Lektion 01

Was ist Informatik?

Herkunft und Geschichte des Fachs · Skript, Kapitel 1

Worum geht es in der Informatik?

Informatik ist weit mehr als das Schreiben von Programmen.

„*Informatik*“ = „Information“ + „Automatik“

Worum geht es in der Informatik?

Informatik ist weit mehr als das Schreiben von Programmen.

$$\text{„}\textit{Informatik}\text{“} = \text{„Information“} + \text{„Automatik“}$$

Die Wissenschaft von der **systematischen** . . .

- Darstellung
- Speicherung
- Übertragung
- Verarbeitung

. . . von **Informationen** mit Hilfe formaler Methoden und technischer Systeme.

Zwei Ebenen

Die Idee

- **Algorithmen**
- **Datenstrukturen**
- **Wie löst man ein Problem?**

Zwei Ebenen

Die Idee

- **Algorithmen**
- **Datenstrukturen**
- **Wie löst man ein Problem?**

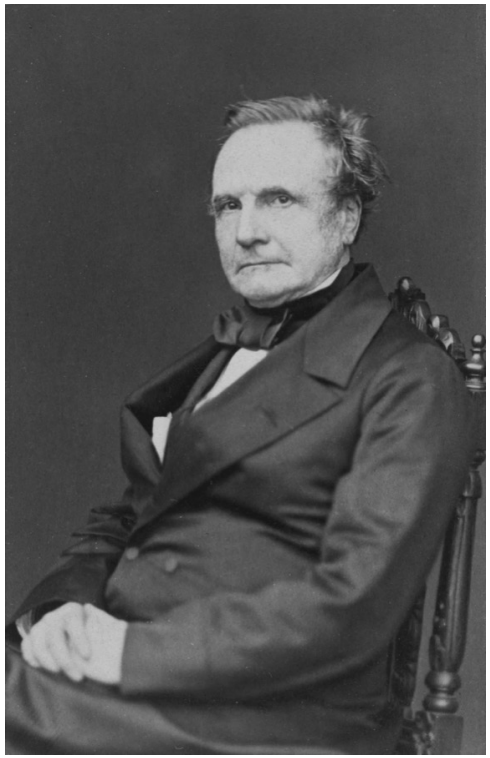
Die Umsetzung

- **Hardware, Schaltungen**
- **Netzwerke, Betriebssysteme**
- **Wie baut man die Lösung?**

Dieses Semester begleitet uns beides – von der Zahl bis zum fertigen Computer.

Sechs Köpfe, die das Fach geprägt haben

Was hat Informatik eigentlich mit diesen Menschen zu tun?



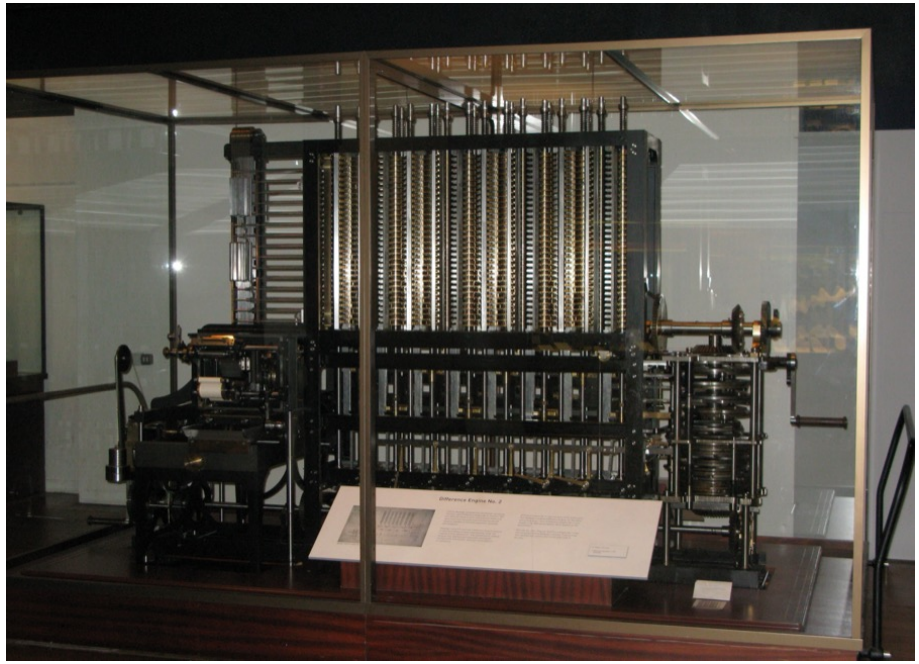
Charles Babbage

1791 – 1871, England

Entwarf die **Difference Engine** und die **Analytical Engine** – mechanische Maschinenideen, die als Vorläufer programmierbarer Computer gelten.

Als ein Abgeordneter ihn fragte, ob bei falscher Eingabe die richtige Antwort herauskommen könne, entgegnete Babbage nur trocken, er könne „die Art der Gedankenverwirrung nicht begreifen“, die eine solche Frage hervorrufen könne.

Babbages Difference Engine



Nachbau der Difference Engine No. 2 (Science Museum London): eine rein mechanische Rechenmaschine mit tausenden Zahnrädern.

Ada Lovelace

1815 – 1852, England

Schrieb 1843 den ersten veröffentlichten **Algorithmus** für eine Maschine (Berechnung der Bernoulli-Zahlen) – gilt daher als erste Programmiererin der Geschichte.

„The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.“

→ *eine Maschine tut nur, was man ihr vorgibt – nicht mehr, nicht weniger.*

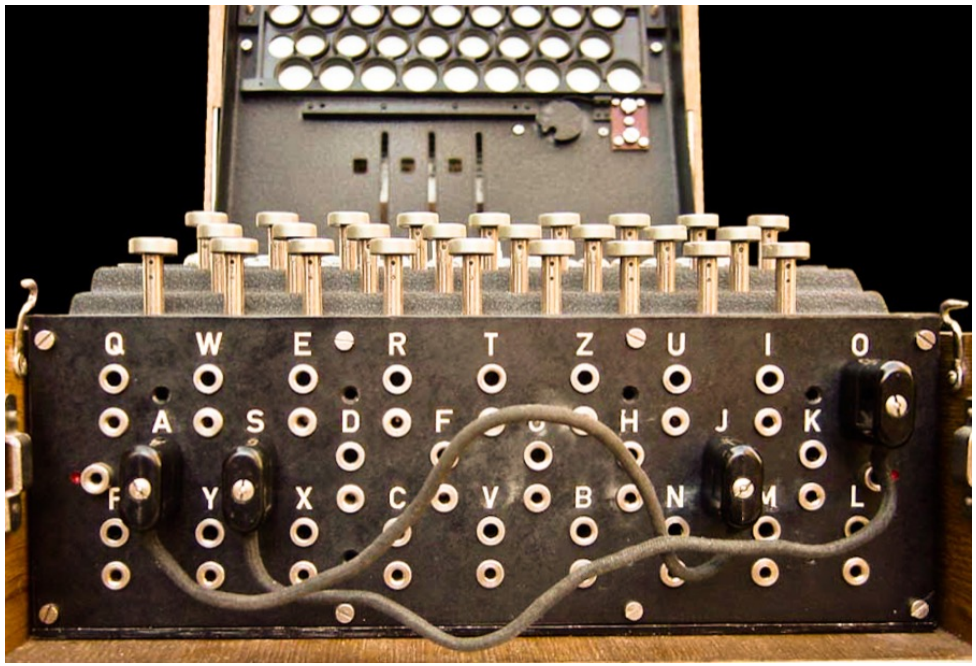
Alan Turing

1912 – 1954, England

Beschrieb 1936 mit der **Turingmaschine** ein universelles Rechenmodell – die theoretische Grundlage jedes heutigen Computers. Half im 2. Weltkrieg, den Enigma-Code zu knacken.

„We can only see a short distance ahead, but we can see plenty there that needs to be done.“

Die Enigma



Eine deutsche Enigma-Verschlüsselungsmaschine: Turings „Bombe“ half, ihren Code zu entschlüsseln – ein Wendepunkt des Krieges.

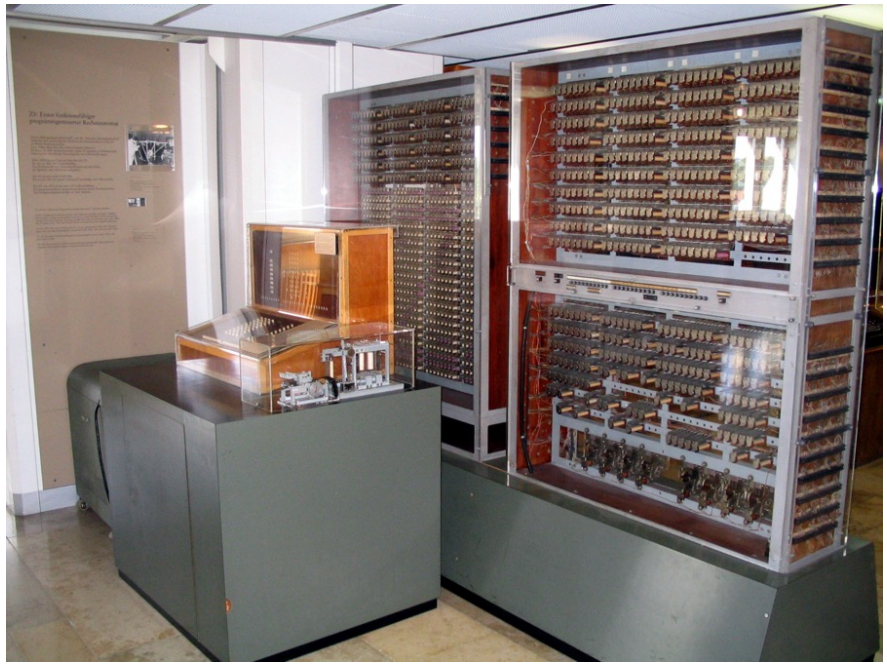
Konrad Zuse

1910 – 1995, Deutschland

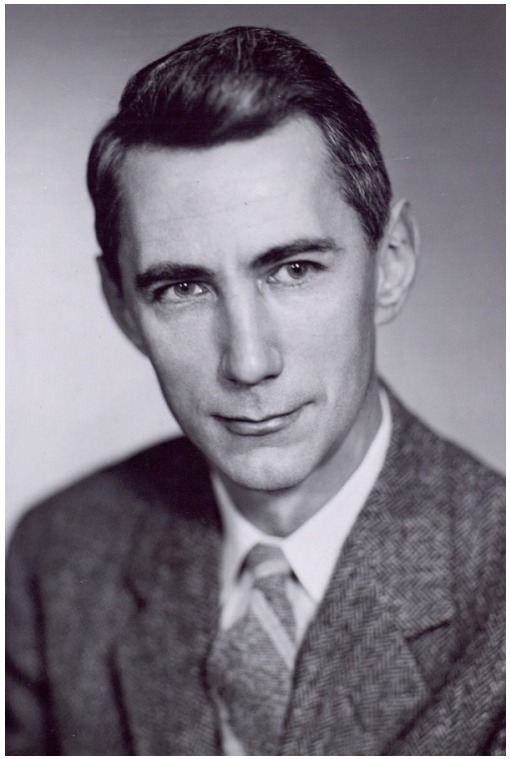
Baute 1941 mit der **Z3** den ersten funktionsfähigen, frei programmierbaren Computer der Welt – elektromechanisch, mit Binärzahlen.

