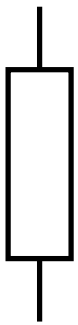
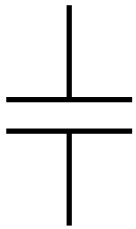
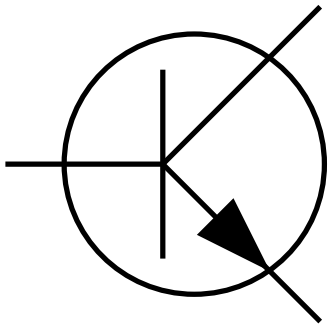
Computergrundlagen
Boolesche Logik, Zahlensysteme und Arithmetik

Axel Arnold

Institut für Computerphysik
Universität Stuttgart

Wintersemester 2010/11

Wie rechnet ein Computer?

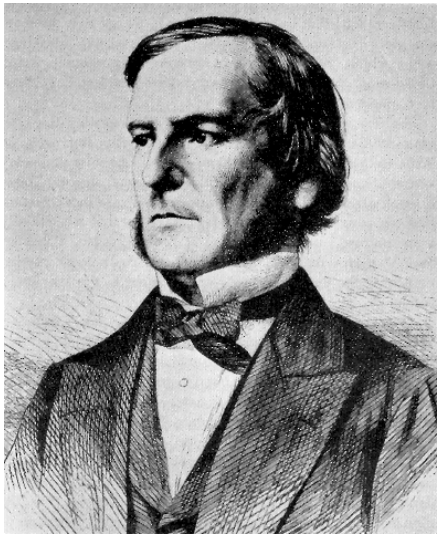




Ein Mikroprozessor

- ist ein Netz von Transistoren, Widerständen und Kondensatoren
- Leitungen kennen nur zwei Zustände: Spannung oder nicht
- Interpretation als ja/nein, 0/1, an/aus, ...
- Schaltungen entsprechen logischen Operationen („Spannung an Pin a und Spannung an Pin b $\Rightarrow$ Spannung am Ausgang“)

Wie kann ich damit rechnen?

Aussagenlogik



G. Boole,
1815 - 1864

- erlaubt formale Beweise über logische Aussagen
- Grundlage der Computerlogik
- kann mathematisch als Algebra formalisiert werden
  – als eine *boolesche Algebra*

Die zweielementige boolesche Algebra

Wir betrachten eine Menge

- mit zwei Elementen 1 („wahr“) und 0 („falsch“)
- mit zwei Verknüpfungen $\vee$ („oder“) und $\wedge$ („und“)
- mit einer einstelligen Verknüpfung $\neg$ („nicht“, Negation)

Ferner gilt für beliebige $a, b, c \in \{0, 1\}$:

1. $a \vee (b \vee c) = (a \vee b) \vee c$,
   $a \wedge (b \wedge c) = (a \wedge b) \wedge c$ (Assoziativität)
2. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$,
   $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ (Distributivität)
3. $a \vee b = b \vee a$, $a \wedge b = b \wedge a$ (Kommutativität)
4. $a \vee (a \wedge b) = a$, $a \wedge (a \vee b) = a$ (Adsorption)
5. $a \vee \neg a = 1$, $a \wedge \neg a = 0$ (Komplemente)

Abgeleitete Gesetze

- Neutralität:
  $a \vee 0 = a, \quad a \wedge 1 = a$
- Idempotenz:
  $a \vee a = a, \quad a \wedge a = a$
- Extremalgesetze:
  $a \vee 1 = 1, \quad a \wedge 0 = 0$
- Doppelnegation: $\neg(\neg a) = a$
- Dualität:
  $\neg 0 = 1, \quad \neg 1 = 0$
- De Morgansche Gesetze:
  $\neg(a \vee b) = \neg a \wedge \neg b, \quad \neg(a \wedge b) = \neg a \vee \neg b$

Die zweielementige boolesche Algebra entspricht genau der Aussagenlogik – und unserem Verständnis von Logik.

Zahlensysteme

- Wie kann ich mit nur zwei Elementen Zahlen darstellen?

