HOCHSCHULE MAINZ
Wirtschaft
Business

Grundlagen der IT
Darstellung und Speicherung von Informationen in Computersystemen

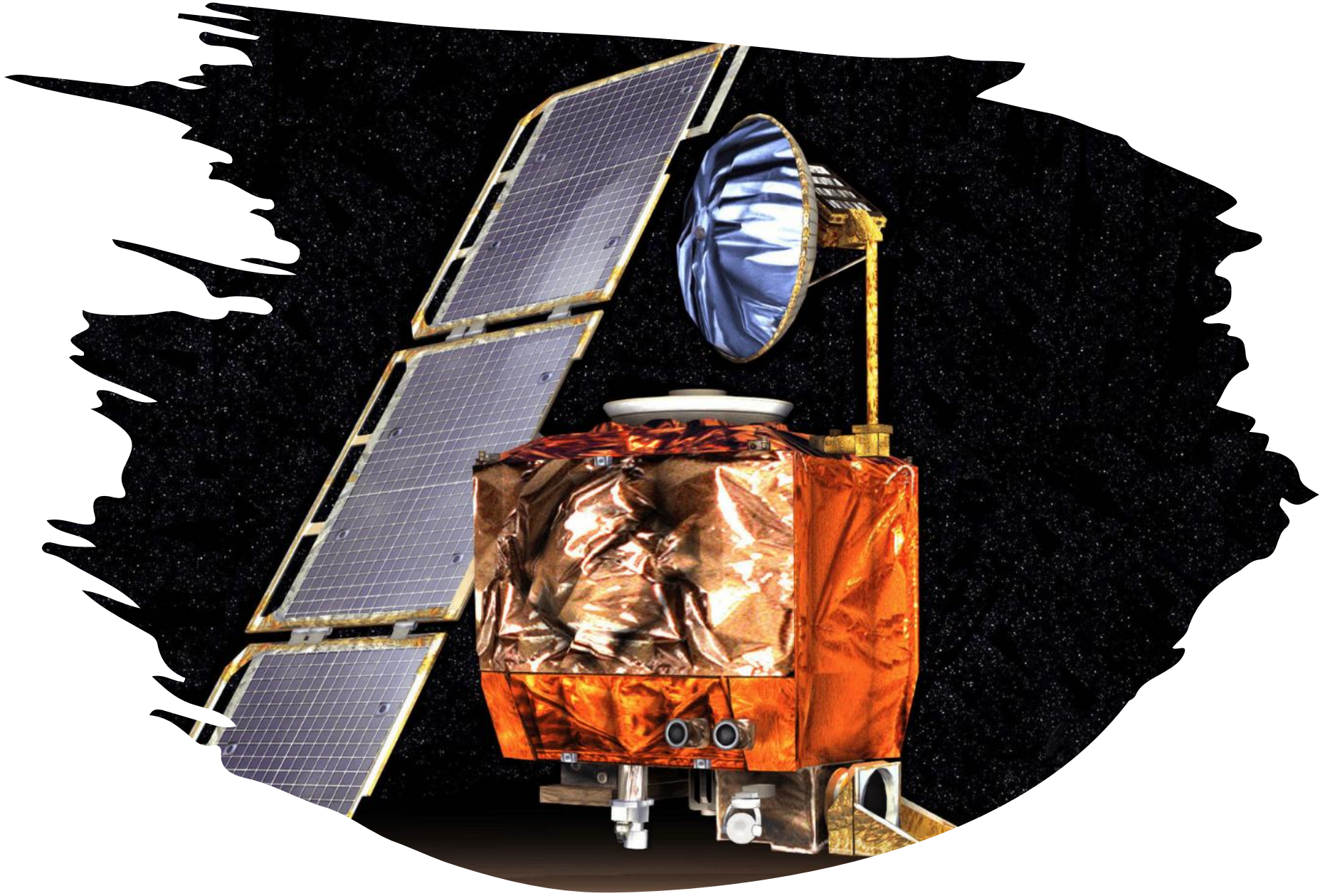
University of
Applied Sciences

Was hat die Ariane 5 mit Speicherung von Informationen zu tun ?

Quelle: ESA

“Just 36 seconds after its maiden launch, the rocket engines failed due to the engineers reusing incompatible code from Ariane 4 and a conversion error from 64-bit to 16-bit data.”

https://raygun.com/blog/costly-software-errors-history

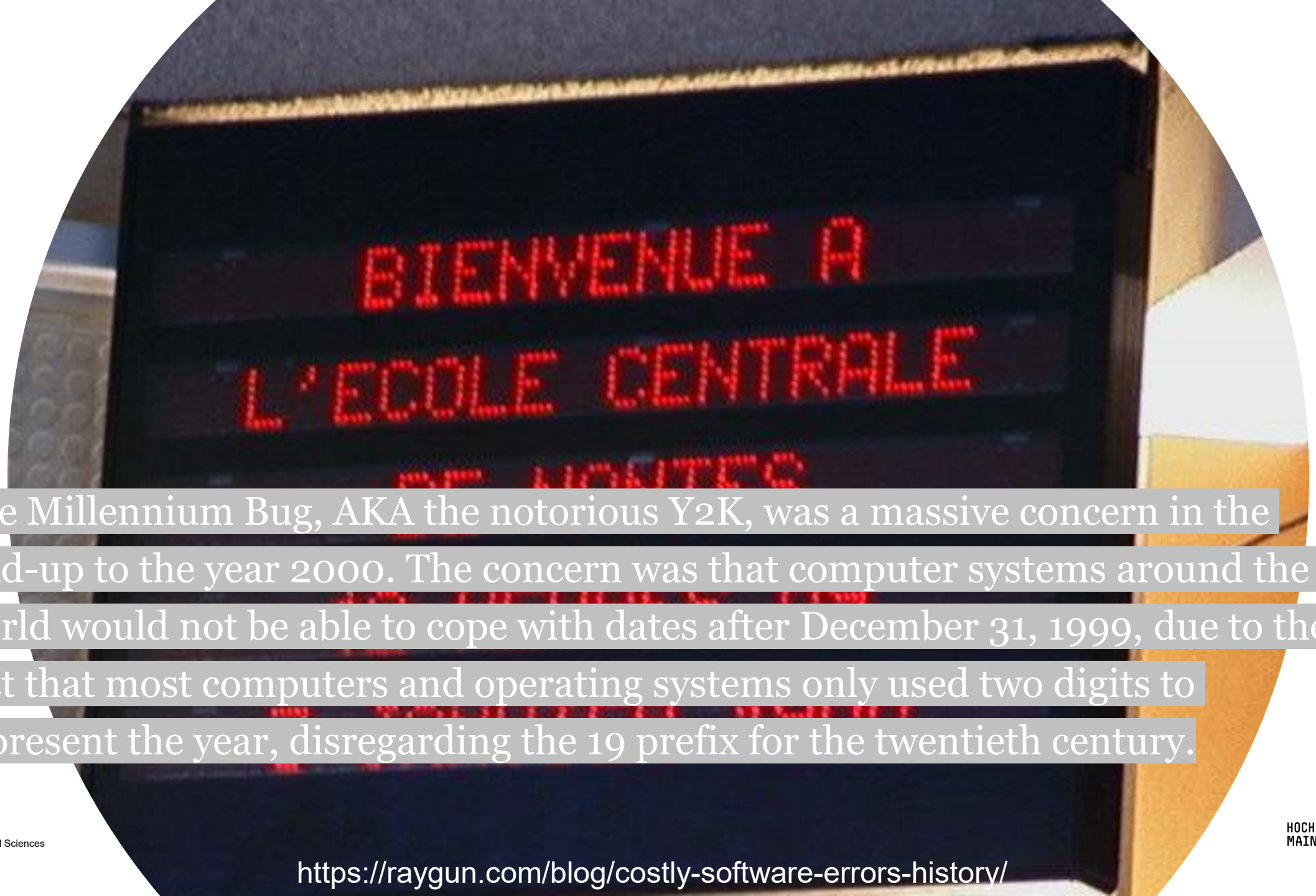


NASA’s Mars Climate Orbiter, 1998

1998 the Mars Climate Orbiter burned up after getting too close to the surface of Mars – “the ground control software produced by Lockheed Martin used imperial measurements, while the software onboard, produced by NASA, was programmed with SI metric units. The overall cost of the failed mission was more than $320 million.”

https://raygun.com/blog/costly-software-errors-history/


BIENVENUE A
L'ECOLE CENTRALE
DE NANTES
12 HEURES 09
3 JANVIER 1900



The Millennium Bug, AKA the notorious Y2K, was a massive concern in the lead-up to the year 2000. The concern was that computer systems around the world would not be able to cope with dates after December 31, 1999, due to the fact that most computers and operating systems only used two digits to represent the year, disregarding the 19 prefix for the twentieth century.

https://raygun.com/blog/costly-software-errors-history/

Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

Es ist insofern wichtig zu wissen, wie Computer Informationen speichern und sie verarbeiten.

Daher besprechen wir das …

… aber wo fangen wir an?

Das Bit

Das Bit

Das Bit

- Die kleinste Informationseinheit, die ein normaler Computer verarbeiten kann ist das Bit (Binary Digit, Binärziffer)
- Der Computer unterscheidet 2 Zustände:
  - 0 kein Strom, keine Spannung
  - 1 Strom, Spannung
- Man kann in einem Bit z. B. eine Binärzahl speichern (eben 0 oder 1)
- Grund: Es ist technisch erheblich einfacher, elektronische Bauteile mit nur jeweils zwei Zuständen (Strom bzw. kein Strom) zu bauen, als weniger Elemente mit z. B. zehn Zuständen (für das Zehnersystem).