Zuse entwickelte die Z3 in seiner Berliner Wohnung, während in den USA gleichzeitig staatlich finanzierte Grossprojekte wie der ENIAC entstanden.

Die Zuse Z3



Nachbau der Z3 im Deutschen Museum München: rund 2 600 Relais, gesteuert durch Lochstreifen.



Claude Shannon

1916 – 2001, USA

Zeigte 1937, dass sich logische Aussagen (UND, ODER, NICHT) mit elektrischen Schaltkreisen realisieren lassen – Geburtsstunde der digitalen Schaltungstechnik (siehe Kapitel 4, „Logische Gatter“).

„Information is the resolution of uncertainty.“

Shannon gilt als Begründer der Informationstheorie und prägte den Begriff ***bit****.*

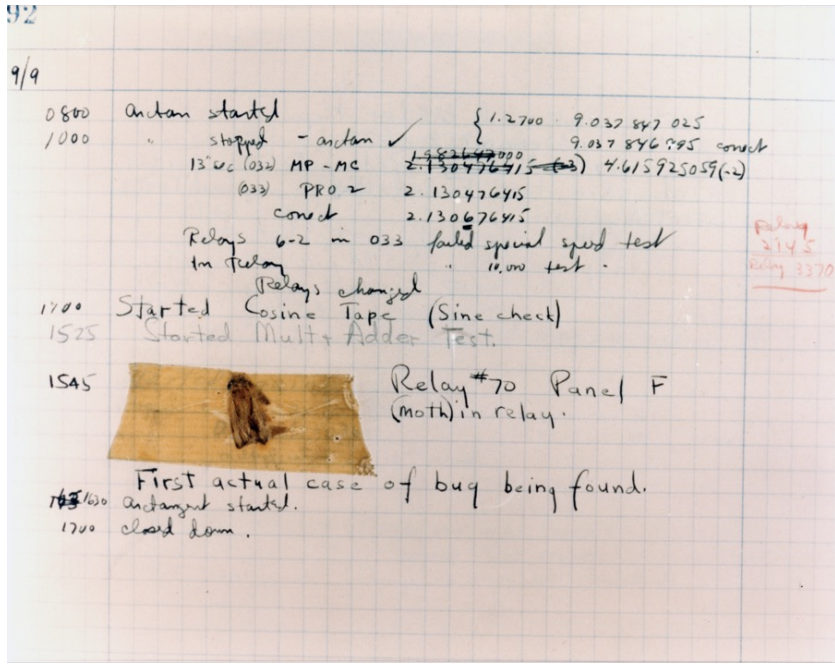
Grace Hopper

1906 – 1992, USA

Entwickelte den ersten **Compiler** und machte Programmiersprachen dadurch für Menschen lesbar (Vorläufer von COBOL).

„It's easier to ask forgiveness than it is to get permission."

Der erste „Bug“



9. September 1947: Hoppers Team findet eine Motte in einem Relais und klebt sie ins Logbuch – daher stammt (der Legende nach) der Begriff **Bug** für einen Programmierfehler.

Woher stammen Zahlen eigentlich?

Bevor wir zu Computern kommen: wie stellt man überhaupt eine Anzahl dar?

Historischer Hintergrund

Der Ishango-Knochen

Ca. 20 000 Jahre alt, gefunden in der heutigen Demokratischen Republik Kongo.

- Deutlich erkennbare, von Menschen eingeritzte Kerben
- Vermutete Bedeutung: eine gezählte Anzahl (z. B. gefangener Fische)

Von der Kerbe zur Zahl

Stellen wir uns vor, die Kerben zählen gefangene Fische.

drei Kerben $\leftrightarrow$  $\leftrightarrow$ $|||$ $\leftrightarrow$ ▪ ▪ ▪

Alle vier Darstellungen verwenden nur **ein einziges** Symbol $\rightarrow$
unäre Zahlendarstellung.

Unäre Zahlendarstellung

Vor- und Nachteile

Was könnten Nachteile dieser Zahlendarstellung sein?

Unäre Zahlendarstellung

Vor- und Nachteile

Was könnten Nachteile dieser Zahlendarstellung sein?

- Wie liest man diese Zahl auf einen Blick?
- Die Darstellungslänge wächst **linear** mit der Zahl – ungeeignet für grosse Zahlen
- Nicht besonders geeignet zum Rechnen

Lektion 02

Basisgrössen & Stellenwertdarstellung

Von römischen Münzen zum Dezimal- und Binärsystem · Skript, Kapitel 2

Rückblick

Letztes Mal: die ***unäre*** Zahlendarstellung.

- Ein einziges Symbol, Länge wächst linear
- Heute: Kulturen der Antike verwendeten bereits ***mehrere*** Symbole mit unterschiedlichem Wert

Römische Zahlen

Sieben Basisgrössen: I, V, X, L, C, D, M

Basisgrössen: Römische Zahlen

Welche Zahl steckt dahinter?

CCCCCLIIII

Basisgrössen: Römische Zahlen

Welche Zahl steckt dahinter?

CCCCCLIIII

554 $\rightarrow$ Gibt es eine kürzere Darstellung?

Basisgrössen: Römische Zahlen

Welche Zahl steckt dahinter?

CCCCCLIIII

554 $\rightarrow$ Gibt es eine kürzere Darstellung?

DLIV

Basisgrössen: Römische Zahlen

Welche Zahl steckt dahinter?

CCCCCLIIII

554 $\rightarrow$ Gibt es eine kürzere Darstellung?

DLIV

Basisgrössen lassen sich wie Münzen verstehen.

Die Münzen-Analogie

Eine Sammlung von Münzen heisst ***Zahlendarstellung*** einer Zahl m, wenn:

- ***Korrektheit***: die Summe der Münzenwerte gleich m ist,
- ***Minimalität***: keine Sammlung mit weniger Münzen existiert, deren Summe m ist.

Beispiel: *VII* ist die minimale römische Münzendarstellung von *sieben* (3 statt 7 Münzen).

Nicht immer eindeutig

Wir haben folgende Münzen: (1) (2) (3) (5) (10)

Wie stellt man die Zahl 4 dar?

Nicht immer eindeutig

Wir haben folgende Münzen: (1) (2) (3) (5) (10)

Wie stellt man die Zahl 4 dar?

- (1) (3)
- (2) (2)

Nicht immer eindeutig

Wir haben folgende Münzen: (1) (2) (3) (5) (10)

Wie stellt man die Zahl 4 dar?

- (1) (3)
- (2) (2)

→ **Die Darstellung ist minimal, aber nicht eindeutig.**

Auftrag

Kapitel 2: Ursprünge & Basisgrössen

Aufgabe 2.1: Römisches Münzsystem um zwei Münztypen erweiter

Aufgabe 2.2: Zeichenanzahl für 1 Milliarde im römischen System b

Stellenwertdarstellung

Vom Dezimal- zum Binärsystem

Warum unterschiedliche Zahlensysteme?

- Zahlensysteme sind „willkürlich“ und historisch entstanden
- Computer arbeiten mit binären Zahlen $\{0, 1\}$
  - Schnelleres, zuverlässigeres Rechnen
  - ⚡ 1 = Strom fliesst, 0 = kein Strom

Stellenwertdarstellung: Beobachtungen

- Alphabet des „normalen“ Zahlensystems:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

  - 10-adisches Zahlensystem, da 10 Zeichen
  - Dezimalsystem (lat. *decem* $\rightarrow$ „zehn“)

- Alphabet des binären Zahlensystems:

$$\{0, 1\}$$

  - 2-adisches Zahlensystem, da 2 Zeichen
  - Binärsystem (lat. *binarius* $\rightarrow$ „zweifach“)

Dezimal $\rightarrow$ Binär: erste Beispiele

Dezimal	**Binär**
0	$?_2$
1	$?_2$
2	$?_2$
3	$?_2$
4	$?_2$
...	...

Dezimal $\rightarrow$ Binär: erste Beispiele

Dezimal	**Binär**
0	0_2
1	1_2
2	10_2
3	11_2
4	100_2
...	...

Dezimal $\rightarrow$ Binär: erste Beispiele

Dezimal	**Binär**
0	0_2
1	1_2
2	10_2
3	11_2
4	100_2
...	...

Wie rechnet man das systematisch um? $\rightarrow$ ***b*-adische Zahlensysteme**

Basisgrössen im Stellenwertsystem

Das Dezimalsystem hat die Basisgrössen

$$10^0 = 1, \quad 10^1 = 10, \quad 10^2 = 100, \quad 10^3 = 1000, \ldots$$

Beispiel: Das Wort 4605 wird interpretiert als

$$\underbrace{\ldots}_{?}\underbrace{10^3}_{?} + \underbrace{\ldots}_{?}\underbrace{10^2}_{?} + \underbrace{\ldots}_{?}\underbrace{10^1}_{?} + \underbrace{\ldots}_{?}\underbrace{10^0}_{?}.$$

Basisgrössen im Stellenwertsystem

Das Dezimalsystem hat die Basisgrössen

$$10^0 = 1, \quad 10^1 = 10, \quad 10^2 = 100, \quad 10^3 = 1000, \ldots$$

Beispiel: Das Wort 4605 wird interpretiert als

$$4 \cdot \underbrace{10^3}_{1000} + 6 \cdot \underbrace{10^2}_{100} + 0 \cdot \underbrace{10^1}_{10} + 5 \cdot \underbrace{10^0}_{1}$$

Basisgrössen im Stellenwertsystem

Das Dezimalsystem hat die Basisgrössen

$$10^0 = 1, \quad 10^1 = 10, \quad 10^2 = 100, \quad 10^3 = 1000, \ldots$$

Beispiel: Das Wort 4605 wird interpretiert als

$$4 \cdot \underbrace{10^3}_{1000} + 6 \cdot \underbrace{10^2}_{100} + 0 \cdot \underbrace{10^1}_{10} + 5 \cdot \underbrace{10^0}_{1}$$

= vier Tausender + sechs Hunderter + null Zehner + fünf Einer.

Definition: b-adische Zahlendarstellung

Für eine Basis $b \geq 2$ definieren wir

$$(a_n a_{n-1} \ldots a_1 a_0)_b := a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b + a_0,$$

wobei $a_0, \ldots, a_n \in \{0, 1, \ldots, b-1\}$ ***Ziffern*** genannt werden.