Sei $B > 0$ eine natürliche Zahl. Dann kann jede natürliche Zahl z *eindeutig* dargestellt werden als

$$z = \sum_{i=0}^{\infty} B^n z_k,$$

wobei für alle k $0 \leq z_k < B$ und nur endliche viele $z_k \neq 0$.

Beispiel

$B = 10$ entspricht unserem Dezimalsystem:

$$1042 = 10^0 \cdot 2 + 10^1 \cdot 4 + 10^3 \cdot 1 = 1042\text{d}$$

$B = 8$ ergibt das Oktalsystem:

$$1042 = 8^0 \cdot 2 + 8^1 \cdot 2 + 8^3 \cdot 2 = 2022\text{o}$$

Binärsystem

- Wie kann ich mit nur zwei Elementen Zahlen darstellen?

Wir benutzen das *Binärsystem* mit $B = 2$ und Ziffern 0 und 1 (Bits)

Beispiele

$$42 = 2^5 + 2^3 + 2^1 = 101010\text{b}$$

- Umrechnung von Binär- auf Dezimalzahlen ist umständlich
- Oktal ist es einfach:

$$1042 = 2^{10} + 2^4 + 2^1 = \overset{2\,1\;\;4\,2\,1\;\;4\,2\,1\;\;4\,2\,1}{10.000.010.010\text{b}} = 2022\text{o}$$

- oder in *Hexadezimalzahlen* ($B = 16$, Ziffern 1–9, A–F):

$$\overset{4\,2\,1\;\;8\,4\,2\,1\;\;8\,4\,2\,1}{100.0001.0010\text{b}} = 812\text{h}, \quad 1010.1111.1111.1110\text{b} = \text{AFFEh}$$

Addieren/Subtrahieren im Binärsystem

Genau wie im Dezimalsystem:

	101010	(Summand *a*)
+	1111	(Summand *b*)
	1110	(Übertrag *c*)
=	111001	(Ergebnis *e*)

	101010	(1. Summand *a*)
-	1111	(2. Summand *b*)
	11111	(geborgt *c*)
=	11011	(Ergebnis *e*)

Betrachten wir die Ziffern als Wahrheitswerte, so gilt:

$$
\begin{aligned}
e_0 &= (a_0 \vee b_0) \wedge \neg(a_0 \wedge b_0) =: a_0 \oplus b_0 \\
c_1 &= a_0 \wedge b_0 \\
e_i &= (a_i \oplus b_i) \oplus c_i \\
c_{i+1} &= (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)
\end{aligned}
$$

Damit kann die Addition als Schleifen-Schaltkreis realisiert werden!

Komplementdarstellung negativer Zahlen

Was ist z.B. -5 = 0 - 5?

```
          0
-       101
    ...11111
-------------
=   ...111011
```

- im Prinzip unendlich viele führende 1er
- in der Praxis endliche Darstellung (z.B. 32 Bit)
- Bei n Bit-Darstellung wird $-z$ als $2^n - z$ dargestellt
- Bei 8 Bit z.B. ist $-5 = 256 - 5 = 251 = 11111011\text{b}$
- Addition und Subtraktion funktionieren ohne Änderung, solange module 2^n gerechnet wird

Multiplizieren im Binärsystem

```
      101010 * 1010
-----------------------
             0     ↓
    +   101010.    ↓
    +        0..   ↓
    + 101010...    ↓
-----------------------
 =    110100100
```

Binäre Multiplikation ist sehr einfach:

- Eine Zahl wird bitweise nach links geschoben (Multiplikation mit 2)
- ist das entsprechende Bit der andere Zahl gesetzt, wird die erste addiert, sonst nicht

Dividieren im Binärsystem

```
   101111   / 101   = 1001   (Ergebnis)
 - 101...              ↑
 ------------
 =    111
  -   0..               ↑
 ------------
  =   111
  -    0.                ↑
 ------------
   =  111
    - 101                  ↑
 ------------
    =  10   (Rest)
```

Binäre Division analog zur Multiplikation:

- Der Divisor wird zunächst ganz nach links geschoben
- Und dann bitweise wieder nach links
- Wenn der Rest größer als der Divisor ist, abziehen, und das entsprechende Bit des Ergebnisses setzen

Darstellung reeller Zahlen

$$\pm 1,23456789 \cdot 10^{\pm 123}$$

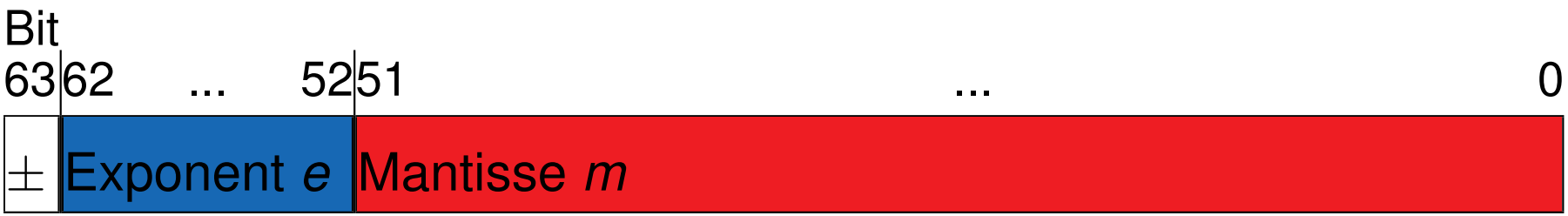
- Eine Fließkommazahl besteht aus:
  - Vorzeichen
  - Mantisse
  - (10er-)Exponent
- Stellen daher nur einen Teil der rationalen Zahlen exakt dar
- Alle anderen Zahlen werden angenähert
- Die Mantissenlänge bestimmt die Genauigkeit der Näherung

Binär entsprechend:

$$\pm 1,00111100000011100101 \cdot 2^{\pm 1111011}$$

IEEE-Fließkommazahlen

Speicherung einer 64-Bit-Fließkommazahl nach IEEE 754-Standard:



- Keine Komplementdarstellung von Mantisse oder Exponent
- Mantisse wird ohne führende Stelle gespeichert, die immer 1 ist
- Der 11-bittige Exponent e ist um 1023 verschoben gespeichert und nimmt Werte von -1022 ($e = 1$) bis 1023 ($e = 2046$) an
- Der Wert einer Zahl ist also:

$$\pm 1, m \cdot 2^{e-1023}$$

Spezielle Werte

Was ist mit den Exponenten -1023 ($e = 0$) und 1024 ($e = 2047$)?

Diese stellen spezielle Werte dar:

$e = 0$ und $m = 0$	0
$e = 0$ und $m \neq 0$	Zahlen der Form $\pm 0, m \cdot 2^{-126}$
$e = 2047$ und $m = 0$	$\pm\infty$
$e = 2047$ und $m \neq 0$	$\pm$NaN (not a number)

- Da die erste Stelle immer 1 ist, kann 0 nur so dargestellt werden
- $\pm\infty$ ergibt sich z.B. bei Berechnung von $\pm 1/0$
- $\pm$NaN ergibt sich z.B. bei Berechnung von $\sqrt{-1}$
- Python fängt NaNs mit Fehlern ab, nicht aber z.B. C

Subtraktion/Addition von Fließkommazahlen

Beispiele:

$$1,0 \cdot 10^{0} + 1,0 \cdot 10^{-5}$$
$$= 1,0 \cdot 10^{0} + 0,00001 \cdot 10^{0} = 1,00001 \cdot 10^{0}$$

$$1,000002 \cdot 10^{0} - 1,000001 \cdot 10^{0}$$
$$= 0,000001 \cdot 10^{0} = 1,0 \cdot 10^{-6}$$

- Verschieben der Mantisse der kleineren Zahl, bis beide denselben Exponenten haben
- Dabei gehen Stellen der kleineren Zahl verloren
- Dann gewöhnliche Addition/Subtraktion der Mantissen
- Schließlich den Exponenten verringern, bis die Mantisse keine führenden Nullen hat