Maßeinheiten für Bytes

- 8 Bit sind ein Byte
- Es gibt Kurznamen für große Bytemengen (ähnlich wie Gramm und Kilogramm)
- Die entsprechenden Maßeinheiten für Kilobyte (KByte), Megabyte (MByte) usw. werden im Dualsystem (Faktor $2^{10} = 1024$) und nicht im Dezimalsystem (Faktor $10^3 = 1000$) angegeben
- Achtung bei Herstellerangaben: Eine Festplatte mit einer Größenangabe von 200 GB ($= 200 \cdot 10^9$ Bytes) ist nur 186 GByte groß!

Maßeinheit		Anzahl von Bytes	KBytes	MBytes
Byte		1		
Kilobyte (KByte)	2^{10}	1024	1	
Megabyte (MByte)	2^{20}	1.048.576	1024	1
Gigabyte (GByte)	2^{30}	1.073.741.824	1.048.576	1024
Terabyte (TByte)	2^{40}	1.099.511.627.776	1.073.741.824	1.048.576
Petabyte (PByte)	2^{50}	1.125.899.906.842.624	1.099.511.627.776	1.073.741.824
Exabyte (EByte)	2^{60}	1.152.921.504.606.846.976	1.125.899.906.842.624	1.099.511.627.776

Agenda

1. Zahlensysteme und Ganzzahlenrepräsentation
2. Repräsentation von Brüchen
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. Codierung von Text
5. Bild- und Videoformate
6. Audio-Formate

Agenda

1. **Zahlensysteme und Ganzzahlenrepräsentation**
2. Repräsentation von Brüchen
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. Codierung von Text
5. Bild- und Videoformate
6. Audio-Formate

Zahlensysteme

- Als Beginn der Datenverarbeitung kann die Erfindung von Zahlensystemen und die Verarbeitung von Zahlen angesehen werden.
- Durch die Abbildung auf Zahlen können unterscheidbare Objekte quantifiziert werden (z. B. Anzahl von Reis-Säcken).
- Ermöglicht auch Berechnungen

Zahlensysteme

- **Additive Zahlensysteme:**
  Der Wert einer Zahl wird durch Addieren der Werte ihrer Ziffern errechnet.
- **Stellenwertsysteme:**
  Die (additive) Wertigkeit eines Symbols hängt von seiner Position ab.
- **Hybride Zahlensysteme:**
  - Hierbei wird eine Grundziffer einem Zeichen vorangestellt, das eine Potenz der Basis wiedergibt; die Werte beider werden miteinander multipliziert.
  - In den europäischen Zahlensystemen kamen solche Hybridsysteme so gut wie nicht vor, wohl aber, schon seit Beginn des zweiten Jahrtausends v. Chr., in Mesopotamien, später auch in China und im Nahen Osten allgemein.
  - Sowohl aus Äthiopien als auch aus Südindien und Sri Lanka sowie der Maya-Kultur sind solche hybriden Zahlensysteme bekannt.

Additive Zahlensysteme

- Beispiel: Römische Zahlen
- Verfügbare Ziffern (bzw. unser Alphabet) mit Entsprechung:
  1 = I, 5 = V, 10 = X, 50 = L, 100 = C, 500 = D, 1000 = M
- Die Ziffern I , X und C dürfen einmal, zweimal oder dreimal nebeneinander stehen
- Die Ziffer M kann beliebig oft nebeneinander stehen
- Die Ziffern V , L und D dürfen nicht wiederholt nebeneinander stehen (sie dürfen in einer Zahl nur einmal vorkommen)
- Steht das Zeichen für eine kleinere Einheit rechts neben dem Zeichen einer größeren Einheit, dann wird die kleinere Einheit auf die größere addiert
- Steht kleinere Einheit links einer Zahl, wird subtrahiert
- Es dürfen nicht zwei oder mehrere kleine Einheiten von der rechts stehenden größeren Einheit abgezogen werden.

Additive Zahlensysteme

Wandeln Sie folgende römische Zahlen in die arabische Zahlschrift um:

- VI
- IV
- XXIX
- MCDVI
- MDCCLVI

Zahlensysteme

- **Additive Zahlensysteme:**
  Der Wert einer Zahl wird durch Addieren der Werte ihrer Ziffern errechnet.
- **Stellenwertsysteme:**
  Die (additive) Wertigkeit eines Symbols hängt von seiner Position ab.
- **Hybride Zahlensysteme:**
  Hierbei wird eine Grundziffer einem Zeichen vorangestellt, das eine Potenz der Basis wiedergibt; die Werte beider werden miteinander multipliziert. In den europäischen Zahlensystemen kamen solche Hybridsysteme so gut wie nicht vor, wohl aber, schon seit Beginn des zweiten Jahrtausends v. Chr., in Mesopotamien, später auch in China und im Nahen Osten allgemein. Sowohl aus Äthiopien als auch aus Südindien und Sri Lanka sowie der Maya-Kultur sind solche hybriden Zahlensysteme bekannt.

Stellenwertsysteme

- Beispiel: Arabische Zahlen
- Unser heutiges Zahlensystem stammt aus Indien und gelangte über den nahen Osten zu uns, weshalb man auch heute noch von *arabischen Ziffern* spricht
- *Stellenwertsystem* mit der Basis zehn

Stellenwertsysteme
Ganzzahlen

- Ein Stellenwertsystem mit der Basis B ist ein Zahlensystem, in dem eine Zahl x nach Potenzen von *B* zerlegt wird.
- Dies bedeutet, dass jeder Position in einer Zahl ein bestimmter Wert zugeordnet wird, der eine Potenz von *B* ist.
- Eine natürliche Zahl n wird demnach durch folgende Summe dargestellt:

$$n = \sum_{i=0}^{N-1} b_i \cdot B^i$$

B = Basis des Zahlensystems (B ∈ N, B ≥ 2)
b = Menge der Ziffern ($b_i \in \mathbb{N}_0, 0 \leq b_i < B$)
N = Anzahl der Stellen

Stellenwertsysteme
Ganzzahlen

- Computer verwenden das *Binärsystem*, auch Dualsystem genannt
- Stellenwertsystem mit der Basis zwei
- Kommt also mit zwei Ziffern (0 und 1) aus
- Nachteil: Binärzahlen besitzen bei gleichem Wert erheblich mehr Stellen, da eine Stelle ja nur zwei Werte repräsentieren kann

Binärzahl:	1	0	1	1
Stellenwert:	2^3	2^2	2^1	2^0

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11$$

Stellenwertsysteme
Ganzzahlen

- Häufig arbeiten wir in der Informatik auch mit dem Hexadezimalsystem, Basis B=16
- Grund: Kompakte und kurze Darstellung von Binärzahlen (4 Binärstellen/Bits lassen sich exakt durch eine Hexadezimalziffer darstellen)
- Da es für die „Ziffern“ zehn, elf, …, fünfzehn im Hexadezimalsystem keine eigene Zifferndarstellung gibt, nimmt man hierfür die Buchstaben A, B, C, D, E, F
- Z. B. ACAEF0AA statt 10101100101011101111000010101010 (B =2) oder 2.897.146.026 (B =10)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	1	2	3	4	5	6	7	8
00000	00001	00010	00011	00100	00101	00110	00111	01000

9	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
9	A	B	C	D	E	F	10	11
01001	01010	01011	01100	01101	01110	01111	10000	10001

Stellenwertsysteme
Ganzzahlen

- Umwandlung von Binär in Hexadezimal ist recht simpel:
  - Eine Hexadezimalzahl kann Werte von 0 bis 15 annehmen und daher 16 Werte repräsentieren
  - Dies entspricht genau 4 Bits
- Somit können wir zur Umwandlung eine Binärzahl in 4 Bit-Gruppen aufteilen (von rechts nach links):
  1010 1100 1010 1110 1111 0000 1010 1010
- Anschließend können wir jede Gruppe durch die ihnen entsprechenden Ziffern des Hexadezimalsystems ersetzen (1010 entspricht z. B. immer A)
  A C A E F 0 A A
- Ein Mensch kann sich in der Regel die Zahl ACAEF0AA leichter merken als 10101100101011101111000010101010

Endianness

- Big-Endian und Little-Endian beschreiben die Reihenfolge, in der ein Computer Mehrbyte-Werte (z. B. 16-Bit, 32-Bit, 64-Bit) im Speicher ablegt.
- Steigt bei einer im Speicher abgelegten Zahl die Wertigkeit einer Stelle mit der wachsenden Adresse, dann ist sie im Little-Endian-Format dargestellt.
- Fällt bei einer im Speicher abgelegten Zahl die Wertigkeit einer Stelle mit der wachsenden Adresse, dann ist sie im Big-Endian-Format dargestellt.

Endianness

– Beispiel:

$$(3101)_{10} = 12 * 16^2 + 1 * 16^1 + 13 * 16^0 = (00\ 00\ 0c\ 1d)_{16}$$

– Big Endian speichert den höherwertigen Teil (00) am Anfang der Adresse und Little Endian ist die Umkehrung von Big Endian (speichert (1d) am Anfang der Adresse).

