2009/2010

Informatik =

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

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

zentrale Gegenstände der Informatik:

- Information
- Algorithmen (\simeq Systematik der Verarbeitung)
- Computer (\simeq Rechenanlagen)

✓
demnächst
heute

Grundbestandteile eines Computers:

- **Hardware**

- ist fest gegeben
- kann angefasst werden
- ist unveränderlich (bis auf Austausch von Komponenten)

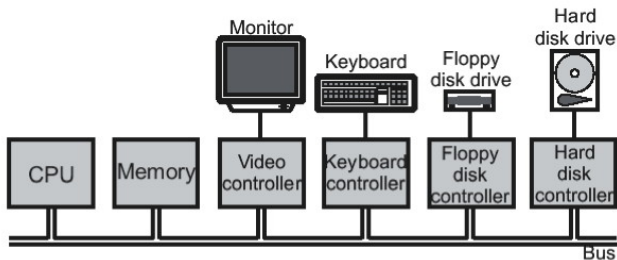
- **Software**

- besteht aus gespeicherten Programmen, die durch Hardware ausgeführt werden
- ist unsichtbar
- ist sehr leicht zu ändern (durch Änderung magnetischer, optischer oder elektrischer Zustände)

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



- 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*)

Software-Komponenten eines Computers:

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

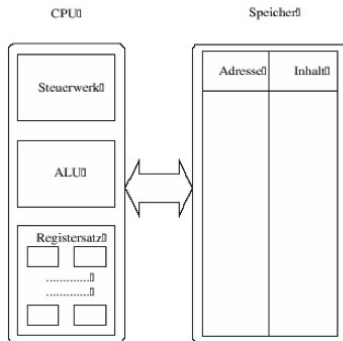
Anwender- Software
Betriebs- System
Hardware

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 adressierbare 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

Größe des digitalen Universums (2007): 281 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?

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 $0, 1, \dots, \beta - 1$):

- schreibe natürliche Zahl z mit n Stellen als Polynom (mit $0 \leq z_i < \beta$):

$$z = z_{n-1} \cdot \beta^{n-1} + z_{n-2} \cdot \beta^{n-2} + \dots + z_1 \cdot \beta^1 + z_0 \cdot \beta^0 =_{\text{def}} \sum_{i=0}^{n-1} z_i \cdot \beta^i$$

- $2009 = 2 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 9 \cdot 10^0$ für $\beta = 10$

wichtige Zahlsysteme:

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

- $23_{10} = 10111_2 = 27_8 = 17_{16}$

- $15_{10} = 1111_2 = 17_8$

Binärcodierung von natürlichen Zahlen

Binärdarstellung von 2009:

Restzahl	Zweierpotenz	Exponent	Restzahl \geq Zweierpotenz?
2009	1024	10	1
985	512	9	1
473	256	8	1
217	128	7	1
89	64	6	1
25	32	5	0
25	16	4	1
9	8	3	1
1	4	2	0
1	2	1	0
1	1	0	1

Damit gilt: $2009_{10} = 11111011001_2$

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

Binärcodierung von natürlichen Zahlen

Rechnen mit Binärdarstellungen:

- Addition:

$$\begin{array}{rcl} 23_{10} & = & 10111_2 \\ 15_{10} & = & 1111_2 \\ \hline 38_{10} & = & 100110_2 \end{array}$$

- Multiplikation:

$$\begin{array}{rcl} 7_{10} \cdot 11_{10} & = & 111_2 \cdot 1011_2 \\ \hline 70_{10} & & 111000_2 \\ 7_{10} & & 1110_2 \\ & & 111_2 \\ \hline 77_{10} & = & 1001101_2 \end{array}$$

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: $n \equiv r \pmod{k}$ (lies: n ist kongruent r modulo k)

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

Exkurs: Modulo-Arithmetik

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

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

Rechenregel für Modulo-Funktion:

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

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

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

$$\text{mod}(11 - 26, 9) = \text{mod}(-15, 9) = 3$$

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

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

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

$$\text{mod}(11 \cdot (-26), 9) = \text{mod}(-286, 9) = ?$$

Binärcodierung von natürlichen Zahlen

(binäre) Arithmetik fixer Länge n :

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

$$2^n = 1\underbrace{0\dots0}_n{}_2$$

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

Für $n = 4$ gilt $01000_2 + 01001_2 \equiv 10001_2 \pmod{10000_2}$,

Konversion:

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

Im Folgenden:

- Dezimal-Binär-Konversion
- (Binär-Dezimal-Konversion)

Dezimal-Binär-Konversion

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

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

wobei d_i Binärdarstellung der Ziffern z_i

Konvertiere 2009_{10} in Binärdarstellung:

$$\begin{aligned}2009_{10} &= 2 \cdot 1000 + 0 \cdot 100 + 0 \cdot 10 + 9 \cdot 1 \\ &= ((2 \cdot 10 + 0) \cdot 10 + 0) \cdot 10 + 9 \\ &= ((10_2 \cdot 1010_2 + 0_2) \cdot 1010_2 + 0_2) \cdot 1010_2 + 1001_2 \\ &= (10100_2 \cdot 1010_2 + 0_2) \cdot 1010_2 + 1001_2 \\ &= 11001000_2 \cdot 1010_2 + 1001_2 \\ &= 11111011001_2\end{aligned}$$

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 $i = n, n-1, \dots, 1$
- h_0 ist die Binärdarstellung

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):

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

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

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

$$\begin{aligned} &\{-(2^n - 2^{n-1}), \dots, -(2^n - (2^n - 1))\} \\ &= \{-2^{n-1}, \dots, -1\} \end{aligned}$$

- Verträglichkeit mit Arithmetik:

$$\begin{aligned} x - y &\equiv x + (2^n - y) \equiv x + \bar{y} \pmod{2^n} \\ \bar{0} &\equiv 2^n - 0 \equiv 0 \pmod{2^n} \end{aligned}$$

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 $\{0, 1, \dots, 7\}$ zusammengefasst

$$2009_{10} = 11111011001_2 = \underbrace{011}_3 \underbrace{111}_7 \underbrace{011}_3 \underbrace{001}_1 = 3731_8$$

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

$$2009_{10} = 11111011001_2 = \underbrace{0111}_7 \underbrace{1101}_d \underbrace{1001}_9 = 7d9_{16}$$

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 (<i>not a number</i>)

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

Wie ist Mantissenwert einer float-Zahl zu interpretieren?

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

- (möglicherweise unendliche) Folge $0, b_1 b_2 \dots_2$ von Binärziffern b_i mit

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

- nicht immer eindeutig, z. B. $\frac{1}{2} = 0,1_2$ und $\frac{1}{2} = 0,011111\dots_2$
- 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 $0,0001100110011\overline{0011}_2 = 0,0\overline{0011}_2$ binäre Expansion einer reellen Zahl z :

$$z = 0,0\overline{0011}_2, \quad 2^4 \cdot z = 1,1\overline{0011}_2$$

Rechte Gleichung – linke Gleichung ergibt:

$$15 \cdot z = (2^4 - 1) \cdot z = 1,1\overline{0011}_2 - 0,0\overline{0011}_2 = 1,1_2 = 2^0 + 2^{-1} = \frac{3}{2}$$

Somit gilt $z = \frac{3}{2 \cdot 15} = \frac{1}{10} = 10^{-1} = 0,1_{10}$.

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

- normalisierte Gleitkommazahl: $1,100110011001100110011\overline{0011}_2 \cdot 2^{-4}$
- Vorzeichen $v = 0$
- Exponent $e = 123_{10} = 01111011_2$ (wegen $e - 127 = -4$)
- Mantisse kann auf- oder abgerundet werden (nur 23 Bit):

$$0\mathbf{01111011}10011001100110011001100_{\text{float}} \approx 0,09999999403954_{10}$$

$$0\mathbf{01111011}10011001100110011001101_{\text{float}} \approx 0,10000000149012_{10}$$

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

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 (Modellrechnung)

- interner Zähler für Zeit seit Aktivierung (1 Takt = 10 s)
- zur Berechnung physikalischer Parameter Zähler mit $\frac{1}{10}$ multipliziert
- Realisierung mittels 24-Bit-Gesamtlänge für Gleitkommaoperationen
- Konversion von $0,1_{10}$ in 24-Bit-Zahl mit Fehler $\approx 6 \cdot 10^{-8}$
- Rakete ca. 100 h aktiv, d.h. Fehler ca. $100 \cdot 60 \cdot 60 \cdot 6 \cdot 10^{-8} \text{s} = 0,0216 \text{s}$
- Scud-Rakete mit ca. $1.676 \text{ m} \cdot \text{s}^{-1}$, d.h. in $0,0216 \text{s}$ ungefähr 36 m

Quelle: GAO/IMTEC-92-26

Wie codiert man binär:

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

Binärcodierung von Texten

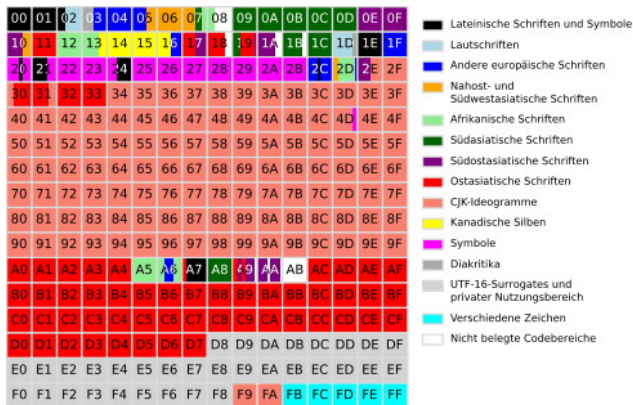
- Text ist Folge von Symbolen
- Symbole einzeln codieren
- n Bit codieren $\leq 2^n$ Buchstaben
- 26 Buchstaben mit 5 Bit ($2^4 < 26 < 2^5$)
- ISO 7 Bit (ASCII)
- ISO 8 Bit (erweitertes ASCII)
- ISO 16 Bit (UTF-16, Unicode)

oct	hex	0	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

- „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
- 2009 entspricht in ASCII-Codierung 32 30 30 39

Binärcodierung von Texten

Unicode: Zeichensätze werden in Blöcken (zu 8 Bit) zusammengefasst



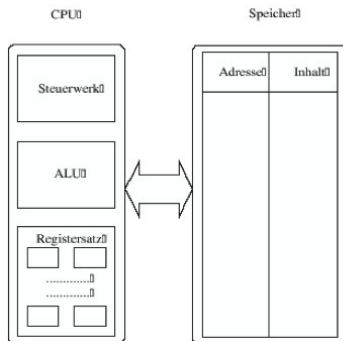
- Buchstabe Z entspricht in Unicode 005a (HTML: `a;`)
- Buchstabe Ю entspricht in Unicode 042e (HTML: `*e;`)

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

Erinnerung:

Software-Komponenten eines Computers:

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

Anwender- Software
Betriebs- System
Hardware

Im Folgenden: Verfeinerung der Komponentenstruktur

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

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

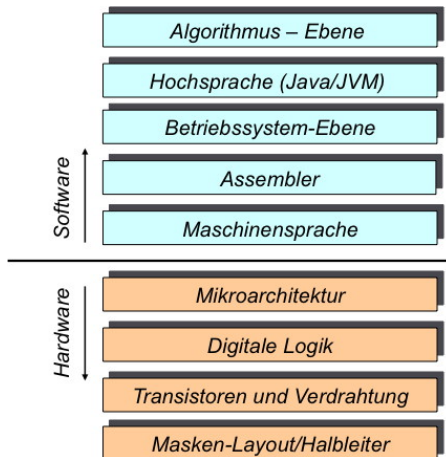
- **Interpretation**

- 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



©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Algorithmus

Algorithmus – Ebene

Hochsprache (Java/JVM)

Betriebssystem-Ebene

Assembler

Maschinensprache

Mikroarchitektur

Digitale Logik

Transistoren und Verdrahtung

Masken-Layout/Halbleiter

Skalarprodukt (Mathematik)

$$\begin{aligned}\vec{a}^T \cdot \vec{b} &= (a_1 \quad a_2 \quad a_3 \quad a_4) \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \\ &= \sum_{i=1}^4 a_i \cdot b_i\end{aligned}$$

©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Hochsprache

Algorithmus – Ebene

Hochsprache (Java/JVM)

Betriebssystem-Ebene

Assembler

Maschinensprache

Mikroarchitektur

Digitale Logik

Transistoren und Verdrahtung

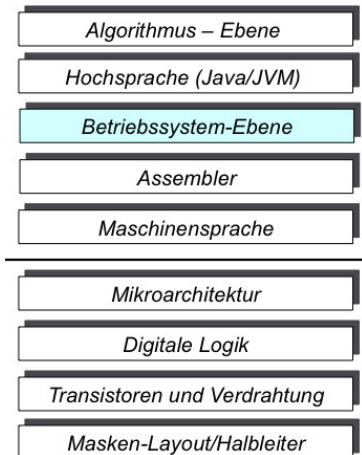
Masken-Layout/Halbleiter

```
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 */
}
```

