



# ZUSAMMENFASSUNG M114

Zusammenfassung zum Modul 114 über  
Codierungs-, Verschlüsselungs- &  
Kompressionsverfahren

Exposee

Zusammenfassung zum Modul 114 von RaviAnand Mohabir

RaviAnand Mohabir  
ravianand.mohabir@stud.altekanti.ch  
<https://dan6erbond.github.io>

## Inhalt

Zahlensysteme.....	3
Dezimalsystem.....	3
Binärsystem.....	3
Oktalsystem.....	3
Hexadezimalsystem.....	3
Codierung.....	3
Codierung von Zeichen.....	3
BCD-Code.....	3
X aus N Code.....	3
ASCII.....	3
ANSI.....	4
Unicode.....	4
GTIN.....	5
GTIN-13.....	5
QR-Code.....	6
Fehlererkennung.....	6
Fehlererkennung durch Parität.....	6
Redundanz.....	7
Speichergrößen.....	7
Umrechnung.....	7
Medien.....	7
Pixelgrafik (Rastergrafik).....	7
Vektorgrafik.....	8
Bildgröße und Farbtiefe.....	8
Grafikformate.....	8
Unkomprimiert.....	8
Komprimiert.....	8
Reduziert.....	8
Videoformate (Videocodecs).....	8
Audioformate.....	9
Unkomprimiert.....	9
Komprimiert.....	9
Komprimiert und reduziert.....	9
Bsp. MP3.....	9
Logische Operatoren.....	10

Arithmetische und logische Operatoren .....	10
NOT .....	10
AND, NAND .....	10
OR, NOR .....	10
XOR .....	10
Farbmodelle .....	10
Wie kommt Farbe in unsere Welt? .....	10
Der Monitor (drei Farben) .....	10
Das additive Farbmodell (RGB) .....	11
Der Drucker (vier Farben) .....	11
Das subtraktive Farbmodell (CMYK) .....	11
Farbangaben .....	11
Zahlendarstellung .....	11
Positive Zahlen .....	11
Negative Zahlen .....	11
Zweierkomplement .....	12
Überlauf .....	12
Gleitkommazahlen .....	12
Praktisches Beispiel .....	13



## Zahlensysteme

### Dezimalsystem

Wir haben wahrscheinlich das Dezimalsystem, weil wir 10 Finger haben. Im Dezimalsystem haben wir 10 verschiedene Symbole (0 bis 9).

Die Basis des Dezimalsystems ist 10, der Wert der Stellen ...,  $10^3$ ,  $10^2$ ,  $10^1$ ,  $10^0$

Beispiel:

$$\begin{array}{c} 1674_d \\ \swarrow \quad \downarrow \quad \downarrow \quad \searrow \\ 1 \times 10^3 + 6 \times 10^2 + 7 \times 10^1 + 4 \times 10^0 \\ 1000 + 600 + 70 + 4 = 1674_d \end{array}$$

### Binärsystem

Die Basis des Binärsystems ist 2, der Wert der Stellen ...,  $2^3$ ,  $2^2$ ,  $2^1$ ,  $2^0$ . Die Symbole sind 0 und 1.

Beispiel:

$$\begin{array}{c} 0110_b \\ \swarrow \quad \downarrow \quad \downarrow \quad \searrow \\ 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ 0 + 4 + 2 + 0 = (\text{dezimal} = 6_d) \end{array}$$

### Oktalsystem

Die Basis des Oktalsystems ist 8, der Wert der Stellen ...,  $8^3$ ,  $8^2$ ,  $8^1$ ,  $8^0$ . Die Symbole sind 0 bis 7.

Beispiel:

$$\begin{array}{c} 1674_o \\ \swarrow \quad \downarrow \quad \downarrow \quad \searrow \\ 1 \times 8^3 + 6 \times 8^2 + 7 \times 8^1 + 4 \times 8^0 \\ 512 + 384 + 56 + 4 = (\text{dezimal} = 956_d) \end{array}$$

### Hexadezimalsystem

Die Basis des Hexadezimalsystems ist 16, der Wert der Stellen ...,  $16^3$ ,  $16^2$ ,  $16^1$ ,  $16^0$ . Die Symbole sind 0 bis F (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

Beispiel:

$$\begin{array}{c} A70B_h \\ \swarrow \quad \downarrow \quad \downarrow \quad \searrow \\ A \times 16^3 + 7 \times 16^2 + 0 \times 16^1 + B \times 16^0 \\ 10 \times 16^3 + 7 \times 16^2 + 0 \times 16^1 + 11 \times 16^0 \\ 40'960 + 1'792 + 0 + 11 = (\text{dezimal} = 42'763_d) \end{array}$$

## Codierung

### Codierung von Zeichen

Mit n Bits können  $2^n$  Kombinationen dargestellt werden. Bsp. 4  $\rightarrow 2^4 = 16$ . Ergibt einen Zahlenraum von 0 bis 15.