Konvertierung in das Dezimalsystem

- Ein Polynom $p(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{N-1} x^{N-1}$ vom Grad $N - 1$ lässt sich mit Hilfe des Horner-Schemas wie folgt darstellen (hier: $x = B$):

$$p(x) = (\ldots(((b_{N-1} \cdot B + b_{N-2}) \cdot B + b_{N-3}) \cdot B + b_{N-4}) \cdot B + \cdots + b_1) \cdot B + b_0$$

- Wie zuvor definiert, gilt für eine in einem Positionssystem mit der Basis B dargestellte natürliche Zahl n: $n = \sum_{i=0}^{N-1} b_i \cdot B^i$ - dies ist genau ein Polynom vom Grad $N - 1$!

- Mit Hilfe des Horner-Schemas können demnach Konvertierungen in das Dezimalsystem einfach durchgeführt werden.
- Beispiele:
  - $(1210)_8 = \big((1 \cdot 8 + 2) \cdot 8 + 1\big) \cdot 8 + 0 = (648)_{10}$
  - $(754)_8 = (7 \cdot 8 + 5) \cdot 8 + 4 = (492)_{10}$

Übung

Konvertieren Sie folgende Zahlen unter Zuhilfenahme des Hornerschemas in das Dezimalsystem:

a) $(375)_8$
b) $(888)_9$
c) $(ADA)_{16}$
d) $(11001)_2$

Konvertierungen in andere Zahlensysteme

- Für die Umwandlung einer Dezimalzahl x in ein Zahlensystem mit der Basis B kann folgender Algorithmus verwendet werden:

  Do while $x \neq 0$:

  1. $x : B \; = \; y \; \text{Rest} \; z$
  2. mache y zum neuen x und fahre mit Schritt 1 fort

- Die ermittelten Reste z (von unten nach oben nebeneinander geschrieben) ergeben die entsprechende Zahl mit der Basis B
- Zum Beispiel:

$(30)_{10} = ?_2$

x				y		z
30	:	2	=	15	Rest	0
15	:	2	=	7	Rest	1
7	:	2	=	3	Rest	1
3	:	2	=	1	Rest	1
1	:	2	=	0	Rest	1

$(43)_{10} = ?_2$

x				y		z
43	:	2	=	21	Rest	1
21	:	2	=	10	Rest	1
10	:	2	=	5	Rest	0
5	:	2	=	2	Rest	1
2	:	2	=	1	Rest	0
1	:	2	=	0	Rest	1

Die Reste z von unten nach oben nebeneinander geschrieben liefern dann die gesuchte Dualzahl: $(30)_{10} = 11110_2$ $(43)_{10} = (101011)_2$

Übung

Konvertieren Sie folgende Zahlen wie folgt:

- $(445)_{10}$ in das Dualsystem ($B = 2$)
- $(7294)_{10}$ in das Oktalsystem ($B = 8$)
- $(87599)_{10}$ in das Hexadezimalsystem ($B = 16$)
- $(754498)_{10}$ in das Dualsystem ($B = 2$)

Kann man große Zahlen (z. B. $(754498)_{10}$) eventuell auch effizienter konvertieren? Tipp: Hex-Hex!

Agenda

1. Zahlensysteme und Ganzzahlenrepräsentation
2. **Repräsentation von Brüchen**
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. Codierung von Text
5. Bild- und Videoformate
6. Audio-Formate

Stellenwertsysteme
Gebrochenen Zahlen

– Bei gebrochenen Zahlen trennt ein Punkt (bzw. in Deutschland und anderen Ländern ein Komma) in der Zahl den ganzzahligen Teil vom Nachkommateil

$$n = \sum_{i=-M}^{N-1} b_i \cdot B^i$$

B = Basis des Zahlensystems ($B \in \mathbb{N}, B \geq 2$)
b = Menge der Ziffern ($b_i \in \mathbb{N}_0, 0 \leq b_i < B$)
N = Anzahl der Stellen vor dem Punkt (bzw. Komma)
M = Anzahl der Stellen nach dem Punkt (bzw. Komma)

Stellenwertsysteme
Gebrochenen Zahlen

- Beispiel:
  - Dual: 0110 1110 , 0011
  - Hexadezimal: 6 E , 3
  - Dezimal: ?

Stellenwertsysteme
Gebrochenen Zahlen

- 2 Möglichkeiten zur Codierung im Computersystem
- Festpunktdarstellung und Gleitpunktdarstellung

- Eine Festkommazahl ist eine Zahl, die aus einer festen Anzahl von Ziffern vor und nach dem Komma besteht.
- Die Position des Kommas ist also fest vorgegeben, daher der Name.
- Durch Einführen eines eigenen Vorzeichenbits können dann noch positive und negative Zahlen unterschieden werden.

- Haupt-Nachteil der Festpunktdarstellung: Die Position/Stelle des Kommas muss allgemeingültig festgelegt werden. (Problem: unflexibles Format, insbesondere bei der Speicherung sehr großer oder sehr kleiner Zahlen)

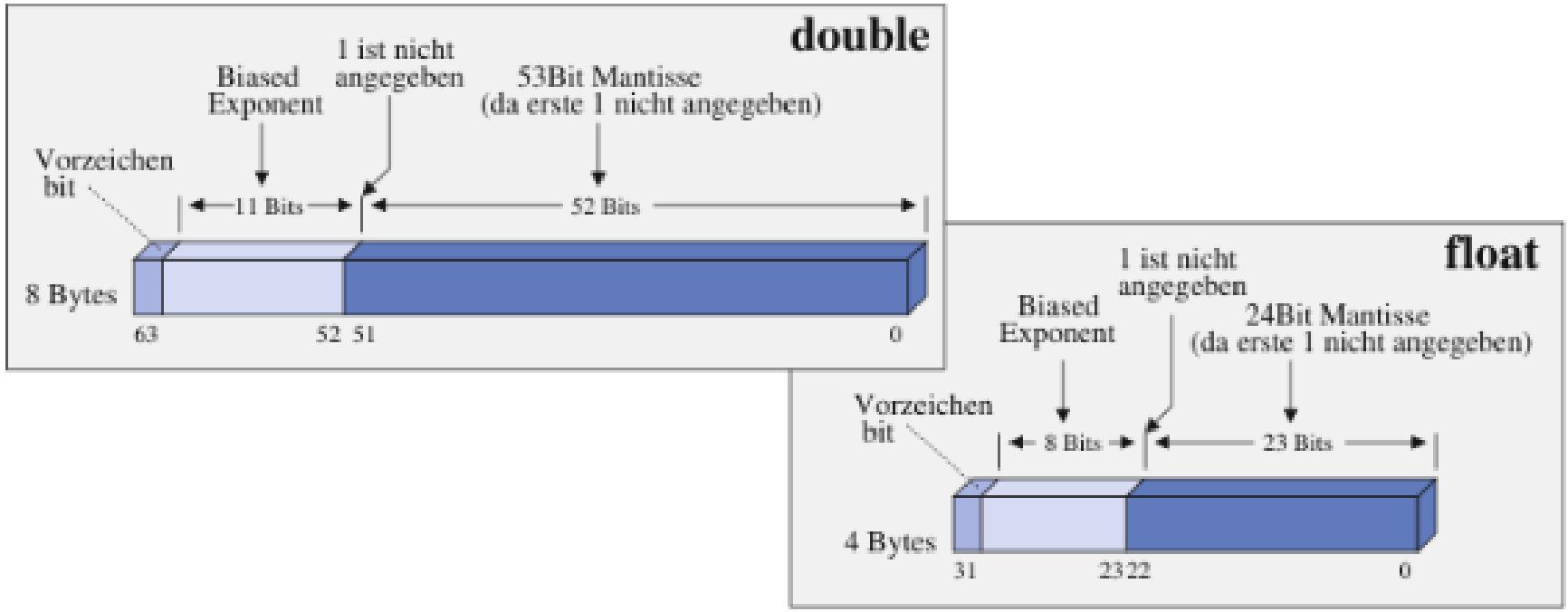
Stellenwertsysteme
Gebrochenen Zahlen

- Gleitpunktdarstellung ist ein standardisiertes IEEE-Format (IEEE 754) der Form $vM \cdot B^N$
- v Vorzeichen-Bit
- M Mantisse
- N ganzzahliger, nichtnegativer Exponent
- z. B.: +1,3756·10^3

- Unterscheidung u. a. zwischen einfacher (Datentyp float, 4 Byte) und doppelter Genauigkeit (Datentyp double, 8 Byte)
- Verfügbar z. B. in C, C++ und Java
- Codierte „Sonderfälle“:

Exponent	Mantisse	Bedeutung
111...111	$\neq 0$	Not a number
111...111	0	$\pm\infty$
0	0	± 0

Stellenwertsysteme
Gebrochenen Zahlen



Konvertierungen in das Dezimalsystem

Für eine in einem Positionssystem mit der Basis B dargestellte echt gebrochene Zahl

$$n{:}\,n = \sum_{i=-M}^{-1} b_i \cdot B^i$$

ist das Horner-Schema

$$n = \frac{1}{B} \cdot \left(b_{-1} + \frac{1}{B} \cdot \left(b_{-2} + \frac{1}{B} \cdot \left(b_{-3} + \ \ldots + \frac{1}{B} \cdot \left(b_{-M+1} + \frac{1}{B} \cdot b_{-M} \right) \ldots \right) \right) \right)$$

Zum Beispiel:

$$0{,}123 \ = \frac{1}{10} \left(1 \ + \frac{1}{10} \left(2 \ + \frac{1}{10} \cdot 3 \right) \right)$$

Gebrochene Zahlen mit beliebiger Basis können somit leicht in das Dezimalsystem transformiert werden.