©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Betriebssysteme



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

©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Assembler

Algorithmus – Ebene

Hochsprache (Java/JVM)

Betriebssystem-Ebene

Assembler

Maschinensprache

Mikroarchitektur

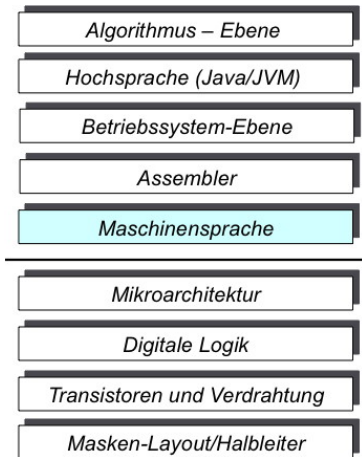
Digitale Logik

Transistoren und Verdrahtung

Masken-Layout/Halbleiter

	LOC	Data_Segment		
	GREG			
N	TETRA	4	Vektordimension	
ADR_A1	TETRA	1	a1	
	TETRA	2	a2	
	TETRA	3	a3	
	TETRA	4	a4	
ADR_B1	TETRA	10	b1	
	TETRA	20	b2	
	TETRA	30	b3	
	TETRA	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	
	LDT	w,N	Lade Vektordimension	
Start	MUL	w,w,4	4 Byte Wortlänge	
	BZ	w,Ende	wenn fertig Ende	
	SUB	w,w,4	w=w-4	
	LDT	y,u,w	y=<u+w>	
	LDT	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

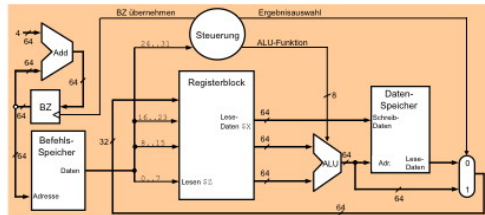
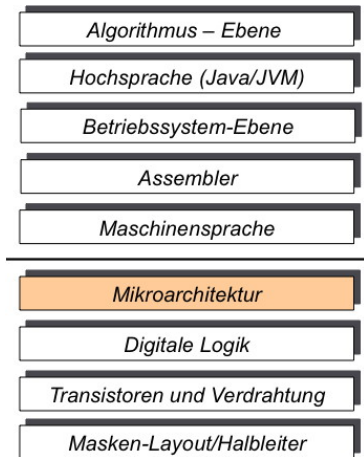
Abstraktionsschichten: Maschinensprache



```
0x00000000000000100: e3040000
0x00000000000000104: 2301fe08
0x00000000000000108: 2302fe28
0x0000000000000010c: 8d03fe00
0x00000000000000110: 19030308
0x00000000000000114: 42030007
0x00000000000000118: 25030308
0x0000000000000011c: 8c050103
0x00000000000000120: 8c060203
0x00000000000000124: 18050506
0x00000000000000128: 20040405
0x0000000000000012c: f1ffffffa
0x00000000000000130: 00000000
.....
0x20000000000000000: 00000004
0x20000000000000004: 00000001
0x20000000000000008: 00000002
0x2000000000000000c: 00000003
0x20000000000000010: 00000004
0x20000000000000014: 0000000a
0x20000000000000018: 00000014
0x2000000000000001c: 0000001e
0x20000000000000020: 00000028
```

©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Mikroarchitektur



©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Digitale Logik

Algorithmus – Ebene

Hochsprache (Java/JVM)

Betriebssystem-Ebene

Assembler

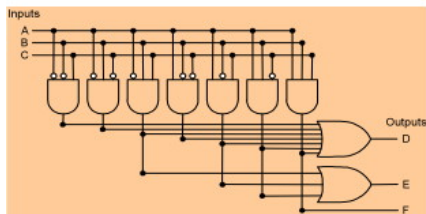
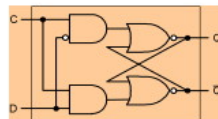
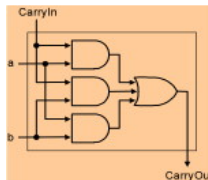
Maschinensprache