### BCD-Code

Der BCD-Code codiert jede Ziffer einer Dezimalzahl mit 4 Bits. Zwei Ziffern ergeben ein Byte (8 Bits). Der Nachteil beim BCD-Code ist die hohe Speichernutzung.

### X aus N Code

Für eine Ziffer werden n-Bits benötigt. Jede Ziffer besteht aus einer festen Länge und wird mit gleich vielen (x) Einsen codiert.

2 aus 5  $\rightarrow$  Immer 2 Bits sind 1, jede Ziffer benötigt 5 Bits. Auch hier ist der Nachteil die hohe Speichernutzung, der Vorteil die einfache Fehlererkennung.

### ASCII

Mit dem ASCII-Code werden nicht nur Zahlen, sondern auch Buchstaben codiert. ASCII = American Standard Code for Information Interchange. Jedem Zahlenwert ist in einer Tabelle einem Zeichen zugeordnet. 7bit  $\rightarrow 2^7 \rightarrow 128$  Zeichen  $\rightarrow$  Englisch und Steuerzeichen. 8bit  $\rightarrow 256$  Zeichen (sprachspezifisch). Der Nachteil von ASCII ist das man nur genügend Speicher für englische und einige europäische Zeichen hat. ASCII Zeichen kann man mit der ALT-Taste + dem Code eingeben.

## ANSI

Bei der ANSI Codierung hat man 8bit pro Zeichen also insgesamt 256 Zeichen. Dies reicht aus für die englischen und europäischen Zeichen sowie Sonderzeichen. Die ersten 128 Zeichen sind mit dem ASCII-Zeichensatz gleich. Der ANSI-Code ist der ursprüngliche Zeichensatz von Windows. Auch ANSI-Zeichen können mit der ALT-Taste eingegeben werden.

## Unicode

Mit Unicode sollte man alle im Gebrauch befindlichen Schriftsysteme und Zeichen codieren. Dazu werden 17 Ebenen à je  $2^{16} = 65'536$  Zeichen verwendet. Die meisten Zeichen sind in der ersten Ebene abgebildet → alle europäischen Sprachen. Mit Unicode wird nur der Aufbau des Zeichensatzes beschrieben, nicht dessen Codierung.

### Unicode BMP

Die meisten Zeichen sind in der Eben 0 BMP (Basic Multilingual Plane) abgespeichert.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

■ Lateinische Schriften und Symbole

■ Lautschriften

■ Andere europäische Schriften

■ Nahost- und Südwestasiatische Schriften

■ Afrikanische Schriften

■ Südasiatische Schriften

■ Südostasiatische Schriften

■ Ostasiatische Schriften

■ CJK-Ideogramme

■ Kanadische Silben

■ Symbole

■ Diakritika

■ UTF-16-Surrogates und privater Nutzungsbereich

■ Verschiedene Zeichen

■ Nicht belegte Codebereiche

### UTF-32 & UTF-16

UTF-16 ist die praktische Codierung von Unicode: Unicode Transformation Format (UTF). UTF-32 ist die einfachste Codierung, benötigt jedoch 32bit pro Zeichen → sehr viel Platz. UTF-16 speichert die Zeichen der BMP als 16-bit Werte und die Zeichen der anderen Ebenen als 32bit Werte. Bspw. belegt ein A 16bit, der Violschlüssel 32bit.

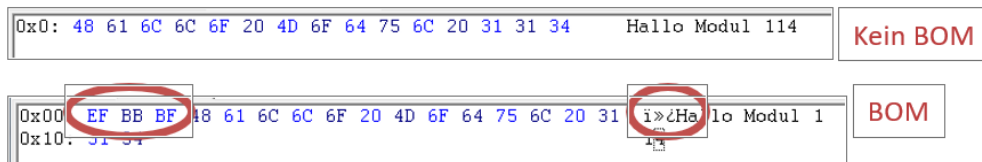
### UTF-8

Die UTF-8 Codierung wurde für Sprachen mit lateinischen Zeichen optimiert. Die Zeichen werden variabel in 1, 2, 3 oder 4 Bytes gespeichert also zwischen 8 bis 32bit. Die Zeichen 0 – 127 aus dem ASCII-Zeichensatz benötigen 8bit, die Zeichen der Unicode BMP 16bit und weitere Zeichen zwischen 24 und 32bit. UTF-8 wird oft für Webseiten verwendet.