Konvertierungen in andere Zahlensysteme

- Für die Umwandlung des **Nachkommateils** einer Dezimalzahl in ein anderes Zahlensystem mit Basis B kann folgender Algorithmus verwendet werden:

  Do while $x \neq 0$:

  1. $x \cdot B = y$ Überlauf z (z ist also der ganzzahlige Anteil)
  2. mache Nachkommateil von y zum neuen x und fahre mit Schritt 1 fort

- Die ermittelten Überläufe z (von oben nach unten nebeneinander geschrieben) ergeben die entsprechende Zahl mit der Basis B
- Zum Beispiel:

$(0.34375)_{10} = (0.01011)_2$

x		y	z
0.34375	· 2 =	0.6875	Ueberl. 0
0.6875	· 2 =	1.375	Ueberl. 1
0.375	· 2 =	0.75	Ueberl. 0
0.75	· 2 =	1.5	Ueberl. 1
0.5	· 2 =	1.0	Ueberl. 1
0	· 2 =	0.0	Ueberl. 0

Die Überläufe z von oben nach unten nach 0. nebeneinander geschrieben liefern dann die gesuchte Zahl.

$(0.408203125)_{10} = (0.321)_8$

x		y	z
0.408203125	· 8 =	3.265625	Ueberl. 3
0.265625	· 8 =	2.125	Ueberl. 2
0.125	· 8 =	1	Ueberl. 1
0	· 8 =	0	Ueberl. 0

Konvertierungen in andere Zahlensysteme

- Achtung: Die Konvertierung geht nicht immer genau auf!
- Manche gebrochenen Zahlen, die sich ganz genau im Dezimalsystem darstellen lassen, lassen sich leider nicht ganz genau als Dualzahl darstellen.
- Beispiel $(0{,}1)_{10}$

x	y	z
$0.1 * 2 = 0.2$	Überlauf	
$0.2 * 2 = 0.4$	Überlauf	
$0.4 * 2 = 0.8$	Überlauf	
$0.8 * 2 = 1.6$	Überlauf	
$0.6 * 2 = 1.2$	Überlauf	
$0.2 * 2 = 0.4$	Überlauf	
$0.4 * 2 = 0.8$	Überlauf	
$0.8 * 2 = 1.6$	Überlauf	
$0.6 * 2 = 1.2$	Überlauf	

  Das Bitmuster 0011 wiederholt sich hier ständig.
- Wir müssen daher definieren, wie viele Nachkommastellen wir speichern möchten (also die Genauigkeit)

Konvertierungen in andere Zahlensysteme

<u>Achtung:</u> Bei der Umwandlung von Dezimalzahlen zu Binärzahlen kann es leicht zu Rundungsfehlern kommen. Daher sollte man grundsätzlich float- oder double-Werte niemals auf <u>exakte</u> Gleichheit prüfen!

Agenda

1. Zahlensysteme und Ganzzahlenrepräsentation
2. Repräsentation von Brüchen
3. **Binäre Arithmetik und Repräsentation von negativen Zahlen**
4. Codierung von Text
5. Bild- und Videoformate
6. Audio-Formate

Binäre Addition

Rechenregeln:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 Übertrag 1
- 1 + 1 + 1 (vom Übertrag) = 1 Übertrag 1

Zum Beispiel:

```
  0 1 0 1 1 0 1 =  45
  0 1 1 0 1 1 0 =  54
-----------------------
  1 1 1 1       = Übertrag
-----------------------
  1 1 0 0 0 1 1 =  99
```

Übung

Addieren Sie die folgenden Dualzahlen:

1)

```
  0 1 0 1 1 0 1
+ 0 0 0 1 0 1 1
+ 0 0 1 0 0 0 1
+ 0 0 0 1 0 1 0
```

2)

```
  0 1 1 0 0 1
+ 0 0 1 1 0 0
+ 0 0 0 0 1 1
+ 0 0 1 0 0 1
```

3)

```
  0 1 1 0 0 1 0
+ 0 0 1 1 0 1 0
+ 0 0 0 1 1 0 0
+ 0 0 1 0 0 1 1
```

4)

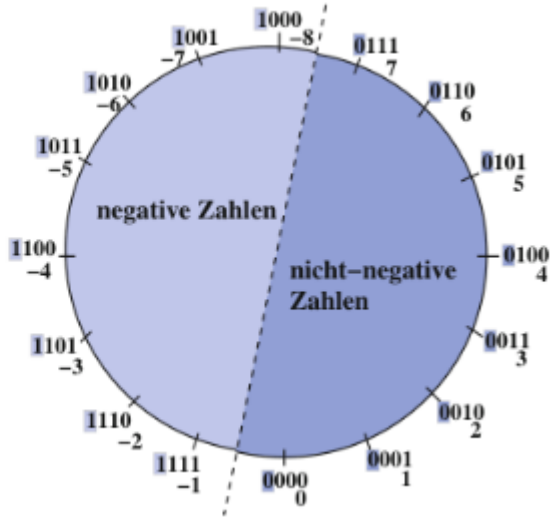
```
  0 1 0 1 1 0 1
+ 0 0 0 1 1 1 1
+ 0 0 0 1 0 0 0
+ 0 0 1 0 1 0 1
+ 0 0 0 1 1 0 1
```

Negative Ganzzahlen und Subtraktion

- Um im Computer mit einem reinen Addierwerk auszukommen, versucht man, die Subtraktion auf eine Addition zurückzuführen
- Umsetzung: Verfahren der Zweier-Komplementbildung
- Visualisierung über einen „Zahlenring“ (hier für 4 Bits):

Alle Kombinationen, bei denen das 1. Bit (Vorzeichenbit) gesetzt ist, repräsentieren dabei negative Zahlen:

0000 = 0	1000 = −8
0001 = 1	1001 = −7
0010 = 2	1010 = −6
0011 = 3	1011 = −5
0100 = 4	1100 = −4
0101 = 5	1101 = −3
0110 = 6	1110 = −2
0111 = 7	1111 = −1



- Wertebereich mit s Bits: $-2^{s-1} = -8$ bis $-2^{s-1} - 1 = 7$

Negative Ganzzahlen und Subtraktion

- Regeln für die Bildung eines Zweier-Komplements:
  1. Ist das 1. Bit mit 1 besetzt, so handelt es sich um eine negative Zahl.
  2. Der Wert einer negativen Zahl wird dabei im Zweier-Komplement dargestellt. Zweier-Komplement zu einem Wert bedeutet dabei, dass zunächst jedes einzelne Bit invertiert (umgedreht) wird, und dann auf die so entstandene Bitkombination die Zahl 1 aufaddiert wird.

Zweier-Komplement zu 5:

Dualdarstellung von 5: 0101

Komplementieren von 5: 1010
+ 1: 0001

= −5: 1011

Zweier-Komplement zu -5:

Dualdarstellung von −5: 1011

Komplementieren von −5: 0100
+ 1: 0001

= 5: 0101

- Vorteil: Computer müssen nicht subtrahieren; jede Subtraktion $a - b$ kann durch eine Addition $a + -b$ ausgedrückt werden.

Negative Ganzzahlen und Subtraktion

- Problem: „Überlauf“

```
6 - 2 = 6 + -2
    0110 =   6
  + 1110 =  -2
  ---------
   10100 =   4
```

- das überlaufende Bit wird verworfen
- problematisch?

Negative Ganzzahlen und Subtraktion

– Problem: „Überlauf“

$$
\begin{array}{rl}
\multicolumn{2}{l}{\mathbf{6 - 2 = 6 + -2}} \\
0110 & = \quad 6 \\
+\ 1110 & = -2 \\
\hline
1|0100 & = \quad 4
\end{array}
$$

– das überlaufende Bit wird verworfen
– problematisch?

$$
\begin{array}{rl}
-(9)_{10}: & (10111)_2 \\
+\ (-13)_{10}: & (10011)_2 \\
\hline
(+10)_{10}: 1| & (01010)_2
\end{array}
$$

Überlauf

- Wird versucht, in einem Datentyp einen Wert abzulegen, der nicht in diesen Datentyp passt, so werden einfach die vorne überhängenden Dualziffern abgeschnitten.
- in Programmiersprachen wie C/C++ und auch Java werden beim Abspeichern von Zahlen, die außerhalb des Wertebereichs eines Datentyps liegen, kein Fehler gemeldet
- Die überhängenden Bits werden abgeschnitten!
- Mit diesem falschen Wert wird dann einfach weiter gearbeitet, was schließlich zu falschen Ergebnissen führt.
- Beim Entwurf eines Programms sollte also genau darauf geachtet werden, dass die während des Programmablaufs zu erwartenden Zahlen niemals außerhalb der Wertebereiche der dafür gewählten Datentypen liegen (oder neuere Implementierungen wählen).
- Beispiel:

```
public static void main(String[] args) {
    int x = Integer.MAX_VALUE;
    System.out.println(x);      // 2147483647
    x++;                        // Überlauf
    System.out.println(x);      // -2147483648  (MIN_VALUE)
}
```

```
x=Math.addExact(x, 1);
Exception in thread "main" java.lang.ArithmeticException: integer overflow
    at java.base/java.lang.Math.addExact(Math.java:911)
    at Main.main(Main.java:8)
```