Beispiel ($b = 3$): $\quad 211_3 = 2 \cdot 3^2 + 1 \cdot 3^1 + 1 \cdot 3^0 = 22$

Wichtige Beobachtung

Warum sind im b-adischen System nur Ziffern $0, \ldots, b-1$ erlaubt?

Wichtige Beobachtung

Warum sind im b-adischen System nur Ziffern $0, \ldots, b-1$ erlaubt?

- Beispiel: 13 Objekte im Dezimalsystem mit möglichst wenigen „Münzen“ darstellen.
- 10 Einer lassen sich stets durch 1 Zehner ersetzen $\rightarrow$ 9 Münzen gespart.
  - Münzen vorher: ① ① ① ① ① ① ① ① ① ① ① ① ①
  - Münzen nachher: ⑩ ① ① ①
- Allgemein: b Münzen vom Typ b^k lassen sich immer durch eine Münze vom Typ b^{k+1} ersetzen.

Warum ist das für die Informatik wichtig?

Die Bedingung $a_k \in \{0, 1, \ldots, b-1\}$ bringt zwei entscheidende Vorteile:

1. **Minimale Darstellung (Speichereffizienz):**
   Die Darstellung verbraucht so wenige Stellen (Bits) wie möglich.

2. **Eindeutigkeit (Kanonische Form):**
   Jede Zahl besitzt exakt *eine* Bitfolge $\rightarrow$ Computer können Werte ohne Mehrdeutigkeit vergleichen und speichern.

Warum ist das für die Informatik wichtig?

Die Bedingung $a_k \in \{0, 1, \ldots, b-1\}$ bringt zwei entscheidende Vorteile:

1. **Minimale Darstellung (Speichereffizienz):**
   Die Darstellung verbraucht so wenige Stellen (Bits) wie möglich.

2. **Eindeutigkeit (Kanonische Form):**
   Jede Zahl besitzt exakt *eine* Bitfolge $\rightarrow$ Computer können Werte ohne Mehrdeutigkeit vergleichen und speichern.

Das Bündeln von b Einheiten zur nächsten Stelle entspricht physikalisch dem Übertrag (*Carry-Bit*) in Prozessoren!

Auftrag

Kapitel 2: Stellenwertdarstellung

Aufgabe 2.3 & 2.4: Binärzahl $11011_2 \rightarrow$ Dezimal und Dezimal 14

Aufgabe 2.5: Darstellung der Zahl 29 im Binärsystem mit $\{0, 1\}$

Challenge: **Aufgabe 2.6 (Challenge)**: Warum ist Basis $b = 1$ nicht

Lektion 03

Umrechnung & Hexadezimalsystem

Der Greedy-Algorithmus und die Basis 16 · Skript, Kapitel 2

Rückblick

$$(a_n a_{n-1} \ldots a_1 a_0)_b := a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b + a_0$$

Heute: Wie rechnet man eine Dezimalzahl systematisch in ein anderes System um?

Umrechnung: Basis 5

Wie lautet die 5-adische Darstellung von 84_{10}? Verfügbare Potenzen: $5^0 = 1,\ 5^1 = 5,\ 5^2 = 25$ ($5^3 = 125 > 84$).

Umrechnung: Basis 5

Wie lautet die 5-adische Darstellung von 84_{10}? Verfügbare Potenzen: $5^0 = 1,\ 5^1 = 5,\ 5^2 = 25$ ($5^3 = 125 > 84$).

Grösstmögliche Potenz $5^2 = 25$: passt **3-mal** in 84. Rest:

$$84 - 3 \cdot 25 = 9.$$

Umrechnung: Basis 5

Wie lautet die 5-adische Darstellung von 84_{10}? Verfügbare Potenzen: $5^0 = 1,\ 5^1 = 5,\ 5^2 = 25$ ($5^3 = 125 > 84$).

Grösstmögliche Potenz $5^2 = 25$: passt **3-mal** in 84. Rest: $84 - 3 \cdot 25 = 9$.

$5^1 = 5$ passt **1-mal** in 9. Rest: $9 - 5 = 4$.

Umrechnung: Basis 5

Wie lautet die 5-adische Darstellung von 84_{10}? Verfügbare

Potenzen: $5^0 = 1,\ 5^1 = 5,\ 5^2 = 25$ ($5^3 = 125 > 84$).

Grösstmögliche Potenz $5^2 = 25$: passt **3-mal** in 84. Rest:
$84 - 3 \cdot 25 = 9$.

$5^1 = 5$ passt **1-mal** in 9. Rest: $9 - 5 = 4$.

$5^0 = 1$ passt **4-mal** in 4. Rest: 0.

Umrechnung: Basis 5

Wie lautet die 5-adische Darstellung von 84_{10}? Verfügbare

Potenzen: $5^0 = 1,\ 5^1 = 5,\ 5^2 = 25$ ($5^3 = 125 > 84$).

Grösstmögliche Potenz $5^2 = 25$: passt **3-mal** in 84. Rest:
$84 - 3 \cdot 25 = 9$.

$5^1 = 5$ passt **1-mal** in 9. Rest: $9 - 5 = 4$.

$5^0 = 1$ passt **4-mal** in 4. Rest: 0.

$$84_{10} = 314_5$$

Der Greedy-Algorithmus

- Wir haben stets mit der **grössten** Basisgrösse begonnen

Der Greedy-Algorithmus

- Wir haben stets mit der **grössten** Basisgrösse begonnen
- . . . und so viele Münzen davon verwendet wie möglich

Der Greedy-Algorithmus

- Wir haben stets mit der **grössten** Basisgrösse begonnen
- . . . und so viele Münzen davon verwendet wie möglich
- Diese Methode heisst ***greedy*** („gierig“)

Übung: Dezimal $\rightarrow$ Binär

Wandeln Sie 29_{10} ins Binärsystem um.

$$a_4 \cdot \underbrace{2^4}_{=?} + a_3 \cdot \underbrace{2^3}_{=?} + a_2 \cdot \underbrace{2^2}_{=?} + a_1 \cdot \underbrace{2^1}_{=?} + a_0 \cdot \underbrace{2^0}_{=?}$$

Übung: Dezimal $\rightarrow$ Binär

Wandeln Sie 29_{10} ins Binärsystem um.

$$a_4 \cdot \underbrace{2^4}_{=16} + a_3 \cdot \underbrace{2^3}_{=8} + a_2 \cdot \underbrace{2^2}_{=4} + a_1 \cdot \underbrace{2^1}_{=2} + a_0 \cdot \underbrace{2^0}_{=1}$$

Übung: Dezimal $\rightarrow$ Binär

Wandeln Sie 29_{10} ins Binärsystem um.

$$\overbrace{1}^{a_4} \cdot \underbrace{2^4}_{=16} + \overbrace{1}^{a_3} \cdot \underbrace{2^3}_{=8} + \overbrace{1}^{a_2} \cdot \underbrace{2^2}_{=4} + \overbrace{0}^{a_1} \cdot \underbrace{2^1}_{=2} + \overbrace{1}^{a_0} \cdot \underbrace{2^0}_{=1}$$

Übung: Dezimal $\rightarrow$ Binär

Wandeln Sie 29_{10} ins Binärsystem um.

$$\overbrace{1}^{a_4} \cdot \underbrace{2^4}_{=16} + \overbrace{1}^{a_3} \cdot \underbrace{2^3}_{=8} + \overbrace{1}^{a_2} \cdot \underbrace{2^2}_{=4} + \overbrace{0}^{a_1} \cdot \underbrace{2^1}_{=2} + \overbrace{1}^{a_0} \cdot \underbrace{2^0}_{=1}$$

$$\text{Resultat: } 29_{10} = 11101_2$$

Auftrag

Kapitel 2: Umrechnungen zwischen Darstellungen

Aufgabe 2.7: Ordnen Sie die Zahlen $43_{10}, 2201_3, 11101_2, 33_4, 142_5,$

Challenge: **Aufgabe 2.8 (Challenge)**: Eindeutigkeit der 10-adische

Das Hexadezimalsystem

Basis 16 – warum reichen Ziffern allein nicht mehr aus?

Wie viele Ziffern braucht ein b-adisches System?

- Binärsystem (2 Ziffern): grösste Ziffer ist 1 $(= 2 - 1)$
- Dezimalsystem (10 Ziffern): grösste Ziffer ist 9 $(= 10 - 1)$
- Allgemein: grösste Ziffer in einem b-adischen System ist $b - 1$

Was tun wir, wenn $b > 10$? Die dezimale „10“ ist ja selbst schon aus zwei Ziffern zusammengesetzt...

Das Hexadezimalsystem ($b = 16$)

Ziffern sind letztlich nur „Zeichnungen“ für Mengen – wir können also auch Buchstaben verwenden:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

z. B. steht B für die Zahl *elf*.

Frage: Wie lautet 27_{10} hexadezimal?

Das Hexadezimalsystem ($b = 16$)

Ziffern sind letztlich nur „Zeichnungen“ für Mengen – wir können also auch Buchstaben verwenden:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

z. B. steht B für die Zahl *elf*.

Frage: Wie lautet 27_{10} hexadezimal?

$$27_{10} = 1 \cdot 16^1 + 11 \cdot 16^0 = 1B_{16}$$

Warum Hexadezimal in der Informatik?

Was ist die grösste binäre Zahl, die man mit 8 Bits (einem Byte) darstellen kann?

Warum Hexadezimal in der Informatik?

Was ist die grösste binäre Zahl, die man mit 8 Bits (einem Byte) darstellen kann?

$$11111111_2 = 255_{10} = FF_{16}$$

Zwei Hex-Ziffern genügen, um ein ganzes Byte kompakt darzustellen!

Auftrag

Kapitel 2: Hexadezimalsystem

Aufgabe 2.9: Zahl 27_{10} im Hexadezimalsystem darstellen

Aufgabe 2.10: Zahlen 44, 23, 90, 124, 306 hexadezimal ausgeben

Lektion 04

Bits, Bytes & Hexadezimalsystem

Die kleinste Informationseinheit, Byte-Strukturen und Basis 16 · Skript, Kapitel 2

Bits & Bytes

Wie zählt und speichert man mit Nullen und Einsen?

Bits & Bytes

- **bit** = „*binary digit*“, „binäre Ziffer“ (eine 0 oder eine 1)

Bits & Bytes

- **bit** = „*binary digit*“, „binäre Ziffer“ (eine 0 oder eine 1)
- **byte** = „Bissen“ (von engl. *bite* = Bissen, aus 8 bits) 😋

Bits & Bytes

- **bit** = „*binary digit*“, „binäre Ziffer“ (eine 0 oder eine 1)
- **byte** = „Bissen“ (von engl. *bite* = Bissen, aus 8 bits) 😋
- Mit einem Byte lassen sich $2^8 = 256$ verschiedene Werte darstellen

Führende Nullen

Die binäre Repräsentation der Dezimalzahl 22:

$$00010110_2 = 22_{10}$$

Bei Binärzahlen werden – anders als bei Dezimalzahlen – üblicherweise **alle** Bits ausgeschrieben.