### Unicode: Byte Order Mark BOM

Computersystem speichern Bytes nicht immer in der gleichen Reihenfolge ab. Little Endian: niederwertigstes Byte zuerst → PC/Windows. Big Endian: höchstwertiges Bytes zuerst → Mainframe.

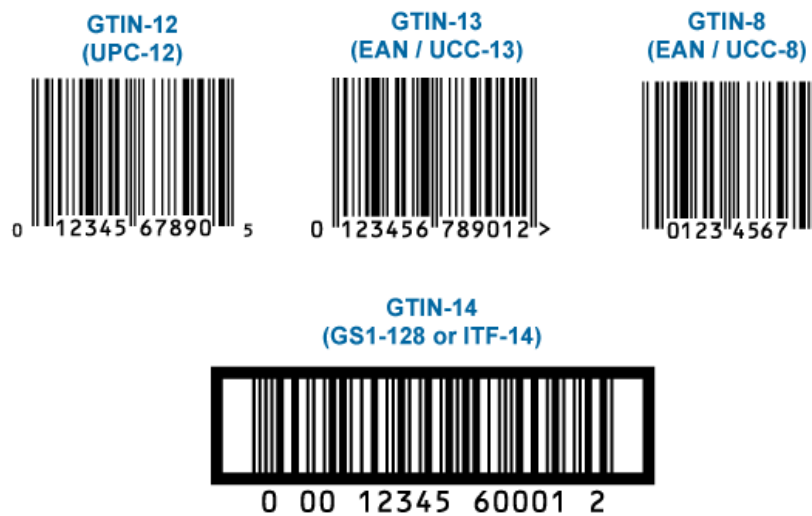
Wie es abgespeichert ist, ist einem binären Wort aber nicht anzusehen. In den ersten Bytes einer Textdatei kann diese Ordnung als Byte Order Mark BOM abgespeichert werden.



## GTIN

Global Trade Item Number (GTIN)

Die Zahlen werden mit einem Strichcode codiert. Ein GTIN-Code kann verschiedene Längen haben.



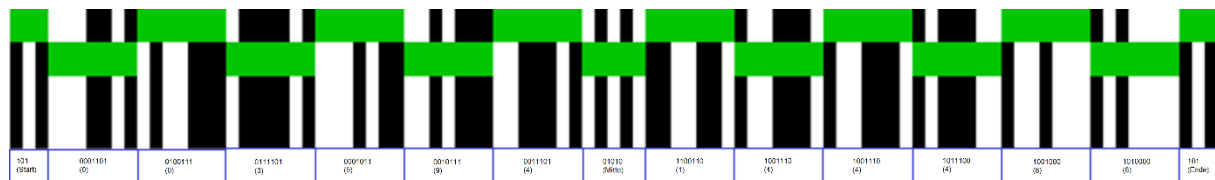
## GTIN-13

Der GTIN-13 ist eine sehr weit verbreitete Version des GTIN-Codes und besitzt 95 gleich breite Bereiche. Eine schwarze Breite bedeutet 1, eine weisse Breite 0.

Aufbau:

Start, 6 Zahlen links, Mitte, 6 Zahlen rechts, Ende

Die 13. Ziffer (erste Stelle) wird links enkodiert. Die Folge '101' ist am Start sowie am Ende zu finden. In der Mitte die Folge '01010'. Jede Ziffer ist mit 7bit (Streifen) codiert.



Rechte Ziffern werden immer gerade kodiert!!!



Ziffer	Muster			Kodierung der 13. Ziffer
	links		rechts	
	ungerade	gerade	(gerade)	
0	0001101	0100111	1110010	UUUUUU GGGGGG
1	0011001	0110011	1100110	UUGUGG GGGGGG
2	0010011	0011011	1101100	UUGGUG GGGGGG
3	0111101	0100001	1000010	UUGGGU GGGGGG
4	0100011	0011101	1011100	UGUUGG GGGGGG
5	0110001	0111001	1001110	UGGUUG GGGGGG
6	0101111	0000101	1010000	UGGGUU GGGGGG
7	0111011	0010001	1000100	UGUGUG GGGGGG
8	0110111	0001001	1001000	UGUGGU GGGGGG
9	0001011	0010111	1110100	UGGUGU GGGGGG

### Fehlererkennung

- Zur Fehlererkennung ist eine Prüfziffer bei Ziffer 12 hineincodiert.
- Berechnung der Prüfziffer:
  - Jeder der 12 Ziffern ist ein Multiplikator (1 oder 3 zugeordnet)