Auslöschung

Ist $x < 2^{-l_m}$, wobei l_m die Bitlänge der Mantisse ist, so ist in Fliesskommaarithmetik

$$x \rightarrow x + 1 - 1 \rightarrow (x + 1) - 1 \rightarrow 1 - 1 \rightarrow 0$$

- *Auslöschung* der vorderen Stellen bei Subtraktion ungefähr gleich großer Zahlen führt zu fehlenden hinteren Stellen im Ergebnis
- Im Beispiel verschwindet die kleinere Zahl x komplett
- Ist $x \approx 2^{-l_m}$, verliert x fast alle signifikanten Stellen
- Formeln daher wenn möglich stets so umformen, dass keine Auslöschung passieren kann

Beispiel: quadratische Gleichung

- Lösen der quadratischen Gleichung $x^2 + ax + b = 0$:

$$x_\pm = -\frac{a}{2} \pm \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

- Bei kleinem b ist $\sqrt{\left(\frac{a}{2}\right)^2 - b} \approx \frac{a}{2}$
- $a > 0$: *Auslöschung* bei der größeren Nullstelle x_+
- $a < 0$: *Auslöschung* bei der kleineren Nullstelle x_-
- Ausweg: Es gilt

$$x_+ x_- = \left(-\frac{a}{2} + \sqrt{\left(\frac{a}{2}\right)^2 - b}\right)\left(-\frac{a}{2} - \sqrt{\left(\frac{a}{2}\right)^2 - b}\right) = b$$

- Berechne also nur die stabile Summe (x_- bei $a > 0$, sonst x_+), und die zweite Nullstelle durch $x_+ = b/x_-$ bzw. $x_- = b/x_+$

Multiplikation von Fließkommazahlen

Beispiel:

$$1,01 \cdot 10^{0} \times 1,01 \cdot 10^{5}$$
$$= 1,0101 \cdot 10^{0+5} = 1,0101 \cdot 10^{5}$$

- Multiplikation und Division sind unkritisch, es kann zu keiner Auslöschung kommmen
- Allerdings zu Über- oder Unterläufen des Exponenten
- In diesem Fall ist das Ergebnis ∞ bzw. 0
- Meist kann man dies aber durch geeignete Umformungen auch vermeiden, z.B. durch logarithmisches Rechnen

Fehlerquellen bei numerischen Rechnungen

- **Numerische Fehler**
  - Auslöschung
  - Rundungsfehler durch endliche Mantissenlänge
- **Algorithmische (Verfahrens-) Fehler**
  - Abschneidefehler: Unendliche Summen müssen durch endliche Summen ersetzt werden
  - Diskretisierungsfehler: Funktionsauswertung immer nur an endlich vielen Punkten
- **Modellierungsfehler**
  Das der Rechnung zugrundeliegende Modell erfasst wesentliche Details des Experiments nicht
- **Datenfehler**
  Genauigkeit der Anfangswerte (z.B. aus einem Experiment)

Beispiel: klassische Lennard-Jones-Simulation

- **Modellierungsfehler:**
  - entspricht keinem bekannten Atom/Molekül
  - vernachlässigt quantenmechanische Effekte
  - periodische Randbedingungen ersetzen unendliches Volumen
- **Algorithmische Fehler:**
  - Das Potential $U_{LJ}(r)$ wird für $r \gg 0$ rasch klein, daher betrachtet man nur Teilchenpaare mit kleiner Distanz
    - Fehler durch die weggelassenen Wechselwirkungen?
    - Wie klein kann man den Abschneideradius wählen?
  - Simulation mit diskretem Zeitschritt
    - Wie groß darf der Zeitschritt sein?
    - Wie lange muss man simulieren?

Was kann man tun?

- **Datenfehler**
  - Bessere Messverfahren entwickeln
  - Algorithmen wählen, die mit Störungen zurechtkommen – **Stabilität**
- **Numerische Fehler und algorithmische Fehler**
  - Algorithmen nutzen, die numerische Fehler minimieren
  - Algorithmen mit bekannten Fehlerabschätzungen einsetzen – **Konvergenz**
- **Modellierungsfehler**
  - Das Modell verfeinern
  - Dazu sind leistungsfähigere Algorithmen nötig – **Effizienz**

Wie kann ich Stabilität, Konvergenz und Effizienz eines Verfahrens beschreiben?