Frage: Wie stellt man 42_{10} mit einem Byte dar?

Führende Nullen

Die binäre Repräsentation der Dezimalzahl 22:

$$00010110_2 = 22_{10}$$

Bei Binärzahlen werden – anders als bei Dezimalzahlen – üblicherweise **alle** Bits ausgeschrieben.

Frage: Wie stellt man 42_{10} mit einem Byte dar?

$$42_{10} = 00101010_2$$

Grössenordnungen

Einheit	**Grösse**	Beispiel
KB (Kilobyte)	10^3 B	eine Text-Datei
MB (Megabyte)	10^6 B	eine Musik-Datei
GB (Gigabyte)	10^9 B	eine Video-Datei
TB (Terabyte)	10^{12} B	kleiner Firmen-Server

Das Hexadezimalsystem

Warum brauchen wir die Basis 16 in der Informatik?

Das Hexadezimalsystem ($b = 16$)

Binärzahlen wie 11111111_2 sind lang und unübersichtlich. Das Hexadezimalsystem nutzt 16 Ziffern:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

$$A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$$

Notwendigkeit & Vorteil von Hexadezimal

Da $16 = 2^4$, entspricht **1 Hex-Ziffer** genau **4 Bits** (einem Halbbyte/Nibble).

1 Byte (8 Bits) = exakt 2 Hex-Ziffern!

$$11111111_2 = FF_{16} = 255_{10}$$
$$10101100_2 = AC_{16} = 172_{10}$$

Viel kompakter, besser lesbar und weniger anfällig für Tippfehler!

Auftrag

Kapitel 2: Bits, Bytes & Hexadezimal

Aufgabe 2.1 & 2.2: 42_{10} als Byte ausdrücken & grösste Zahl mit 1

Aufgabe 2.3 & 2.4: Hexadezimale Umrechnungen (27_{10}, 44, 23, 90

Lektion 05

Kodierung von Bildern & Farben

Pixel, RGB und Farbtiefe · Skript, Kapitel 2

Ein Schwarz-Weiss-Bild

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

Welches Bild wird dargestellt?
Wie viel Bit Speicherplatz benötigt es?

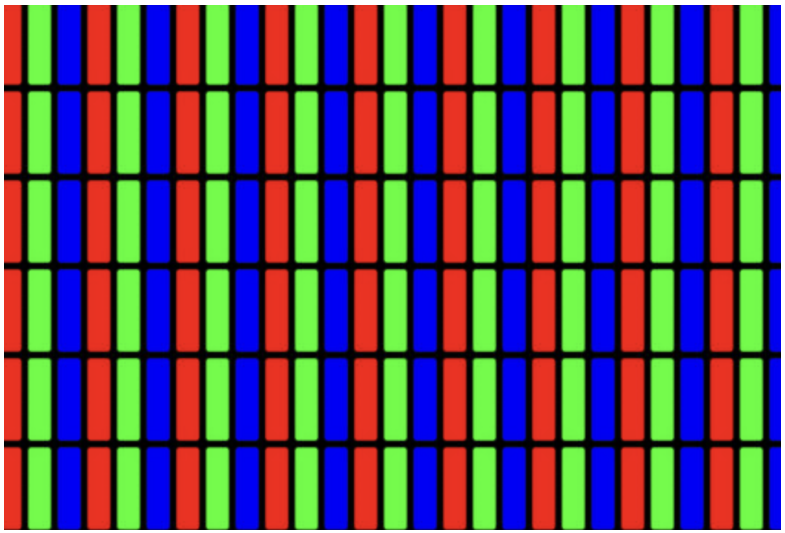
Ein Schwarz-Weiss-Bild

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

Welches Bild wird dargestellt?
Wie viel Bit Speicherplatz benötigt es?

Ein Kreuz $\cdot$ $5 \cdot 5 = 25$ **Bit**

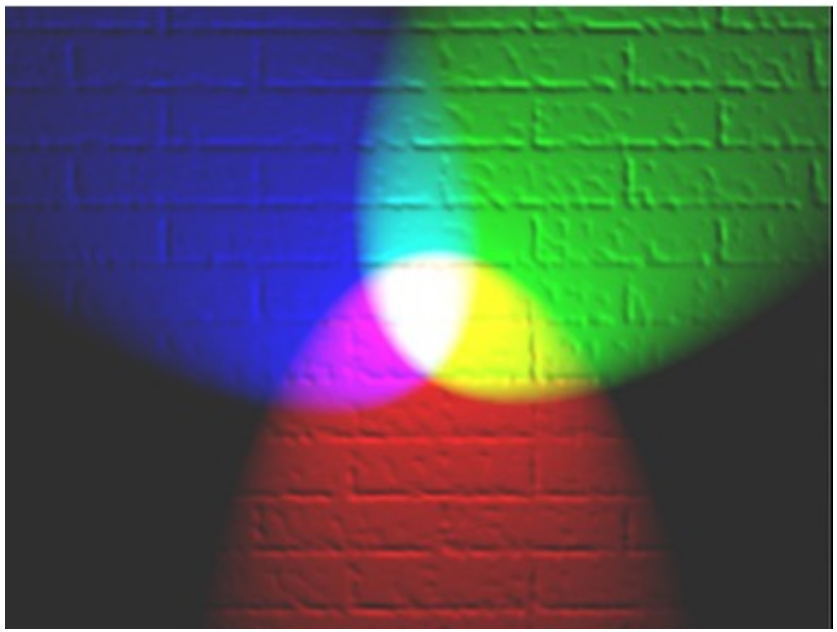
Vom Pixel zum Bildschirm

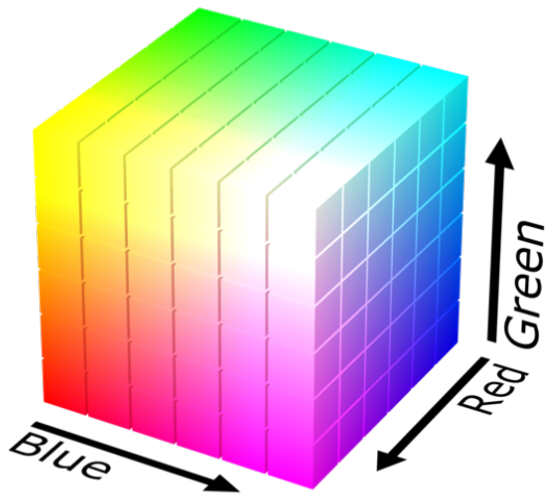


Pixel = „picture“ + „element“

- Winzige rot/grün/blau leuchtende Light-Emitting Diodes (LEDs)
- Kleinstes Bildelement eines Bildschirms

Additive Farbmischung





Jede Farbe wird durch drei Werte (Rot, Grün, Blau) zwischen 0 und 255 kodiert.

RGB und Hexadezimal

24 Bit pro Farbe (je 8 Bit für R, G, B) $\rightarrow$ hexadezimal viel kompakter:

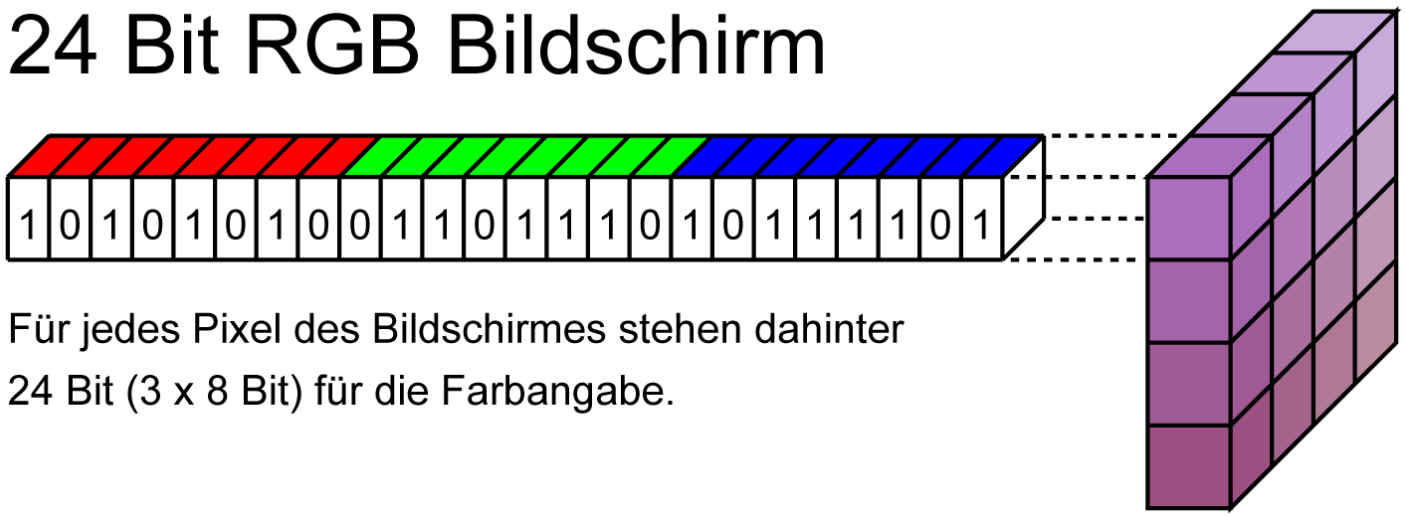
`11111111000000000000000` = `#FF0000` = `255, 0, 0` = reines Rot

`100000001000000010000000` = `#808080` = `128, 128, 128` = grau

Wie viele Farben lassen sich damit insgesamt darstellen?

$2^{24} \approx 16.7$ Mio.

Farbtiefe



Für jedes Pixel des Bildschirmes stehen dahinter
24 Bit (3 x 8 Bit) für die Farbangabe.

Farbtiefe: Beispiele

1 Bit
2 mögliche Werte

2 Bit
4 mögliche Werte

4 Bit
16 mögliche Werte

8 Bit
256 mögliche Werte

Je geringer die Farbtiefe, desto weniger fein abgestuft.

GIF und Farbpaletten

Graphics Interchange Format (GIF)-Bilder unterscheiden meist 256 Farben. Wie viele Bits pro Pixel braucht das?

GIF und Farbpaletten

GIF-Bilder unterscheiden meist 256 Farben. Wie viele Bits pro Pixel braucht das?

8 Bit

Statt für jedes Pixel dieselbe Anzahl Bits pro Farbe zu speichern, wird oft ein ***Index*** auf eine Farbpalette verwendet.