	13. Ziffer = erste Stelle der Nummer												Prüfziffer
Klartext:	4	0	0	8	5	3	5	1	2	9	2	1	6
Prüfziffer:													6
Nutzziffernfolge:	4	0	0	8	5	3	5	1	2	9	2	1	(Nutzziffer x Gewichtung)
Gewichtungsfaktoren:	1	3	1	3	1	3	1	3	1	3	1	3	
Einzelprodukte:	4	0	0	24	5	9	5	3	2	27	2	3	
Summe Einzelprodukte:	$4 + 0 + 0 + 24 + 5 + 9 + 5 + 3 + 2 + 27 + 2 + 3 = 84$												
Modulo 10:	$84 \text{ Mod. } 10 = 4 \text{ (} 84 / 10 = 8, \text{ Rest } 4 \text{)}$												
Differenz zu 10 ergibt die Prüfziffer:	$10 - 4 = 6$												

### QR-Code

Englisch für Quick Response («Schnell Antwort»). Der QR-Code ist ein zweidimensionaler Code und kann Text, URLs usw. codieren. QR-Codes können mit einer Software oder App gelesen werden (meist auf dem Tablet oder Smartphone per Kamera).

### Fehlererkennung

Fehler können durch falsch übertragene oder gespeicherte Bits entstehen.

#### Fehlererkennung durch Parität

Mithilfe der geraden («even») oder ungeraden («odd») Parität kann man Fehler finden.

Beispiel:

Even: 0010 1100 → Quersumme 3 → Paritätsbit 1, damit Quersumme gerade wird → 0010 1100 1

Odd: 0010 1100 → Quersumme 3 → Paritätsbit 0, damit Quersumme ungerade wird → 0010 1100 0

Mit dieser Methode können einfache Übertragungsfehler entdeckt werden. Sie ist sehr simpel kann aber nur einfach Fehler erkennen.

## Redundanz

Redundanz ist der Teil einer Nachricht, der keine Informationen enthält. Bei der Fehlererkennung fügen wir bewusst Redundanz hinzu um einen Fehler zu erkennen. Die Information bleibt die gleiche. Bei einer Fehlererkennung mit Parität benötigen wir für die Information 8bit und die Redundanz 1bit. Die Redundanz beträgt somit 1/9 oder 11%. Mehr Redundanz erhöht die Fehlererkennung und Fehlerkorrektur. Aber es entsteht eine grössere Datenmenge.

## Speichergrössen

Das Betriebssystem meldet bei einer 1TB grossen HDD 931 Gigabyte, wieso?

Weil der Computer nicht mit dem Dezimalsystem rechnet, werden für die Speichergrössen das Binärsystem verwendet mit einer Auswahl von Potenzen welche den Zehnerpotenzen nahekommen:

1 Kilobyte =  $2^{10}$  Bytes = 1024 Bytes      Bei Kilobytes ist die Abweichung 24 Bytes => 2.4%

Bei einem Exabyte aber schon 15%!

Die Lösung ist die Einführung neuer Präfixe. Die ersten Zwei Buchstaben der Dezimalpräfixe und dann «bi für binär»: Mega → Mebi

Binärpräfixe	
IEC-Name (IEC-Symbol)	Bedeutung
Kibibyte (KiB)	$2^{10}$ Byte = 1.024 Byte
Mebibyte (MiB)	$2^{20}$ Byte = 1.048.576 Byte
Gibibyte (GiB)	$2^{30}$ Byte = 1.073.741.824 Byte
Tebibyte (TiB)	$2^{40}$ Byte = 1.099.511.627.776 Byte
Pebibyte (PiB)	$2^{50}$ Byte = 1.125.899.906.842.624 Byte
Exbibyte (EiB)	$2^{60}$ Byte = 1.152.921.504.606.846.976 Byte
Zebibyte (ZiB)	$2^{70}$ Byte = 1.180.591.620.717.411.303.424 Byte
Yobibyte (YiB)	$2^{80}$ Byte = 1.208.925.819.614.629.174.706.176 Byte

## Umrechnung

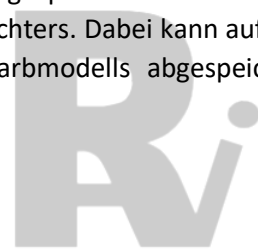
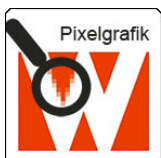
Dez in Bin:      Anzahl Bytes /  $2^{\text{grösste mögliche Zehnerzahl}}$  → 4 MegaBytes → 4'000'000 Bytes /  $2^{20}$  = 3.81 MebiBytes

Bin in Dez:      Anzahl Bytes •  $2^{\text{Präfix}}$  → 65 MebiBytes → 65 •  $2^{20}$  Bytes = 68'157'440 Bytes → 68.2 MegaBytes

## Medien

### Pixelgrafik (Rastergrafik)

Bei einer Pixelgrafik wird die Farbinformation von jedem einzelnen Pixel abgespeichert. Die einzelnen Pixel haben keinen Bezug zueinander. Das Bild entsteht im Kopf des Betrachters. Dabei kann auf eine Farbpalette zugegriffen werden, oder es werden Farbwerte eines Farbmodells abgespeichert. Pixelgrafiken werden meist für Fotos verwendet.