Landau-Symbole

Seien f, g zwei Funktionen.
Falls

$$\lim_{x\to a} \frac{|f(x)|}{|g(x)|} < \infty,$$

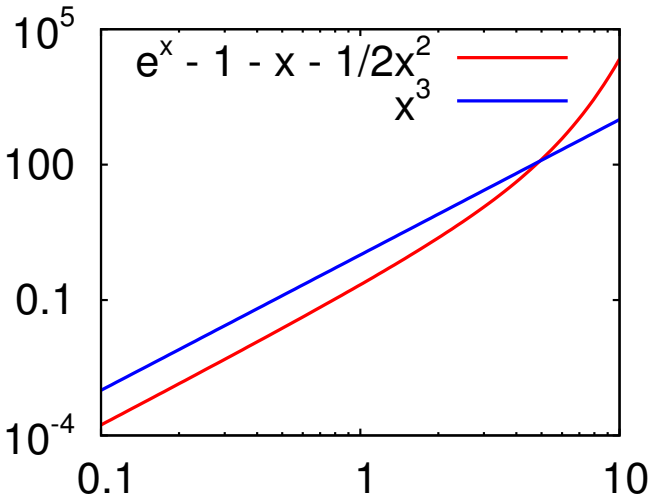
so schreiben wir kurz $f = \mathcal{O}_{x\to a}(g)$ („f ist von der Ordnung g“).

- Es gilt zum Beispiel

$$e^x - \left(1 + x + \frac{1}{2}x^2\right) = \mathcal{O}_{x\to 0}(x^3),$$

bzw.

$$e^x = 1 + x + \frac{1}{2}x^2 + \mathcal{O}_{x\to 0}(x^3).$$



- Bubble-Sort hat einen asymptotischen Aufwand von $\mathcal{O}(N^2)$ für Listenlängen $N \to \infty$.

Beispiel: Verlet-Integrator

Taylorentwicklung für die Position eines Teilchens gemäß der Newtonschen Gleichung:

$$x(t+\Delta t) = x(t) + \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)\Delta t^2 + \frac{1}{6}\dddot{x}(t)\Delta t^3 + \mathcal{O}(\Delta t^4)$$

und analog

$$x(t-\Delta t) = x(t) - \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)\Delta t^2 - \frac{1}{6}\dddot{x}(t)\Delta t^3 + \mathcal{O}(\Delta t^4)$$

Addition ergibt das *Verlet*-Verfahren zur numerischen Integration der Bewegungsgleichung:

$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + a(t)\Delta t^2 + \mathcal{O}_{\Delta t\to 0}(\Delta t^4)$$

- Fehler ist von der Ordnung $\mathcal{O}_{\Delta t\to 0}(\Delta t^4)$
- Mit beliebig kleinem Zeitschritt Δt wird das Verfahren exakt
- Man sagt, das Verfahren **konvergiert**

Eigenschaften von Algorithmen

Die wesentlichen Eigenschaften eines Algorithmus können mit Hilfe der Landau-Symbole charakterisiert werden:

- **Stabilität**: Wir betrachten den Fehler $\Delta f(x)$ in Abhängigkeit von der Genauigkeit Δx der Eingabedaten. Dann soll
$$\Delta f(x) = \mathcal{O}_{\Delta x \to 0}(\Delta x^n)$$
mit möglichst großem n sein.
- **Konvergenz**: Z.B. der Fehler als Funktion des Zeitschritts Δt. Dann soll
$$\Delta f(x) = \mathcal{O}_{\Delta t \to 0}(\Delta t^n)$$
sein mit möglichst großem n.
- **Effizienz**: Die Rechenzeit $T(N)$ z.B. bei N Teilchen. Dann soll
$$T(N) = \mathcal{O}_{N \to \infty}(N^n)$$
mit möglichst *kleinem* n sein.