Typische Wertebereiche für die Datentypen auf 32-Bit-Architekturen

Datentyp-Bezeichnung	Bitzahl	Wertebereich
`char, signed char`	8	–128...127
`unsigned char`	8	0...255
`short, signed short`	16	–32 768...32 767
`unsigned short`	16	0...65 535
`int, signed int`	32	–2 147 483 648...2 147 483 647
`unsigned, unsigned int`	32	0...4 294 967 295
`long, signed long`	32	–2 147 83 648...2 147 483 647
`unsigned long`	32	0...4 294 967 295
`float`	32	$1.2 \cdot 10^{-38}$...$3.4 \cdot 10^{38}$
`double`	64	$2.2 \cdot 10^{-308}$...$1.8 \cdot 10^{308}$
`long double`	96	$3.4 \cdot 10^{-4\,932}$...$1.1 \cdot 10^{4\,932}$

– 64 Bit-Architektur: weitgehend gleich, aber z. B. long ist in der Regel 64 Bit und long double ist 128 Bit lang (zugehörige Wertebereiche sind entsprechend größer)

Multiplikation und Division

- ganzzahlige Multiplikation bzw. Division kann in einem Computer mittels wiederholter Addition durchgeführt werden
  z. B. 3*11 = 11+11+11
- in den Sonderfällen des Multiplikators bzw. Divisors von 2, 4, 8, … kann die Multiplikation bzw. Division einfach auch durch eine Verschiebung von entsprechend vielen Bits nach links bzw. rechts erfolgen: Bei 2 um ein Bit, bei 4 um 2 Bits, bei 8 um 3 Bits usw.
- Analog wie im schriftlichen multiplizieren des dezimalen Systems („Shift-and-Add“):
  1. Schrittweise Bitweises addieren
  2. alles addieren (nach Wertigkeit)
- In modernen CPUs gibt es dedizierte Multiplikationseinheiten (paralleler Ablauf und spezielle Schaltungen)

Agenda

1. Zahlensysteme und Ganzzahlenrepräsentation
2. Repräsentation von Brüchen
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. **Codierung von Text**
5. Bild- und Videoformate
6. Audio-Formate

Codierung von Zeichen - ASCII

- ASCII = American Standard for Coded Information Interchange
- ASCII-Code ist eine Norm zur binären Kodierung von Zeichen
- Kodierung erfolgt in 7 Bits (1 Bit ungenutzt), so dass mit dem Standard ASCII-Code 128 verschiedene Zeichen dargestellt werden können
- Von den 128 Zeichen sind 33 nicht druckbar (z. B. Tabulatorzeichen, Escape, Protokollzeichen wie Übertragungsende)
- Speziell normierte ASCII-Code-Erweiterungen nutzen das erste Bit, um zusätzlich weitere 128 Zeichen darstellen zu können

Codierung von Zeichen - ASCII

– 7 Bit

ASCII-Zeichentabelle, hexadezimale Nummerierung

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1…	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2…	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Codierung von Zeichen - ASCII

– 8. Bit am Beispiel der Windows Codepage 1252 (westeuropäische Windows Zeichensatztabelle)

8…	€		‚	*ƒ*	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9…		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
A…	*NBSP*	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	*SHY*	®	¯
B…	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C…	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D…	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E…	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F…	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Codierung von Zeichen - ASCII

– Problem: Viele verschiedene inkompatible Codepages

874	Thai
932	Japanisch
936	Vereinfachtes Chinesisch
949	Koreanisch
950	Traditionelles Chinesisch
1200	Unicode UTF-16, little endian
1201	Unicode UTF-16, big endian
1250	Mitteleuropäisch
1251	Kyrillisch
1252	Westeuropäisch
1253	Griechisch
1254	Türkisch
1255	Hebräisch
1256	Arabisch
1257	Baltisch
1258	Vietnamesisch

Codierung von Zeichen - Unicode

- Unicode strebt die möglichst vollständige Erfassung aller bekannten Zeichen aus gegenwärtigen und vergangenen Schriftkulturen an (inklusive Steuerzeichen, mathematischer Formelzeichen, Emojis, …)
- Version 17.0 definiert 159.801 Zeichen, darunter auch 3.790 Emojis

1F600	😀	grinning face
1F601	😁	grinning face with smiling eyes
1F602	😂	face with tears of joy
1F603	😃	smiling face with open mouth (cf. ☺)
1F604	😄	smiling face with open mouth and smiling eyes
1F605	😅	smiling face with open mouth and cold sweat

Codierung von Zeichen - Unicode

- UTF-8 ist die am weitesten verbreitete Kodierung für Unicode-Zeichen
- Im April 2023 verwendeten 97,9 % aller Websites UTF-8
- Bei der UTF-8-Kodierung wird jedem Unicode-Zeichen eine Byte-Kette mit einer Länge von einem bis zu vier Byte zugeordnet
- Die ersten 128 Unicodezeichen werden durch ein Byte dargestellt (entspricht 7-Bit-ASCII)
- Englischsprachige Texte lassen sich daher im Regelfall sogar mit nicht-UTF-8-fähigen Texteditoren ohne Beeinträchtigung bearbeiten.
- zusätzliche Zeichen in europäischen Sprachen werden mit zwei Byte codiert – Texte in solchen Sprachen benötigen in Summe nur wenig mehr als ein Byte pro Zeichen
- Zeichen aus indischen und fernöstlichen Schriften benötigen 3 Byte und seltene Zeichen 4 Byte

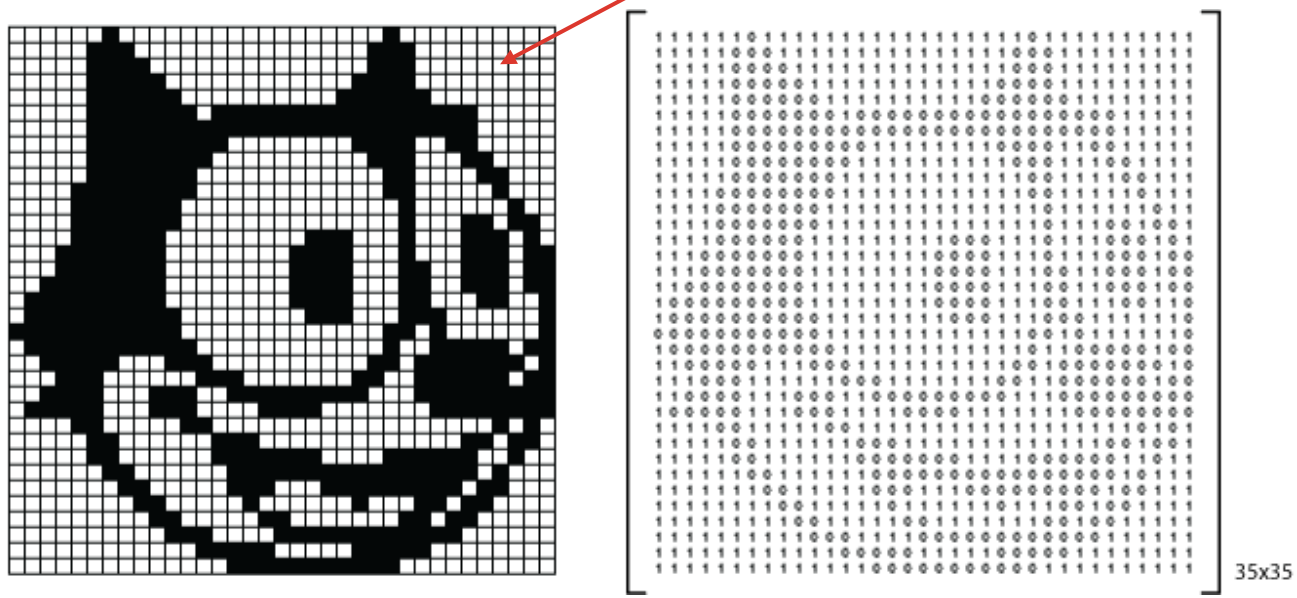

01001101 01100001
01110101 01110011
(Maus)
Das war Binärcode

Agenda

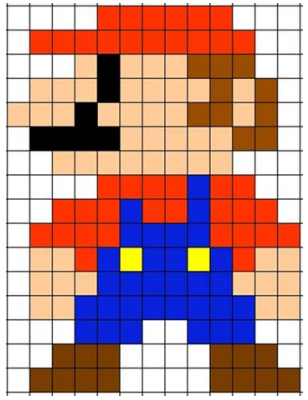
1. Zahlensysteme und Ganzzahlenrepräsentation
2. Repräsentation von Brüchen
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. Codierung von Text
5. **Bild- und Videoformate**
6. Audio-Formate