## Vektorgrafik

Bei einer Vektorgrafik werden die einzelnen Muster durch geometrische Formen gebildet. Bsp. Ein Kreis wird nicht aus einzelnen Pixel gezeichnet. Es wird nur angegeben, dass ein Kreis mit dem Radius  $r$  an der Stelle  $XY$  gezeichnet werden soll.

## Bildgrösse und Farbtiefe

Je mehr Bits für die Farbinformation zur Verfügung stehen, umso mehr Farben kann eine Grafik aufweisen. Das ist die Farbtiefe. 8bit → 256 Farben, 14bit → 16,7 Mio. Farben → Truecolor. Gute Grafikformate benutzen also rund 3Byte pro Pixel.

Wie viele Bytes belegt eine Datei...

- Mit 300 • 480 Bildpunkten (Pixel)?  $300 \cdot 480 \cdot 3 \text{ Bytes} = 432'000 \text{ Bytes} \rightarrow 421.9 \text{ KiB}$
- Mit der neuen Handykamera (16 Megapixel)?  $16 \cdot 10^6 \cdot 3 \text{ Bytes} = 48'000'000 \text{ Bytes} \rightarrow 45.8 \text{ MiB}$
- Mit einer Transparenz? + 8bit pro Pixel

Somit ist ein Pixel oft mit 32bit codiert.

## Grafikformate

### Unkomprimiert

Unkomprimiert bedeutet: Jeder Pixel (Farbe und oft auch die Helligkeit) wird einzeln abgespeichert: Windows Bitmap (.bmp), Graphics Interchange Format (.gif), Windows Icon (.ico), Tagged Image File Format (.tif), Rohdatenformat (.raw)

### Komprimiert

Beim Komprimieren wird mit einem Algorithmus die Dateigrösse verkleinert. Z.B. können mehrere weisse Punkte (Pixel) zusammengefasst werden. Grundsätzlich sollen bei der Komprimierung keine Informationen verloren gehen. Unter anderem wird in Portable Network Graphics komprimiert.

### Reduziert

Mit der Reduktion werden von Bilddateien Informationen weggerechnet. Oft wird die Reduktion mit der Komprimierung kombiniert um eine möglichst kleine Datei abzuspeichern. Diese Verfahren wird unter anderem in JPEG verwendet.

### Beispiel JPEG

Bei der Reduktion in JPEG wird das RGB-Farbmodell in das YCbCr-Farbmodell umgerechnet. Dieses Farbmodell stellt ein Bild nach Helligkeit und Farbabweichung Richtung Blau/Gelb (Cb) und Rot/Türkis (Cr). Die Bilder werden dann in 8x8 Quadrate zerlegt und einem der 64 Blöcke (Dichte der Helligkeit) zugeordnet. Die Daten werden danach noch komprimiert.

## Videoformate (Videocodecs)

Die vielen verschiedenen Videoformate unterscheiden sich in der Qualität und dem Einsatzgebiet. Die meisten davon sind verlustbehaftet, das heisst, dass sie nicht verlustfrei sind. Das Video ist in einem Container verpackt. Die Bildspur und Tonspur ist dort verpackt. Jede Spur wird einzeln verarbeitet und codiert. Es gibt viele Videoformate. Einige bekannte sind:

- |                                       |                                      |
|---------------------------------------|--------------------------------------|
| - MPEG-1: Video-CDs                   | - AVI: Veraltet aber weit verbreitet |
| - MPEG-2: DVDs                        | - OGG: OpenSource Videoformat        |
| - MPEG-4: Internet Video, BluRay etc. | - Flash: Nicht mehr so populär       |
| - WebM: Video in HTML5                |                                      |

## Audioformate

Ton kann durch Digitalisierung des Tonsignals für den Computer nutzbar gemacht werden. Analog-Digital-Convert (ADC).

Zum Abspielen werden die digitalen Signale wieder in Analoge umgewandelt. Digital-Analog-Convert (DAC).

Vorgang der Digitalisierung:

- Abtastung: Wie oft pro Sekunde wird das Signal gemessen?
- Quantisierung: Mit wie vielen Bits wird ein Messwert gespeichert?
- Codierung: Wie werden die Dateien abgespeichert?