Auftrag

Kapitel 2: Bilder & Farben

Übungen: Schwarz-Weiss-Bild, Pixel-Speicherbedarf & GIF-Palette

Übungen: Hexadezimale Farbzuordnung & Speicherbedarf 500 × 50

Challenge: **Challenge**: Weitere Farben erraten

Lektion 06

Kodierung von Buchstaben

ASCII und UTF-8 · Skript, Kapitel 2

Wie kodiert man Buchstaben?

Computer speichern nur Nullen und Einsen. Wie könnte man damit **Buchstaben** repräsentieren?

ASCII

American Standard Code for Information Interchange (ASCII) – eine der ersten Text-Kodierungen

- 1 Byte pro Zeichen – z. B. 0100 0001 $\rightarrow$ A
- 1 Bit war für Fehlerkontrolle reserviert (Paritätsbit)

ASCII

ASCII – eine der ersten Text-Kodierungen

- 1 Byte pro Zeichen – z. B. 0100 0001 $\rightarrow$ A
- 1 Bit war für Fehlerkontrolle reserviert (Paritätsbit)

Frage: Wie viele Zeichen kann man mit einem Byte abbilden? Und mit nur 7 nutzbaren Bits (ASCII)?

ASCII

ASCII – eine der ersten Text-Kodierungen

- 1 Byte pro Zeichen – z. B. `0100 0001` $\rightarrow$ A
- 1 Bit war für Fehlerkontrolle reserviert (Paritätsbit)

Frage: Wie viele Zeichen kann man mit einem Byte abbilden? Und mit nur 7 nutzbaren Bits (ASCII)?

$$2^8 = 256, \quad 2^7 = 128$$

ASCII-Tabelle (Auszug)

Dezimal	Hexadezimal	Binär	Zeichen
32	20	00100000	Space
48	30	00110000	0
49	31	00110001	1
...	...	...	...
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
...	...	...	...
97	61	01100001	a
98	62	01100010	b

Vollständige Tabelle: Skript, Kapitel 3.

Entschlüsseln Sie!

01101000 01100101 01101100 01101100 01101111

Nutzen Sie die ASCII-Tabelle im Skript: welches Wort ist das?

Entschlüsseln Sie!

`01101000 01100101 01101100 01101100 01101111`

Nutzen Sie die ASCII-Tabelle im Skript: welches Wort ist das?

`hello`

Die Grenzen von ASCII

128 Zeichen reichen bei weitem nicht für Umlaute, Akzente, Emojis, Sprachen wie Chinesisch, Arabisch, ...

→ Unicode Transformation Format — 8-bit (UTF-8): bis zu **4 Bytes** pro Zeichen

Die Grenzen von ASCII

128 Zeichen reichen bei weitem nicht für Umlaute, Akzente, Emojis, Sprachen wie Chinesisch, Arabisch, ...

$\rightarrow$ UTF-8: bis zu **4 Bytes** pro Zeichen

Frage: Wie viele Zeichen kann man mit 4 Bytes maximal abbilden?

Die Grenzen von ASCII

128 Zeichen reichen bei weitem nicht für Umlaute, Akzente, Emojis, Sprachen wie Chinesisch, Arabisch, ...

$\rightarrow$ UTF-8: bis zu **4 Bytes** pro Zeichen

Frage: Wie viele Zeichen kann man mit 4 Bytes maximal abbilden?

$2^{32} = 4\,294\,967\,296$ (etwas weniger nutzbar, da Bits zur Längenangabe gebraucht werden)

Auftrag

Kapitel 3: ASCII & UTF-8

Aufgabe 3.3: Wie viele Zeichen lassen sich mit 7-Bit ASCII abbilde

Aufgabe 3.4: Grösste binäre Zahl mit 8 Bits und in Hexadezimal

Aufgabe 3.5: Maximale Anzahl Zeichen bei 4-Byte UTF-8

Lektion 07

Dateiformate & Speicherplatz

Wie man aus Bits ein Dateiformat erkennt · Skript, Kapitel 3

Eine einfache Frage

Wir wissen: Computer speichern nur Nullen und Einsen.

Woher „weiss“ ein Computer, ob eine Bitfolge ein Bild, eine Zahl, ein Buchstabe oder etwas anderes ist?

Dateiformate geben Bedeutung

Datei	Bedeutung (Beispiel)
`.txt`	Zeichen: $< \mid \ldots$
`.doc`	Formatierbefehl: „Einzug 1.75 cm“
`.xls`	Zahl 9662249 in einer Zelle
`.gif`	Bildpunkt, indexiert (256 Farben, 1 Byte/Pixel)
`.bmp`	Bildpunkt, direkt (16.7 Mio. Farben, 3 Bytes/Pixel)

Dieselbe Bitfolge kann je nach Dateiformat ganz unterschiedlich interpretiert werden!

Speichergrössen

Masseinheit	**Dezimalsystem**		Grössenordnung
KB (Kilobyte)	10^3 B	1'000 B	eine Text-Datei
MB (Megabyte)	10^6 B	1'000'000 B	eine Musik-Datei
GB (Gigabyte)	10^9 B	1'000'000'000 B	eine Video-Datei
TB (Terabyte)	10^{12} B	1'000'000'000'000 B	kleiner Firmen-Server
PB (Petabyte)	10^{15} B	...	Facebook-Server (Meta)
EB (Exabyte)	10^{18} B	...	alle CERN-Daten
ZB (Zettabyte)	10^{21} B	...	alle Daten ($\sim$ 100 ZB)

Speichergrössen

Masseinheit	**Dezimalsystem**		Grössenordnung
KB (Kilobyte)	10^3 B	1'000 B	eine Text-Datei
MB (Megabyte)	10^6 B	1'000'000 B	eine Musik-Datei
GB (Gigabyte)	10^9 B	1'000'000'000 B	eine Video-Datei
TB (Terabyte)	10^{12} B	1'000'000'000'000 B	kleiner Firmen-Server
PB (Petabyte)	10^{15} B	...	Facebook-Server (Meta)
EB (Exabyte)	10^{18} B	...	alle CERN-Daten
ZB (Zettabyte)	10^{21} B	...	alle Daten ($\sim$ 100 ZB)

Wie nennt man ein „Megagramm“ geläufiger? $\rightarrow$ eine Tonne

Diskussion

Weshalb haben Festplatten häufig eine leicht kleinere Speichergrösse als angegeben, insbesondere unter Windows?

Diskussion

Weshalb haben Festplatten häufig eine leicht kleinere Speichergrösse als angegeben, insbesondere unter Windows?

Hersteller rechnen mit $1\,\mathrm{GB} = 10^9$ Byte, das Betriebssystem oft mit $1\,\mathrm{GiB} = 2^{30}$ Byte – ein kleiner, aber spürbarer Unterschied.

Auftrag

Kapitel 3: Dateiformate & Speicherbedarf

Aufgabe 3.15: Video anschauen (Passwort: BinCode)

Aufgabe 3.16: Üblicher/gebräuchlicher Name für ein „Megagramm

Challenge: **Aufgabe 3.17 (Challenge)**: Warum Festplatten unter W

Lektion 08

Logische Gatter I

Vom Stromkreis zum Gatter: NOT, AND, OR · Skript, Kapitel 4

Vom Stromkreis zur Information

Ein Chip ist ein Netzwerk aus leitenden Verbindungen und Schaltern.

Zentrale Idee: Elektrische Zustände werden als logische Zustände interpretiert.

niedrige Spannung $\longleftrightarrow$ 0, hohe Spannung $\longleftrightarrow$ 1.

Warum ausgerechnet zwei Zustände?

- In echten Schaltungen gibt es Rauschen, Verzögerungen, Fertigungstoleranzen
- Man arbeitet daher mit ***Spannungsbereichen***, nicht mit exakten Werten
- Zwei klar unterscheidbare Zustände lassen sich sehr zuverlässig erkennen

Erinnerung an Lektion 1: Shannon zeigte 1937, dass sich Boolesche Logik so elektrisch realisieren lässt.

Vom Transistor zum Gatter

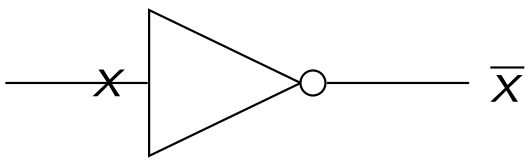
- ***Transistoren*** = extrem schnelle elektronische Schalter
- Ein logisches ***Gatter*** ist eine konkrete Schaltung aus mehreren Transistoren
- Reale Gatter haben technische Kennwerte: Propagationsverzögerung, Leistungsaufnahme, Fan-out

Boolesche Grundverknüpfungen

Funktionen, die Bits als Eingabe erhalten und **genau ein** Bit ausgeben.

NOT **AND** **OR**

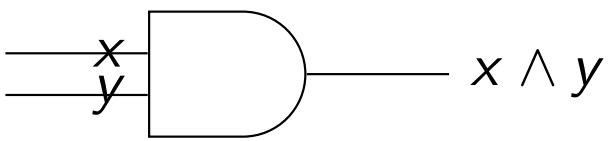
NOT – Negation



x	$\overline{x}$
0	1
1	0

Ein Eingang, negierter Ausgang.

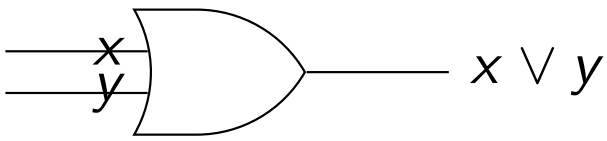
AND – Konjunktion



x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Ausgang 1 nur, wenn **beide** Eingänge 1 sind.

OR – Disjunktion



x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Ausgang 1, wenn **mindestens ein** Eingang 1 ist.

Übung: Gatter kombinieren

Ein Fahrstuhl darf sich nur bewegen, wenn die Tür geschlossen *und* entweder das Übergewicht nicht überschritten *oder* der Notfall-Modus deaktiviert ist.

Modellieren Sie diese Bedingung mit AND-, OR- und NOT-Gattern.

Übung: Gatter kombinieren

Ein Fahrstuhl darf sich nur bewegen, wenn die Tür geschlossen *und* entweder das Übergewicht nicht überschritten *oder* der Notfall-Modus deaktiviert ist.

Modellieren Sie diese Bedingung mit AND-, OR- und NOT-Gattern.

$$t \wedge (\overline{u} \vee \overline{n})$$

Auftrag

Kapitel 4: Logische Grundverknüpfungen

Aufgabe 4.1: Modellieren Sie die Fahrstuhl-Bedingung mithilfe von

Lektion 09

Logische Gatter II

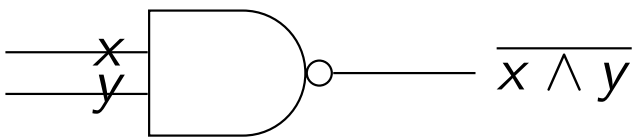
NAND, funktionale Vollständigkeit und der Halbaddierer · Skript, Kapitel 4

Rückblick

Letztes Mal: NOT, AND, OR – die drei elementaren Booleschen Verknüpfungen.