Mikroarchitektur

Digitale Logik

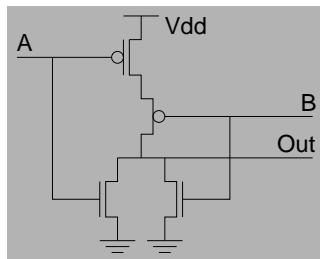
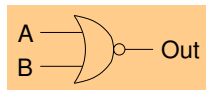
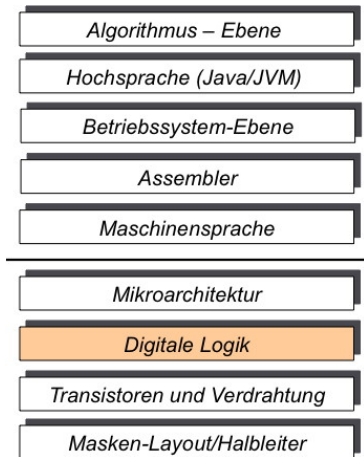
Transistoren und Verdrahtung

Masken-Layout/Halbleiter



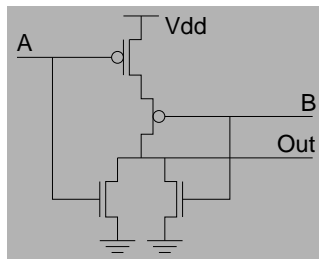
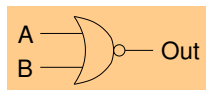
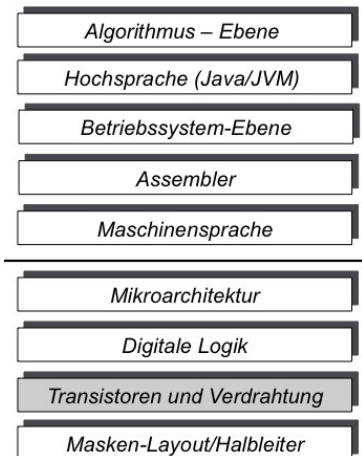
©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Digitale Logik



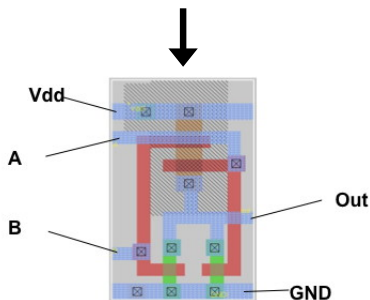
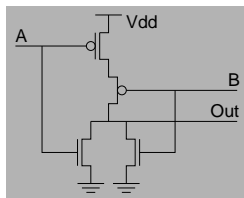
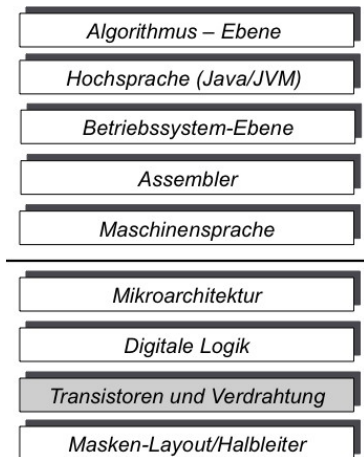
©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Transistoren und Verdrahtung



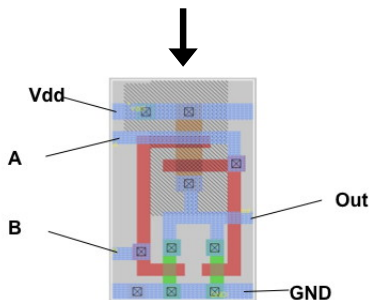
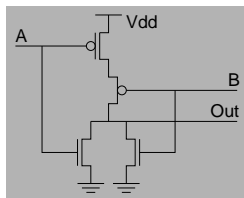
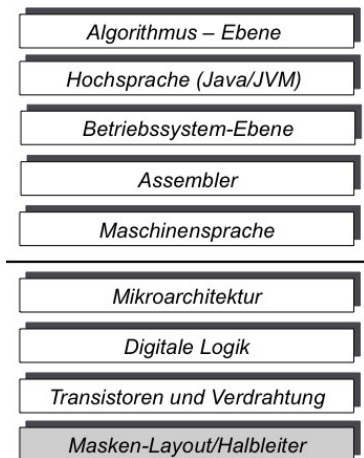
©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Transistoren und Verdrahtung



©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

Abstraktionsschichten: Masken-Layout/Halbleiter



©Klaus Diepolt, Lehrstuhl für Datenverarbeitung, Fakultät für Elektrotechnik und Informationstechnik, TU München

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

Damit:

Vorteil:

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

Nachteil:

- Effizienzverlust durch Interpretation (partiell behandelbar: JIT)