Bei CD-Qualität wird mit 44'100 Hertz (44'100 Mal pro Sekunde) das Tonsignal abgetastet. Für die Speicherung eines Wertes stehen 16Bit (2 Bytes) zur Verfügung. Das ganze wird in Stereo abgespeichert.

Beispiel zur Berechnung der Dateigrösse:

- Musikstück von 10 Minuten  $\rightarrow 10 \cdot 60s = 600s$
- $600s \cdot 44'100 \text{ Hz} \cdot 2 \text{ Kanäle} \cdot 2 \text{ Bytes} = 105'840'000 \text{ Bytes} \rightarrow = 100.9 \text{ MiB}$
- Durch Reduktion der Abtastrate und/oder Abtasttiefe sinkt die Qualität aber auch die Dateimenge

## Unkomprimiert

AIFF, WAV

## Komprimiert

MPEG-4 Audio Lossless Coding (ALS)

Windows Media Audio Lossless (WMA Lossless)

## Komprimiert und reduziert

MP3  $\rightarrow$  .mp3

MPEG-1 Audio, MPEG-2 Audio, MPEG-4 Audio

Ogg-Vorbis  $\rightarrow$  .ogg

Windows Media Audio  $\rightarrow$  .wma

## Bsp. MP3

- Psychoakustisches Modell:
  - o Adaptive Hörschwelle: Wir hören Töne im Frequenzbereich der menschlichen Sprache besser als andere (1kHz bis 5kHz)
  - o Maskierung: Laute hohe Töne unterdrücke leise hohe Töne
  - o Vor- und Nachmaskierung: Ein Sprung von Laut auf Leise lässt die leisen Töne verschwinden
- Das Signal wird
  - o Durch das psychoakustische Modell gefiltert (reduziert)
  - o Quantisiert
  - o Komprimiert
  - o Codiert



## Logische Operatoren

### Arithmetische und logische Operatoren

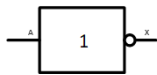
Ein Computer kann nicht nur mathematisch rechnen. → plus, minus usw.

Er kann auch logische Operationen durchführen. Da er binär rechnet, kennt der Computer nur 2 Zustände: Logisch 1: Wahr, True; Logisch 0: Falsch, False

#### NOT

NOT ist die Umkehr. True → False, False → True

Symbol



Wahrheitstabelle

A	X
0	1
1	0

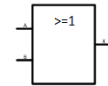
Gleichung

$$X = \neg A$$

#### OR, NOR

Wenn einer Eingänge wahr sind, ist der Ausgang wahr.

Symbol



Wahrheitstabelle

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

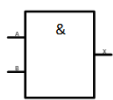
Gleichung

$$X = A \vee B$$

#### AND, NAND

Wenn beide Eingänge wahr sind, ist der Ausgang wahr.

Symbol



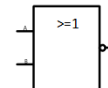
Wahrheitstabelle

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Gleichung

$$X = A \wedge B$$

Symbol



Wahrheitstabelle

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Gleichung

$$X = \neg(A \vee B)$$

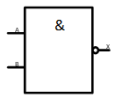
dul 114, Logische Operatoren

BerufsBild

#### XOR

Wenn genau eine der Eingänge wahr ist, ist der Ausgang wahr.

Symbol



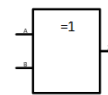
Wahrheitstabelle

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Gleichung

$$X = \neg(A \wedge B)$$

Symbol



Wahrheitstabelle

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Gleichung:

$$X = (A \wedge \neg B) \vee (\neg A \wedge B)$$

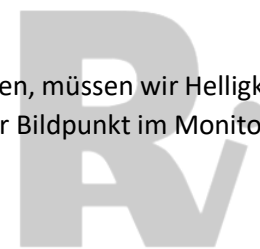
## Farbmodelle

### Wie kommt Farbe in unsere Welt?

Licht besteht aus elektromagnetischen Wellen. Nur ein Teil der Frequenzen des Lichts sind für das Auge sichtbar. Trifft Licht auf einen Gegenstand, so werden gewisse Frequenzen absorbiert (aufgenommen) und andere reflektiert (zurückgeworfen). Wir sehen eine Farbe. Schwarz absorbiert alle Frequenzen. Weiss reflektiert alle Frequenzen.

### Der Monitor (drei Farben)

Die Grundfarbe des Monitors ist schwarz. Wenn wir Farben erzeugen wollen, müssen wir Helligkeit hinzufügen (addieren). Die Grundfarben sind Rot, Grün, Blau → RGB. Jeder Bildpunkt im Monitor besteht aus diesen drei Farben.



## Das additive Farbmodell (RGB)

Durch die Mischung der drei Farben (red, green, blue) entstehen weitere Farben. Durch die Intensität (Stärke) der Farben können weitere Farben gemischt werden. Treffen alle drei Farben zusammen entsteht weiss.