Heute: ein einziges Gatter, aus dem sich **alle** anderen bauen lassen.

Das NAND-Gatter



x	y	$\overline{x \wedge y}$
0	0	1
0	1	1
1	0	1
1	1	0

$x \uparrow y = \overline{x \wedge y}$ – die Negation von AND.

Funktional vollständig

Aus Kombinationen von NAND-Gattern lassen sich **alle** anderen logischen Funktionen nachbauen.

Übung: Stellen Sie die Wahrheitstabelle von $x \uparrow x$ auf. Was fällt auf?

Funktional vollständig

Aus Kombinationen von NAND-Gattern lassen sich **alle** anderen logischen Funktionen nachbauen.

Übung: Stellen Sie die Wahrheitstabelle von $x \uparrow x$ auf. Was fällt auf?

x	$x \wedge x$	$\overline{x \wedge x}$
0	0	1
1	1	0

$x \uparrow x = \overline{x}$ – **ein NOT-Gatter aus einem einzigen NAND!**

Warum ist das praktisch relevant?

In vielen Fertigungsprozessen ist es technologisch günstig, sehr grosse Schaltungen aus nur **einem** standardisierten Gattertyp aufzubauen – NAND ist ein solcher „Universalbaustein“.

Der Halbaddierer

Wenn Gatter tatsächlich rechnen

Zwei Bits addieren

x	y		Übertrag c	Summe s
0	0		0	0
0	1	$\longrightarrow$	0	1
1	0		0	1
1	1		1	0

Erkennen Sie zwei bekannte Gatter in diesen Spalten?

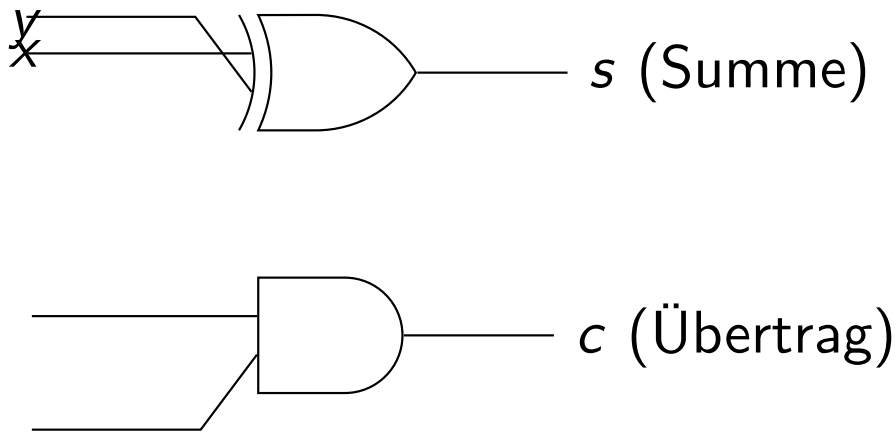
Zwei Bits addieren

x	y		Übertrag c	Summe s
0	0		0	0
0	1	$\longrightarrow$	0	1
1	0		0	1
1	1		1	0

Erkennen Sie zwei bekannte Gatter in diesen Spalten?

$s = x \oplus y$ **(XOR),** $c = x \wedge y$ **(AND)**

Schaltplan des Halbaddierers



XOR liefert die Summe, AND den Übertrag.

Übung: Halbaddierer-Wahrheitstabelle

x	y	Summe s $(x \oplus y)$	Übertrag c $(x \wedge y)$	Binär $(cs)_2$	Dezimal
0	0				
0	1				
1	0				
1	1				

Übung: Halbaddierer-Wahrheitstabelle

x	y	Summe s $(x \oplus y)$	Übertrag c $(x \wedge y)$	Binär $(cs)_2$	Dezimal
0	0	0	0	00_2	0
0	1	1	0	01_2	1
1	0	1	0	01_2	1
1	1	0	1	10_2	2

Der Halbaddierer berechnet alle 4 Fälle der 1-Bit-Addition exakt korrekt!

Der Volladdierer

Mehrstellige Zahlen addieren mit Übertrag

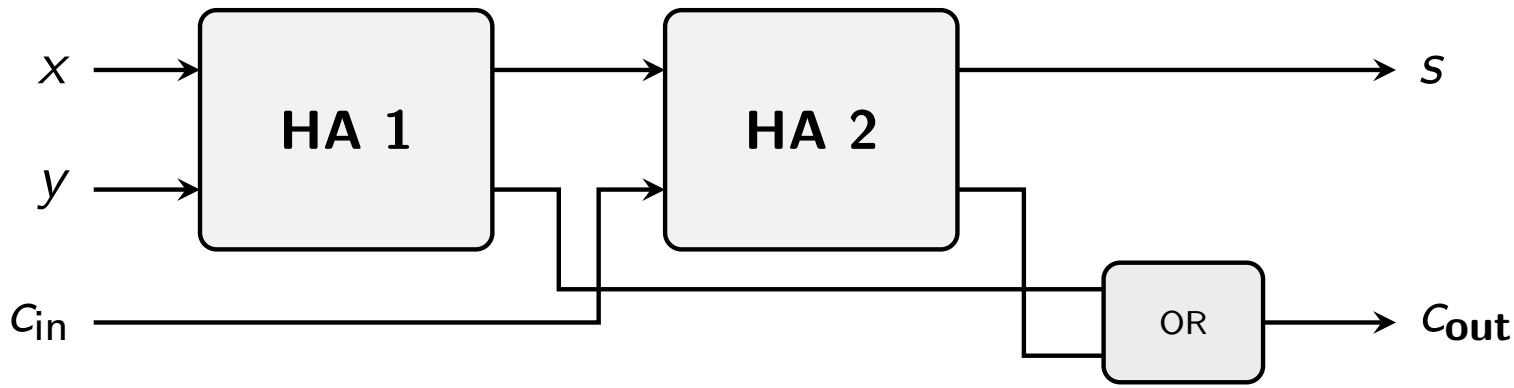
Warum reicht ein Halbaddierer nicht aus?

Beim schriftlichen Addieren mehrerer Stellen entsteht aus der vorherigen Stelle ein **eingehender Übertrag c_{in}**.

Ein **Volladdierer (Full Adder)** verarbeitet **3 Eingänge**: x, y und c_{in}.

Aufbau des Volladdierers

Ein Volladdierer besteht aus **zwei Halbaddieren** und einem **OR-Gatter**:



Schriftliche Binäraddition im Addierwerk

Mehrere Volladdierer werden kaskadiert (*Ripple-Carry-Addierer*), um ganze Binärzahlen zu addieren:

$$
\begin{array}{cccccccc}
 & & 1 & 0 & 1 & 0 & 1 & 1_2 \\
+ & & 1 & 1 & 1 & 0 & 1 & 0_2 \\
 & & \scriptstyle 1 & \scriptstyle 1 & & \scriptstyle 1 & & \\
\hline
 & 1 & 1 & 0 & 0 & 1 & 0 & 1_2 \\
\hline\hline
\end{array}
$$

Jede Spalte wird von genau einem Volladdierer-Baustein berechnet!

Auftrag

Kapitel 4: NAND, Halbaddierer & Volladdierer

Aufgabe 4.2: NOT-Gatter allein mit NAND-Gattern realisieren ($x \uparrow$

Challenge: **Aufgabe 4.3 (Challenge)**: AND-Gatter allein aus NAN

Aufgabe 4.4: Tabellen & Übungen zum Halb- und Volladdierer aus

Lektion 10

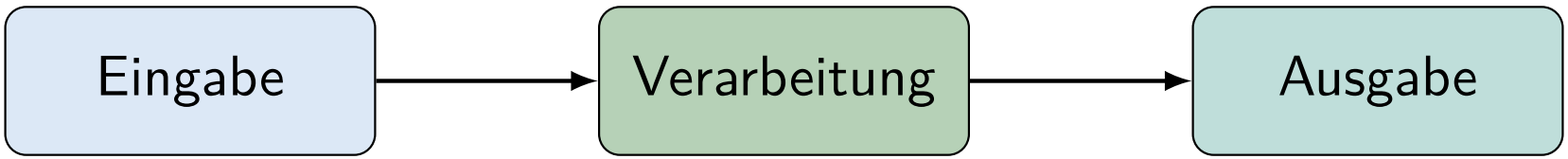
Aufbau eines Computers I

Von-Neumann-Architektur & die CPU im Detail · Skript, Kapitel 5

Vom Gatter zum Computer

Wir kennen jetzt Gatter und den Halbaddierer.

Wie setzt man daraus einen **vollständigen Computer** zusammen?