Bilder

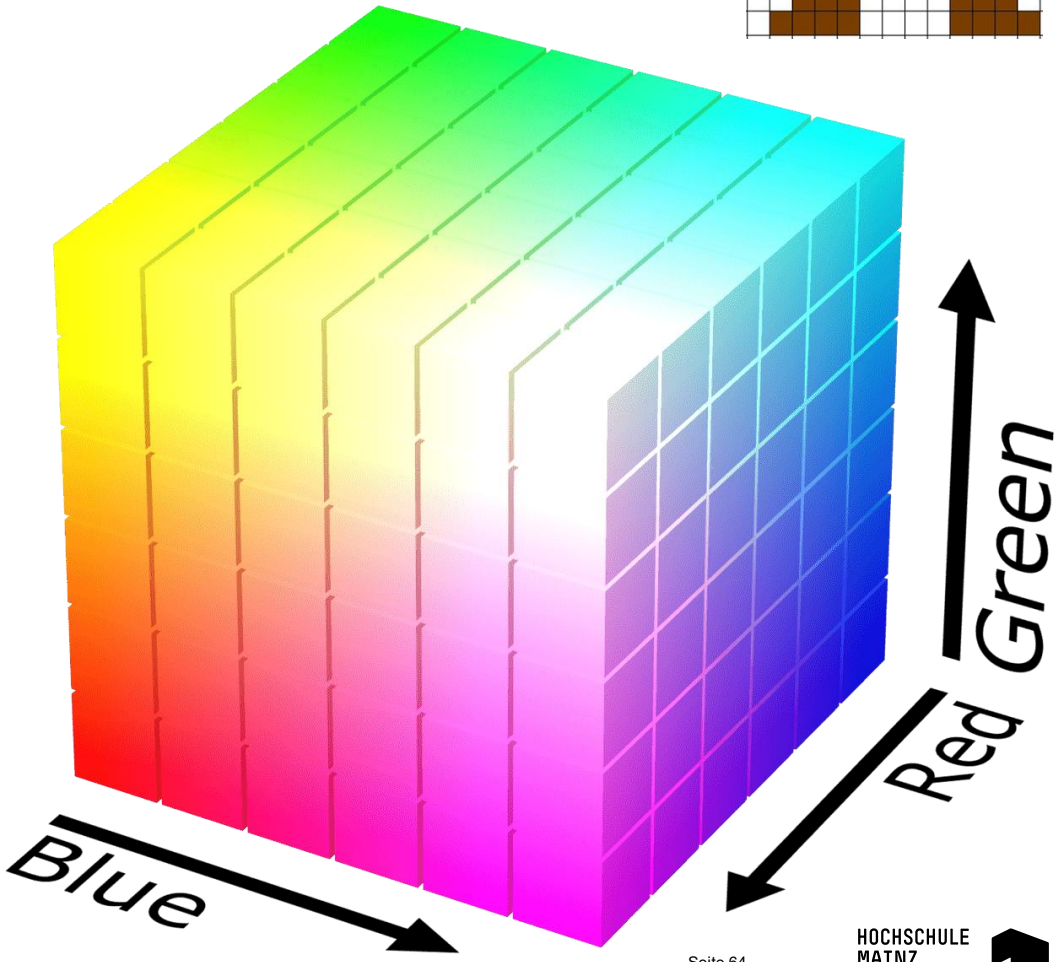
- Bilder werden als Raster- oder Vektorgrafiken gespeichert
- Bei Rastergrafiken wird das Bild aufgeteilt in ein Raster, jeder Bildpunkt in dem Raster heißt Pixel
- Wir speichern insbesondere die Dimensionen des Bildes (Höhe und Breite), die Bildauflösung
- Z. B. 1920x1080 Pixel = Full HD
- die Auflösung einer Bitmap bestimmt ihre Detailgenauigkeit und Klarheit
- Zudem speichern wir die Farbwerte jedes Pixels
- Im einfachsten Fall binär



Bilder



- Wir können Farben speichern
- Die Farbtiefe definiert, wie viele verschiedene Farben ein Pixel annehmen kann.
- wird in Bits pro Pixel (bpp) gemessen
- 8 Bit können 256 Farben und 24 Bit können 16 Millionen Farben speichern
- Beispiel RGB:
  - Speicherung des Anteils von Rot, Grün und Blau
  - 8 Bit je Farbe bei 24 Bit Farbtiefe
  - Farbwahrnehmung durch das additive Mischen dieser Grundfarben

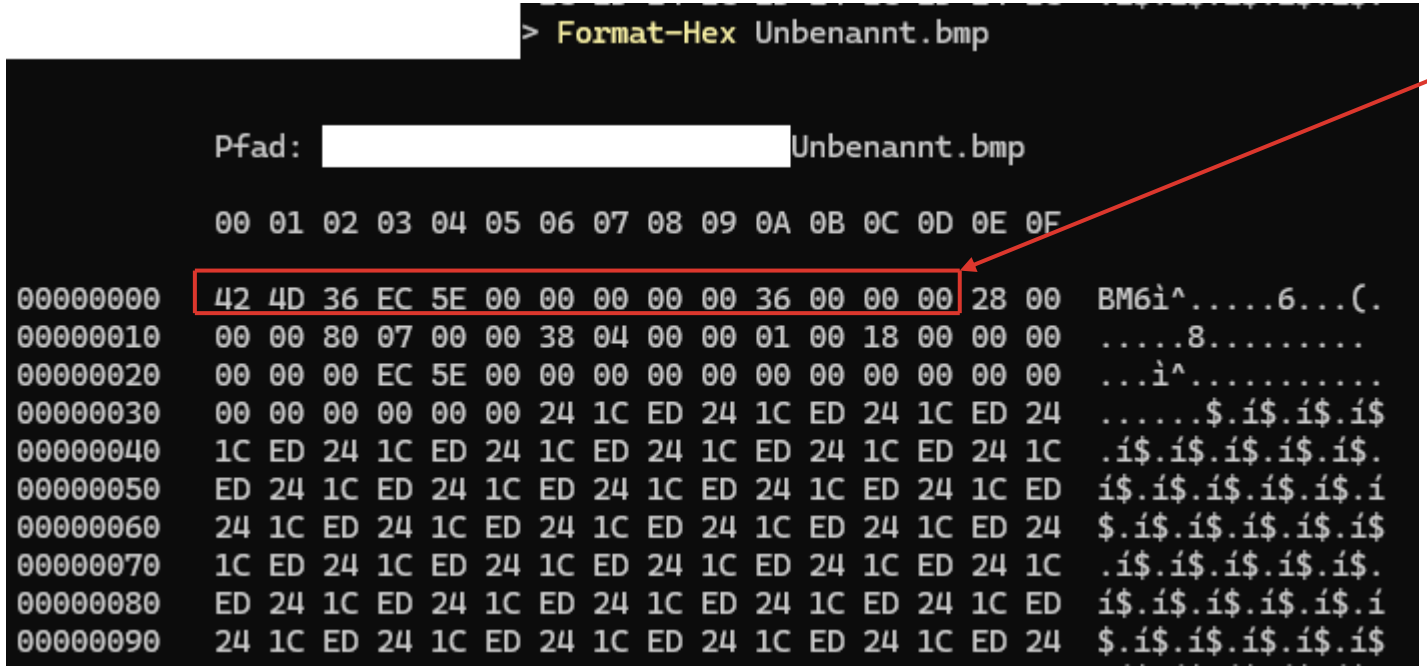


Bilder

– Wie groß ist eine 1920x1080 Pixel große Bitmap mit 24 Bit Farbtiefe?

Bilder

– Und wie könnte das Bild aussehen? (Hier in Hex-Code)



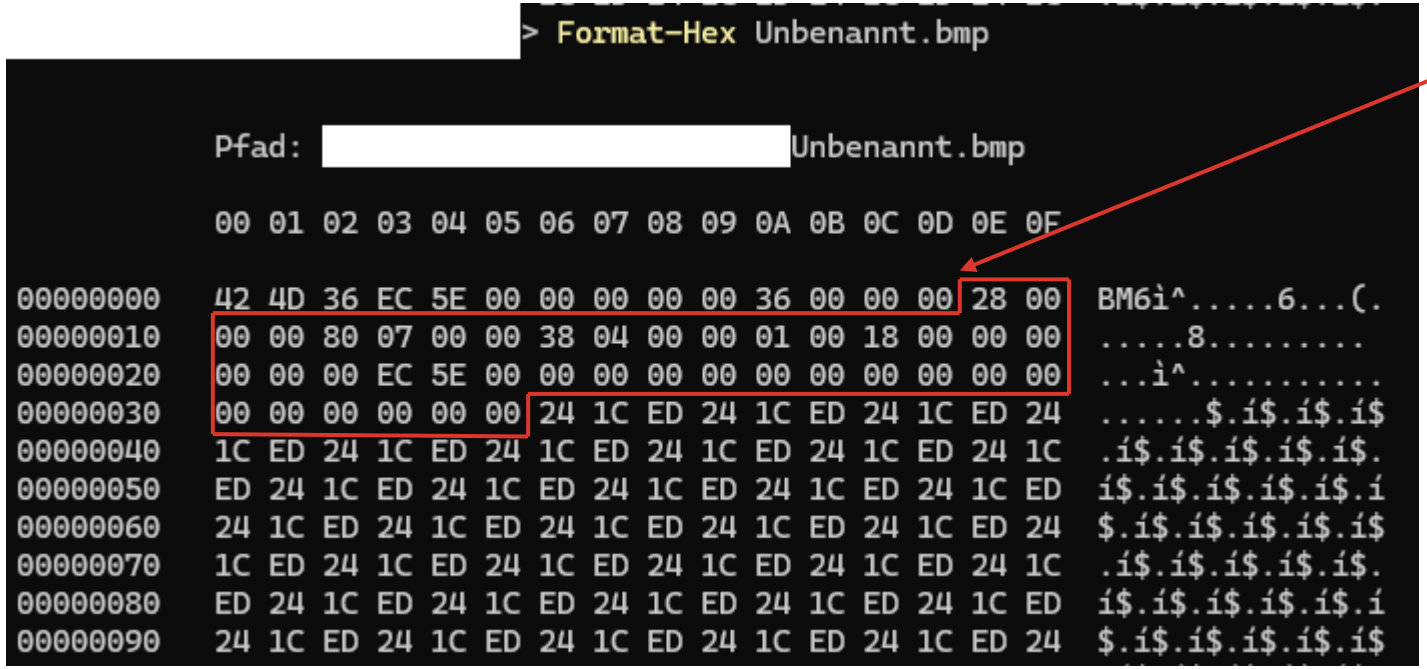
File-Header:

Enthält Informationen über das Bild

- 42 4D = ASCII für „BM“ => Signatur für BMP-Dateien
- 36 EC 5E 00 steht für die Dateigröße und ist im little Endian Format gespeichert, also 6.220.854 Bytes
- 4 Bytes Default-Null-Werte (reserviert)
- 36 00 00 00 ist der Offset zu den Pixel-Daten, hier 54 Byte (14 für den File Header und 40 für den Info-Header)

Bilder

– Und wie könnte das Bild aussehen? (Hier in Hex-Code)



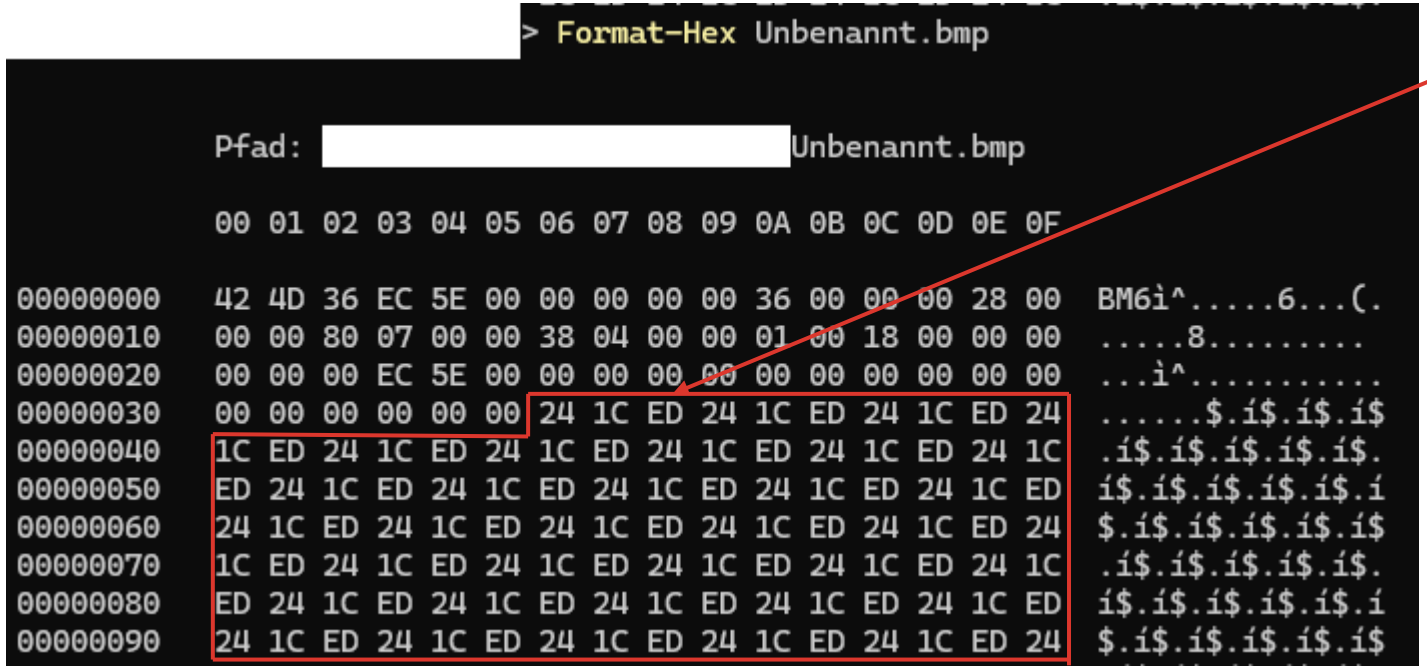
```
> Format-Hex Unbenannt.bmp

        Pfad:                          Unbenannt.bmp

             00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000     42 4D 36 EC 5E 00 00 00 00 00 36 00 00 00 28 00  BM6ì^.....6...(.
00000010     00 00 80 07 00 00 38 04 00 00 01 00 18 00 00 00  .....8.........
00000020     00 00 00 EC 5E 00 00 00 00 00 00 00 00 00 00 00  ...ì^...........
00000030     00 00 00 00 00 00 24 1C ED 24 1C ED 24 1C ED 24  ......$.í$.í$.í$
00000040     1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C  .í$.í$.í$.í$.í$.
00000050     ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED  í$.í$.í$.í$.í$.í
00000060     24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24  $.í$.í$.í$.í$.í$
00000070     1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C  .í$.í$.í$.í$.í$.
00000080     ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED  í$.í$.í$.í$.í$.í
00000090     24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24  $.í$.í$.í$.í$.í$
```

Info-Header:

- 28 00 00 00 = Größe des Info-Headers (hier 40 Byte)
- 80 07 00 00 = Bildbreite in Pixel (hier 1.920 Pixel)
- 38 04 00 00 = Bildhöhe in Pixel (hier 1080 Pixel)
- 01 00 = Anzahl Bild-Ebenen (in der Regel 1)
- 18 00 = Bits pro Pixel = 24, (8 Bit je Kanal)
- …

Bilder

– Und wie könnte das Bild aussehen? (Hier in Hex-Code)



```
> Format-Hex Unbenannt.bmp

        Pfad:                          Unbenannt.bmp

             00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000     42 4D 36 EC 5E 00 00 00 00 00 36 00 00 00 28 00  BM6ì^.....6...(.
00000010     00 00 80 07 00 00 38 04 00 00 01 00 18 00 00 00  .....8.........
00000020     00 00 00 EC 5E 00 00 00 00 00 00 00 00 00 00 00  ...ì^...........
00000030     00 00 00 00 00 00 24 1C ED 24 1C ED 24 1C ED 24  ......$.í$.í$.í$
00000040     1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C  .í$.í$.í$.í$.í$.
00000050     ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED  í$.í$.í$.í$.í$.í
00000060     24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24  $.í$.í$.í$.í$.í$
00000070     1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C  .í$.í$.í$.í$.í$.
00000080     ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED  í$.í$.í$.í$.í$.í
00000090     24 1C ED 24 1C ED 24 1C ED 24 1C ED 24 1C ED 24  $.í$.í$.í$.í$.í$
```

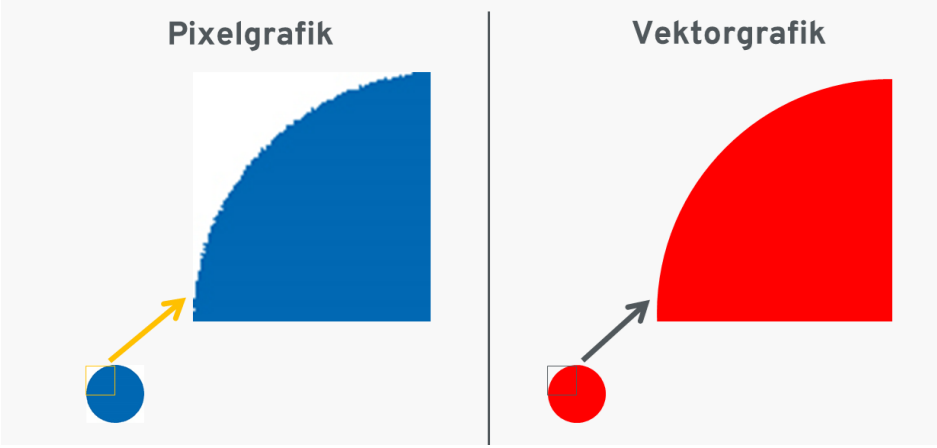
Pixel-Werte:

- Byte 1 24 = Blau (B), hier also 36
- Byte 2 1C = Grün (G), hier also 28
- Byte 3 ED = Rot (R), hier also 237
- Sehr hoher Rotwert, ganz wenig Grün und Blau
- In unserem Beispiel: Wie sieht das Bild aus?