## Der Drucker (vier Farben)

Die Grundfarbe des Druckers ist weiss. → Papierfarbe. Wenn wir Farben erzeugen wollen, müssen wir Helligkeit wegnehmen (subtrahieren). Die Grundfarben sind Türkis (cyan), Purpur (magenta), Gelb (yellow), Schwarz (black) → CMYK. Da die Mischung aus CMY ein Braun ergibt, muss mit einer vierten Farbe Schwarz hinzugefügt werden → black. Jeder Bildpunkt auf dem Papier besteht aus diesen vier Farben.

## Das subtraktive Farbmodell (CMYK)

Durch die Mischung der vier Farben (cyan, magenta, yellow, black) entstehen weitere Farben. Durch die Dichte der Pixel (dots per inch) entstehen für das Auge viele verschiedene Farben.

## Farbangaben

Im RGB-Modell stehen für jede der drei Grundfarben ein Byte zur Verfügung → Werte von 0-255

Bsp. Orange Angabe Dezimal: rgb(253,193,105)

Bsp. Orange Angabe Hex: #FDC169

## Zahlendarstellung

### Positive Zahlen

Positive Zahlen werden binär gleich wie Dezimal dargestellt.

Bsp. 13(d) → 1101(b)

Mit n Bits können  $2^n$  Zahlen dargestellt werden.

### Negative Zahlen

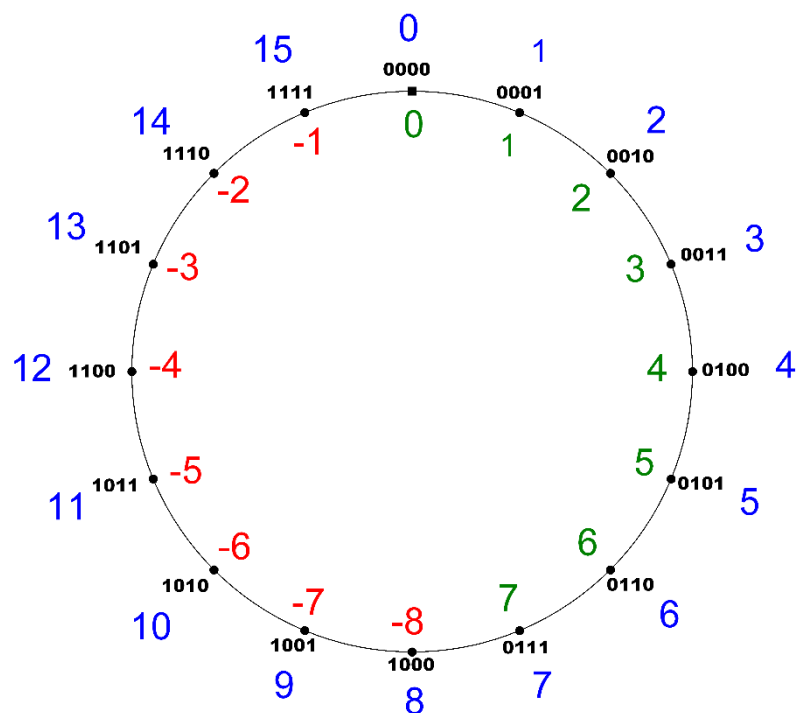
Es gibt zwei Methoden negative Zahlen darzustellen:

#### Methode 1:

Das erste Bit wird als Vorzeichen verwendet:  
0010(b) → +2(d); 1010(b) → -2(d)

#### Methode 2:

Der Zahlenraum wird in der Mitte aufgeteilt. Ein Teil ist negativ, der andere positiv. In diesem Beispiel sind das die Zahlen von -8 bis +7. Es gibt nur noch eine Null! Und wir haben das erste Bit als Vorzeichen behalten.



## Zweierkomplement

Die Negative Zahl ist somit das Zweierkomplement der positiven Zahle.

Beispiel: 5(d)  $\rightarrow$  0101(b)

- Alle Bits invertieren  $\rightarrow 1010$  (b)
- 1 dazuzählen  $\rightarrow 1011$ (b)  $\rightarrow$  das entspricht -5(d)
- Das erste Bit ist eine 1  $\rightarrow$  negative Zahl

Beispiel umgekehrt: Gegeben ist die Zahl 1101(b)

- Alle Bits invertieren  $\rightarrow 0010(b)$
- 1 dazuzählen  $0011(b) \rightarrow$  das entspricht  $+3(d)$
- Das erste Bit ist eine 0  $\rightarrow$  positive Zahl

## Überlauf

Ein Computer hat einen endlich grossen Speicher. Beispiel: Ein Computer mit 4Bit-Speicher.