Die Von-Neumann-Architektur

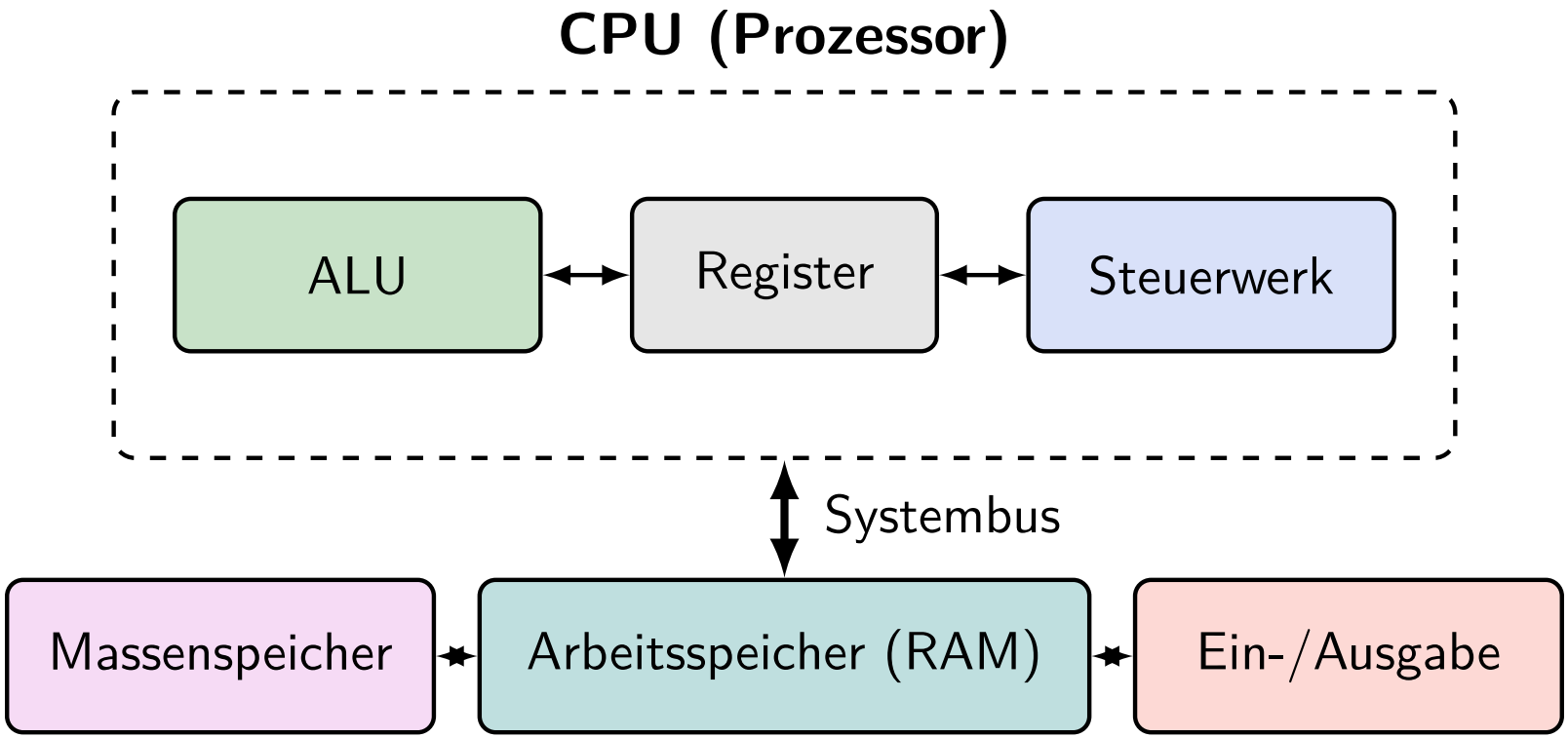
1945, John von Neumann

Das entscheidende Merkmal

Programm und Daten liegen im **gleichen** Speicher.

Das Programm ist selbst nur ein Stück Daten – also eine Bitfolge,
die jederzeit verändert oder ausgetauscht werden kann.

Die Hauptkomponenten



Warum ist das riskant?

Was könnte passieren, wenn ein fehlerhaftes Programm versehentlich seinen eigenen Programmcode überschreibt?

Warum ist das riskant?

Was könnte passieren, wenn ein fehlerhaftes Programm versehentlich seinen eigenen Programmcode überschreibt?

Unvorhersehbares Verhalten oder Absturz – deshalb übernimmt später das Betriebssystem den „Speicherschutz“ (Lektion 12).

Die CPU im Detail

ALU, Steuerwerk, Register

ALU – Arithmetisch-logische Einheit

- Das „Rechenzentrum“ der CPU
- Arithmetische Operationen (Addition, Subtraktion, . . .)
- Logische Operationen (AND, OR, Vergleiche, . . .)
- Intern: eine grosse Anzahl kombinierter Logikgatter (z. B. Volladdierer)

Steuerwerk & Register

Steuerwerk

- **Steuert den Ablauf aller Operationen**
- **Liest Befehle, interpretiert sie**
- **Gibt Steuersignale an ALU & Co.**

Register

- **Sehr schnelle, winzige Zwischenspeicher**
- **Halten die aktuellen Rechenwerte**
- **Typische Grösse: 64 Bit (= 8 Byte)**

Auftrag

Kapitel 5: Von-Neumann-Architektur & CPU

Aufgabe 5.1: Überlegen Sie, warum es problematisch ist, wenn Pro

Lektion 11

Aufbau eines Computers II

RAM, Massenspeicher & der Fetch-Decode-Execute-Zyklus · Skript, Kapitel 5

Der Arbeitsspeicher (RAM)

Stellen Sie sich RAM als eine lange Tabelle vor: jede Zeile ist eine Speicherzelle mit eindeutiger Adresse.

Adresse	**Inhalt (1 Byte)**
`0x00`	`10110100`
`0x01`	`00001111`
`0x02`	`11001010`
⋮	⋮

Die Adressen sind hexadezimal (Lektion 3) – daher das Präfix `0x`.

RAM vs. Cache

- RAM ist ***flüchtig***: Inhalt geht beim Ausschalten verloren
- Typische Grösse: 8–64 GB
- Der ***Cache***: noch schneller, aber viel kleiner – speichert häufig genutzte Daten zwischen

Massenspeicher

HDD Magnetische

Scheiben speichern Bits als winzige magnetisierte Bereiche. Langsam, aber billig und langlebig.

SSD Daten als elektrische

Ladung in Flashspeicher-Zellen. Deutlich schneller, keine beweglichen Teile.

RAM: 30–50 GB/s SSD: 1–7 GB/s HDD: 100–200 MB/s

Übung: Ladezeit

Ein Foto (24 MB) wird von einer SSD mit 2 GB/s Lesegeschwindigkeit geladen. Wie lange dauert das?

Übung: Ladezeit

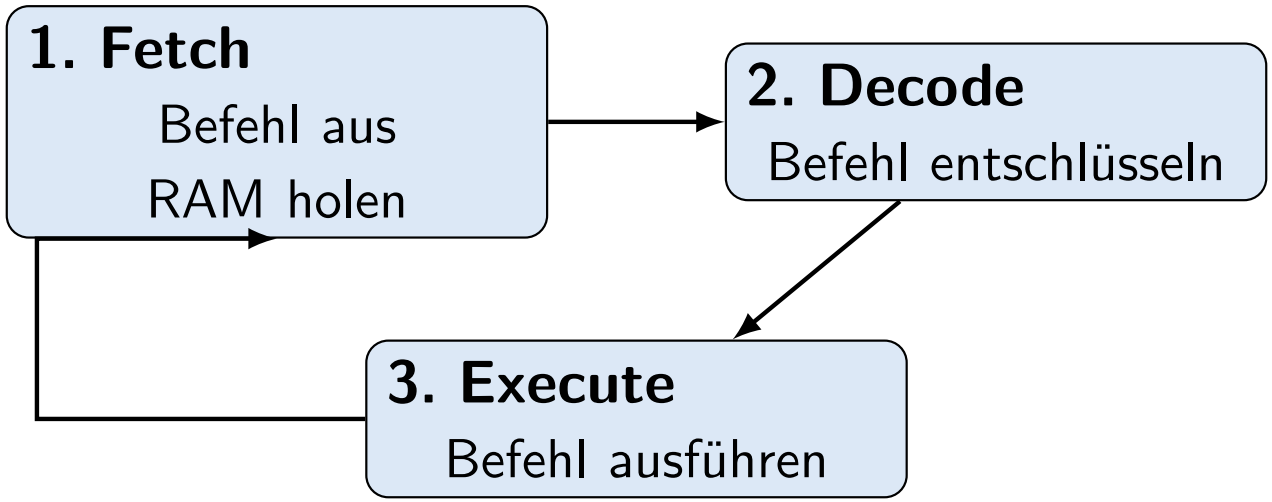
Ein Foto (24 MB) wird von einer SSD mit 2 GB/s Lesegeschwindigkeit geladen. Wie lange dauert das?

$$\frac{24\,\mathrm{MB}}{2000\,\mathrm{MB/s}} = 12\,\mathrm{ms}$$

Fetch – Decode – Execute

Wie die CPU Befehle abarbeitet

Der Fetch-Decode-Execute-Zyklus



Die CPU wiederholt diesen Ablauf Milliarden Mal pro Sekunde.

Was passiert in jeder Phase?

- ***Fetch***: Steuerwerk liest den nächsten Befehl aus dem RAM (Adresse = Programmzähler)

Was passiert in jeder Phase?

- ***Fetch***: Steuerwerk liest den nächsten Befehl aus dem RAM (Adresse = Programmzähler)
- ***Decode***: Befehl wird zerlegt – *was* tun, *mit welchen* Operanden?

Was passiert in jeder Phase?

- ***Fetch***: Steuerwerk liest den nächsten Befehl aus dem RAM (Adresse = Programmzähler)
- ***Decode***: Befehl wird zerlegt – *was* tun, *mit welchen* Operanden?
- ***Execute***: ALU (o. ä.) führt die Anweisung aus, Ergebnis wird gespeichert, Programmzähler rückt weiter

Übung: Taktfrequenz

Ein Prozessor hat 3,6 GHz Taktfrequenz und führt im Schnitt 2 Befehle pro Taktzyklus aus.

Wie viele Befehle führt er pro Sekunde aus? Wie lange dauert ein Programm mit $7{,}2 \times 10^9$ Befehlen?

Übung: Taktfrequenz

Ein Prozessor hat $3{,}6\,\mathrm{GHz}$ Taktfrequenz und führt im Schnitt 2 Befehle pro Taktzyklus aus.

Wie viele Befehle führt er pro Sekunde aus? Wie lange dauert ein Programm mit $7{,}2 \times 10^9$ Befehlen?

$$7{,}2 \times 10^9 \text{ Befehle/s} \quad \rightarrow \quad 1\,\mathrm{s}$$

Auftrag

Kapitel 5: Speicher & FDE-Zyklus

Aufgabe 5.2: Dauer berechnen: Foto (24 MB) von SSD (2 GB/s) l

Aufgabe 5.3: Befehle/s und Ausführungsdauer bei 3,6 GHz berechn

Challenge: **Aufgabe 5.4 (Challenge)**: Teilschritte des Befehls `ADD`

Lektion 12

Betriebssystem & Software-Schichten

Vom Tastendruck bis zum Bildschirm · Skript, Kapitel 6

Das Problem der Komplexität

Ein Computer besteht aus vielen unterschiedlichen Hardwarekomponenten – Prozessor, RAM, Grafikkarte, Tastatur, ...

Sollte jedes Programm direkt mit all dieser Hardware kommunizieren müssen?