Bilder

- Kompression verringert Dateigröße
- Verlustfreie Kompression, z. B. PNG (Portable Network Graphics)
  - Anwendung von Filtern (z. B. „nur Unterschiede zwischen Pixeln speichern“)
  - Bilddaten werden mit Deflate (ähnlich ZIP) komprimiert
  - Größenersparnis ca. 20–70 % (je nach Bild)
- Verlustbehaftete Kompression, z. B. JPEG (Joint Photographic Experts Group)
  - Bild in 8x8-Pixel-Blöcke zerlegen
  - Jeder Block wird in Frequenzen zerlegt (diskrete Kosinustransformation)
  - Für den Menschen kaum wahrnehmbare Details oder schwer wahrnehmbare Frequenzen werden entfernt
  - Restdaten werden effizient kodiert (z. B. Huffman Encoding).
  - Größenersparnis ca. 80–95%, Qualität kann angepasst werden (mehr Kompression => kleinere Datei, aber sichtbare Artefakte)

Bilder



- Alternative: Vektorgrafiken
- bestehen aus mathematisch beschriebenen Formen (Linien, Kurven, Flächen, Text)
- Beispiel: „Zeichne einen Kreis mit Radius 50 und roter Füllung“ statt „speichere 10.000 rote Pixel“
- Vorteile:
  - beliebig skalierbar ohne Qualitätsverlust (geeignet für Logos, Symbole, Diagramme, Schriften, …)
  - Für einfache Bilder kleine Dateigrößen
- Nachteil:
  - nicht gut geeignet für realistische Fotos (zu viele Details, dann extrem viele Formen und sehr große Files)
- Z. B. .svg, .eps, .pdf

Filme

- Filme bestehen zunächst aus einem Container:
  Datenstruktur, mit der einzelne Datenströme verschiedener Formate zu einem Datenstrom zusammengeführt werden („Verpackung“ für Video-, Audio- und Untertitelspuren und Metadaten wie Kapitel)
- Verbreitet ist z. B. der MP4-Container – Standard für fast alles (Streaming, Social Media, Player)

Filme

- Das Video ist eine Reihe von Bildern („Frames“, normalerweise Rasterbilder)
- Die Videospur wird mit Video-Codecs komprimiert
- ein Video-Codec ist ein Algorithmenpaar zur Kodierung und Dekodierung von digitalen Videos
- er bestimmt Qualität und Dateigröße der Videospur
- Weit verbreitet sind z. B.
  - H.264 (AVC) – weit verbreitet (z. B. Instagram, TikTok, WhatsApp, …), Faktor 50–200 kleiner als uncodiertes Bild
  - H.265 (HEVC) – Effizienter (spart ~40 % gegenüber H.264), Standard für 4K/UHD-Streaming (Netflix, Apple TV)
  - AV1 – Offener, lizenzfreier Codec, spart ca. ~30–50 % gegenüber H.264; die Kompression verbessert sich bei höheren Auflösungen (FHD ≈ ~33 %, 8K bis ~80 %)
    => daher zukünftig wichtig für 8K und energiesparendes Streaming

Filme

- Kompression z. B. über
- Intraframe-Kompression:
  - Jeder Frame wird individuell verkleinert (ähnlich wie bei PNG oder JPEG)
  - Einfacher später zu verarbeiten, aber weniger effiziente Kompression
- Interframe-Kompression:
  - Einige Frames werden komplett gespeichert („Keyframes“)
  - Andere Frames speichern nur die Unterschiede zum letzten Keyframe („Deltaframes“)

Agenda

1. Zahlensysteme und Ganzzahlenrepräsentation
2. Repräsentation von Brüchen
3. Binäre Arithmetik und Repräsentation von negativen Zahlen
4. Codierung von Text
5. Bild- und Videoformate
6. **Audio-Formate**

High Pitch Challenge!

Achtung:

Wir hören gleich hohe Töne – wenn Sie Beschwerden mit dem Gehör haben sollten (Tinitus o. ä.) oder empfindlich reagieren, **gehen Sie bitte kurz raus**.

https://www.youtube.com/watch?v=rV4O2TxpDd4

High Pitch Challenge!

Obergrenze der Hörfähigkeit nach Alter:

- 5 Jahre: 21 kHz
- **20 Jahre: 18 kHz**
- **35 Jahre: 15 kHz**
- 50 Jahre: 12 kHz
- 65 Jahre: 9 kHz
- 80 Jahre: 5 kHz

Audio

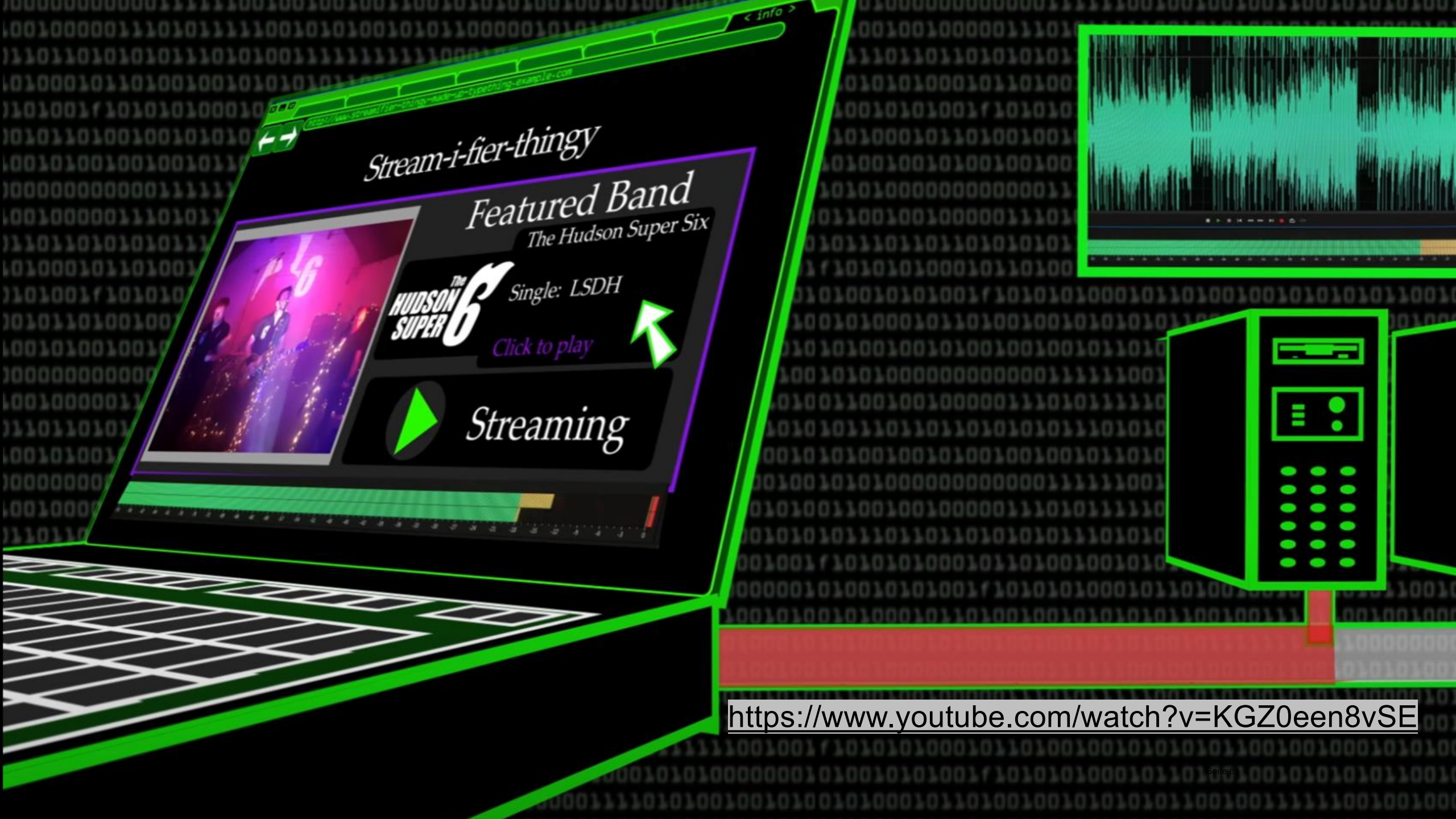
- Anwendungsbeispiele: Spotify, Apple Music, YouTube, Calls, Videokonferenzen, …

- Schallwellen sind analog
- sie werden durch Abtastung (Sampling) und Quantisierung in digitale Werte umgewandelt.
- Beispiel: CD-Qualität = 44,1 kHz (44.100 Messpunkte pro Sekunde), 16 Bit pro Sample.

- Wie groß sind 80 Minuten Musik (unkomprimiert)?

Audio

- Rohdaten wären groß (insbesondere im Maßstab von vor 30 Jahren…)
- deshalb nutzt man Kompression:
  - Unkomprimiert: direkte Speicherung der Samples, sehr große Dateien, höchste Qualität
    z. B. WAV-Dateien – sind Standard im Studio und bei professioneller Audiobearbeitung.
  - Verlustbehaftete Komprimierung: 80-90 % weniger Speicherbedarf, etwas Qualitätsverlust
    Hier werden nur die Signalanteile gespeichert, die das menschliche Gehör auch wahrnehmen kann
    z. B. MP3, AAC, OGG
  - Komprimiert verlustfrei: Datenreduktion von 40-60% gegenüber WAV ohne Qualitätsverlust, Prinzip ähnelt ZIP-Kompression, nur auf Audiodaten optimiert
    z. B. Free Lossless Audio Codec (FLAC)


< info >
Stream-i-fier-thingy
Featured Band
The Hudson Super Six
The HUDSON SUPER 6
Single: LSDH
Click to play
Streaming
https://www.youtube.com/watch?v=KGZ0een8vSE

Es gibt 10 Arten von Menschen:

– Menschen, die Binärcode verstehen,
– Menschen, die ihn nicht verstehen und
– Menschen, die kapieren, dass dieser Witz in Base 3 codiert ist.

HOCHSCHULE MAINZ
Wirtschaft
Business

Vielen Dank für Ihre Aufmerksamkeit!

University of
Applied Sciences

Kontakt

Prof. Dr. Dirk Schweim
Professur für Wirtschaftsinformatik

Hochschule Mainz
University of Applied Sciences
Lucy-Hillebrand-Str. 2
55128 Mainz, Germany

T +49 6131 628-3315
E Schweim@HS-Mainz.de
W www.hs-mainz.de/schweim