- Sie haben die Zahl 7(d)  $\rightarrow$  0111(b) und addieren 1 dazu.
- Das Resultat ist 1000(b)  $\rightarrow$  -8(d)

Das darf natürlich nicht passieren.

## Gleitkommazahlen

Wir müssen aber auch Gleitkommazahlen darstellen können. Gleitkommazahlen sind Näherungen.

Beispiel:  $1/3 = 0.333...$  Und hier ist der Speicher zu Ende!

Die Zahlen können also nur mit einer gewissen Ungenauigkeit gespeichert werden.

## Wie machen wir das im Dezimalsystem?

- Bsp:  $1'856.43 \rightarrow$  schreiben wir auch als  $1.85643 \cdot 10^3$
- Darstellung im Taschenrechner: 1.85643E3
- Wir haben ein Vorzeichen  $\rightarrow$  hier: +
- Eine Mantisse  $\rightarrow 1.85643$
- Die Basis  $\rightarrow 10$
- Und einen Exponenten  $\rightarrow 3$

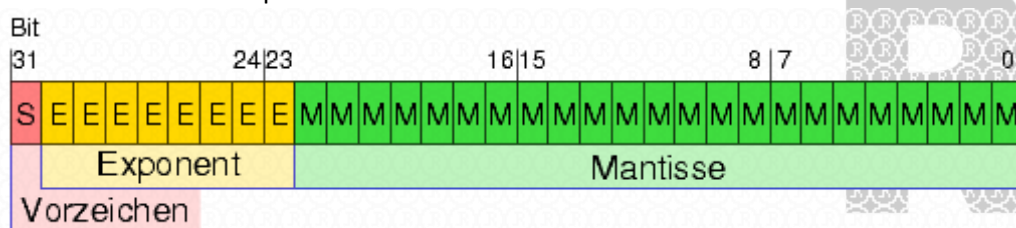
Nun benötigen wir aber ein System mit der Basis 2. Lösung: Gleitkommazahlen nach IEEE 754.

Darstellung:  $x = s \cdot m \cdot b^e$ .

- $s \rightarrow$  Vorzeichen,  $m \rightarrow$  Mantisse,  $b \rightarrow$  Basis 2,  $e \rightarrow$  Exponent

## IEEE 754 definiert verschiedene Grössen

- 32bit bis 64bit
- Wir benutzen das Beispiel mit 32bit:



## Praktisches Beispiel

- 18,4<sub>10</sub> soll in eine Gleitkommazahl umgewandelt werden.
- Zuerst wird die Dezimalzahl in eine Binärzahl gewandelt.
  - Vorkommaanteil = 18
    - 18 / 2 = 9 Rest 0 (Least-Significant Bit)
    - 9 / 2 = 4 Rest 1
    - 4 / 2 = 2 Rest 0
    - 2 / 2 = 1 Rest 0
    - 1 / 2 = 0 Rest 1 (Most-Significant-Bit) = 10010
  - Nachkommaanteil = 0.4
    - 0,4 \* 2 = 0,8 → 0 (Most-Significant-Bit)
    - 0,8 \* 2 = 1,6 → 1
    - 0,6 \* 2 = 1,2 → 1
    - 0,2 \* 2 = 0,4 → 0
    - 0,4 \* 2 = 0,8 → 0
    - 0,8 \* 2 = 1,6 → 1 (Least-Significant-Bit)
  - 
  - = 0,0110011001100110011...
  - 18,4 = 10010,011001100110011...

- Nun wird die Zahl normiert, wir erhalten die Mantisse
  - 10010,01100110011... \* 2<sup>-4</sup> = 1,001001100110011... \* 2<sup>4</sup>
- Berechnung des Exponenten
  - Da wir negative und positive Exponenten darstellen müssen, wird der Exponent um die Hälfte des Zahlenbereichs verschoben (Excess). → 8 Bit → Exzess = (256-2)/2 = 127
  - da 2<sup>4</sup> → Exponent = 4
  - Exponent + Exzess = 4 + 127 = 131<sub>10</sub>
  - Exponent in Binärzahl umrechnen → 10000011
- Vorzeichen berechnen (0 → positiv, 1 → negativ)
  - Die Zahl ist positiv → 0

- Die Gleitkommazahl bilden.
  - 1 Bit Vorzeichen → 0
  - 8 Bit Exponent → 10000011
  - 23 Bit Mantisse → 00100110011001100110011
- 0 10000011 00100110011001100110011
- Die Vorkomma-Eins bei der Mantisse muss nicht gespeichert werden, da dort immer eine 1 steht.