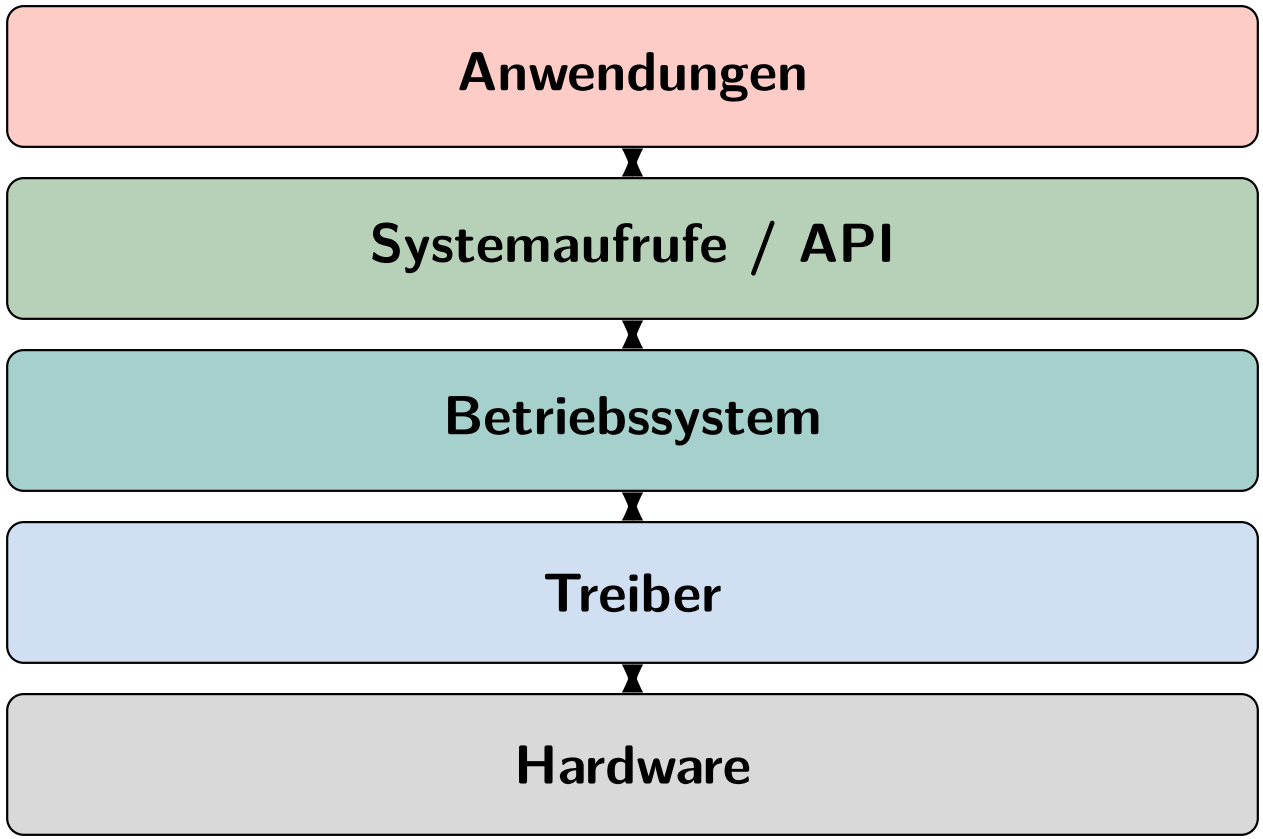
Das Problem der Komplexität

Ein Computer besteht aus vielen unterschiedlichen Hardwarekomponenten – Prozessor, RAM, Grafikkarte, Tastatur, ...

Sollte jedes Programm direkt mit all dieser Hardware kommunizieren müssen?

Nein – dafür gibt es das Betriebssystem.

Der Software-Stack



Jede Schicht abstrahiert die Komplexität der Ebene darunter.

Abstraktion als roter Faden

- Ein Byte abstrahiert von einzelnen Bits (Lektion 4)

Jede Schicht der Informatik baut auf einer einfacheren darunterliegenden Schicht auf.

Abstraktion als roter Faden

- Ein Byte abstrahiert von einzelnen Bits (Lektion 4)
- Ein Gatter abstrahiert von einzelnen Transistoren (Lektion 8)

Jede Schicht der Informatik baut auf einer einfacheren darunterliegenden Schicht auf.

Abstraktion als roter Faden

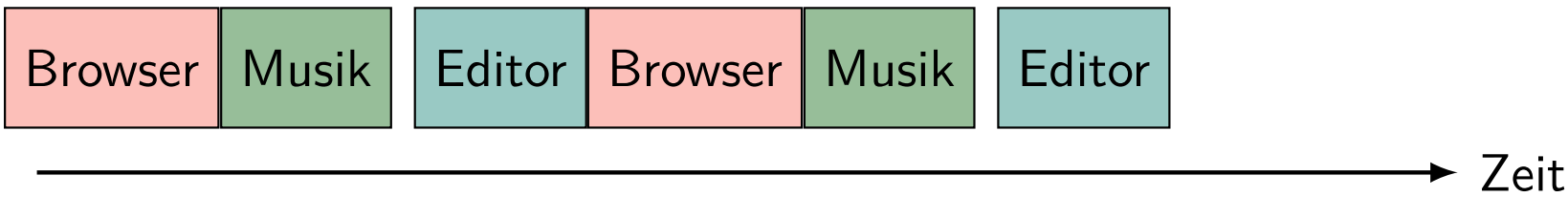
- Ein Byte abstrahiert von einzelnen Bits (Lektion 4)
- Ein Gatter abstrahiert von einzelnen Transistoren (Lektion 8)
- Der Software-Stack abstrahiert von der konkreten Hardware

Jede Schicht der Informatik baut auf einer einfacheren darunterliegenden Schicht auf.

Vier Aufgaben des Betriebssystems

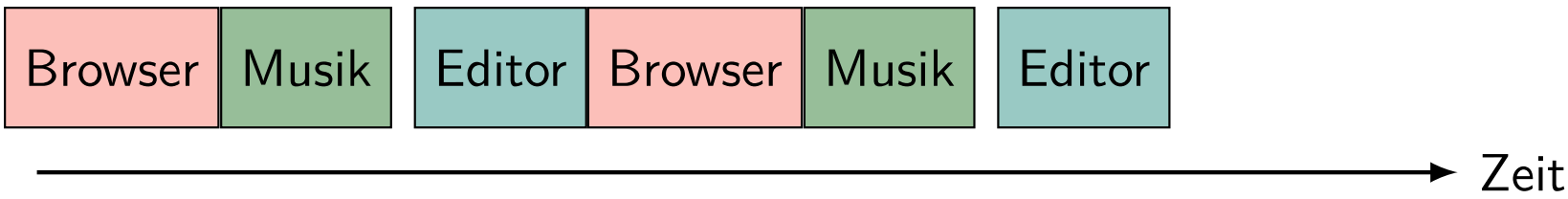
1. ***Prozessverwaltung*** – CPU-Zeit auf mehrere Programme aufteilen
2. ***Speicherverwaltung*** – jedem Prozess einen eigenen RAM-Bereich zuweisen
3. ***Dateisystem*** – Bits in Dateien und Verzeichnisse organisieren
4. ***Geräteverwaltung*** – Treiber für Drucker, Netzwerk, Grafikkarte, ...

Multitasking



Der Prozessor wechselt so schnell zwischen Programmen (***Zeitscheiben***), dass es sich für uns nach Gleichzeitigkeit anfühlt.

Multitasking



Der Prozessor wechselt so schnell zwischen Programmen (***Zeitscheiben***), dass es sich für uns nach Gleichzeitigkeit anfühlt.

Übung: 5 Programme, je 20 ms Zeitscheibe – wann ist ein Programm wieder an der Reihe?

Speicherschutz & Dateisystem

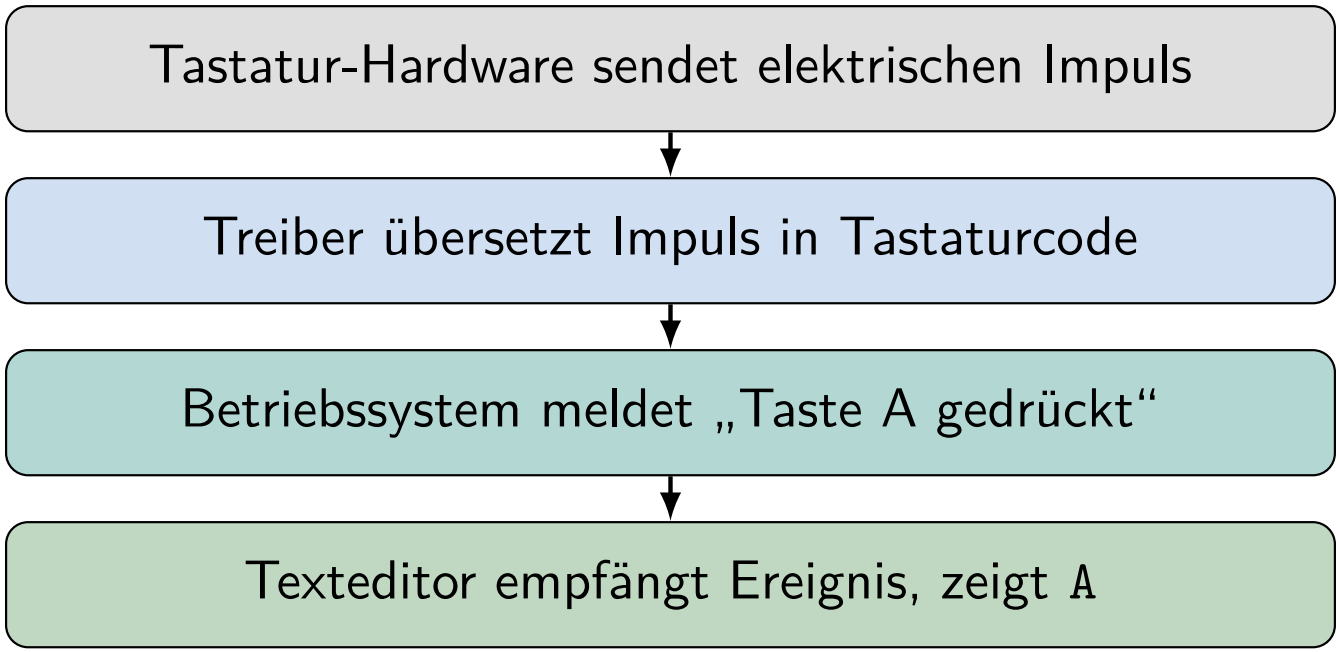
Speicherschutz
Verhindert, dass ein Prozess

in den Speicherbereich eines anderen eingreift – erinnert an das Risiko der Von-Neumann-Architektur (Lektion 10).

Dateisystem Organisiert

rohe Bitsequenzen auf SSD/HDD in Dateien und Verzeichnisse (z. B. NTFS, ext4, APFS).

Vom Tastendruck zum Bildschirm



Jede Schicht hat eine klar definierte Aufgabe.

Bekannte Betriebssysteme

OS	**Hersteller**	**Einsatzgebiet**
Windows 11	Microsoft	PC, Laptops
macOS	Apple	Mac-Computer
Linux	Open Source	Server, eingebettete Systeme
Android	Google	Smartphones, Tablets
iOS	Apple	iPhone, iPad

Über 90 % der Webserver laufen unter Linux – Open-Source-Code, den jede/-r einsehen und verändern kann.

Von der Kerbe im Ishango-Knochen bis zum Betriebssystem

Wir haben den gesamten Weg verfolgt: Zahlendarstellung, Kodierung, logische Gatter, Computer-Hardware und Betriebssystem. Beschreiben Sie in eigenen Worten, was auf allen Ebenen passiert, wenn Sie „Strg+S“ drücken.

Auftrag

Skript „Aufbau und Funktionsweise eines Computers“, Kapitel 6, gesamtes Kapitel

Komponenten dem Software-Stack zuordnen

Challenge: Der ganze Weg von „Strg+S“ bis zur gespeicherten Date