

# Grundfragen der Informatik, KV

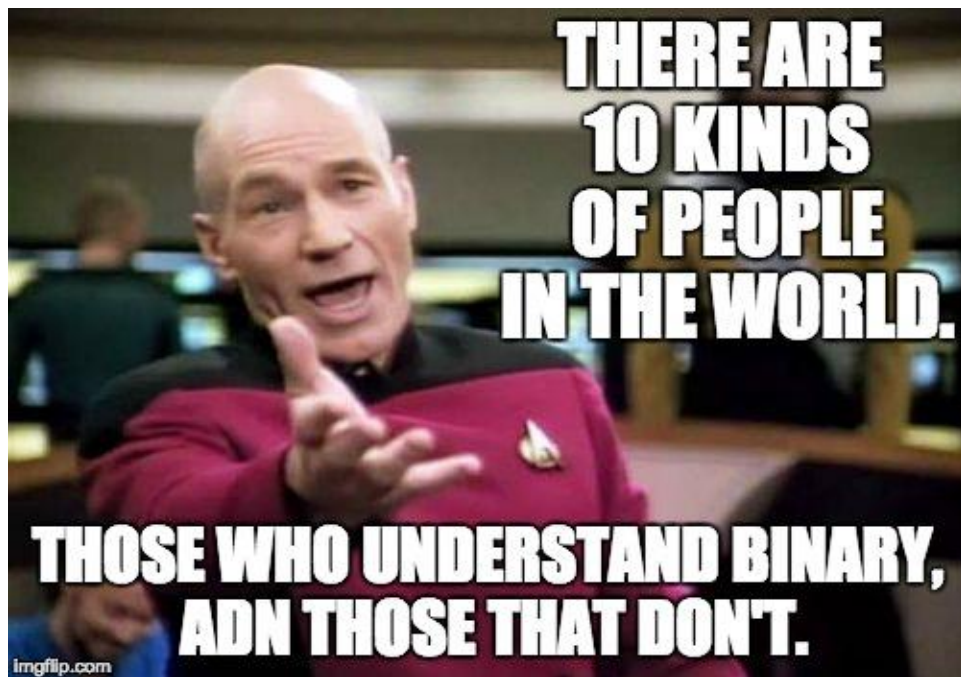
Zahlen, Bits & Bytes,  
Zeichenkodierung



# Überblick

---

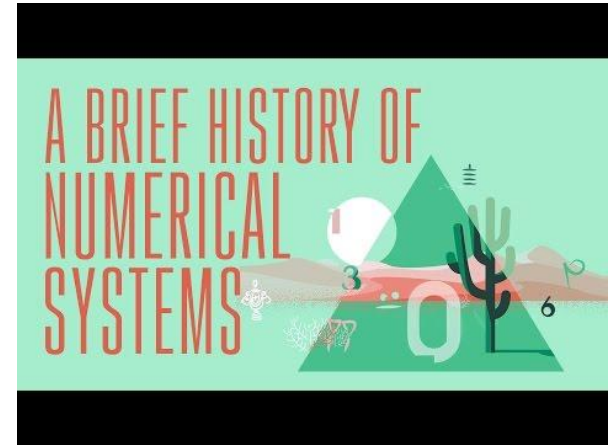
- Zahlensysteme: Binär, Dezimal, Hexadezimal
- Zahlenumwandlung
- Bits und Bytes
- Rechnen mit Binärzahlen
- Overflow und negative Zahlen
- Zeichenkodierung ASCII, UNICODE, UTF



# Additionssystem vs. Stellenwertsystem

- Eine **Zahl** ist eine eine Sequenz von Ziffern beliebiger Langer.
- Menge der Ziffern bildet ein Alphabet.

1	11	21	31	41	51
2	12	22	32	42	52
3	13	23	33	43	53
4	14	24	34	44	54
5	15	25	35	45	55
6	16	26	36	46	56
7	17	27	37	47	57
8	18	28	38	48	58
9	19	29	39	49	59
10	20	30	40	50	



# Zahlensysteme: Binär - Dezimal - Hexadezimal

---

**Alphabet<sub>10</sub> = {0,1,2,3,4,5,6,7,8,9}**

**Alphabet<sub>2</sub> = {0,1}**

**Alphabet<sub>16</sub> = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}**

Dezimal	Dual	Hexadezimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14

# Dezimaldarstellung - Stellenwertsystem

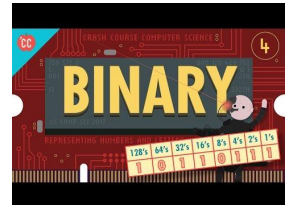
263

100er	10er	1er
2	6	3
$2 \cdot 10^2$	$6 \cdot 10^1$	$3 \cdot 10^0$

## Binärdarstellung

4er	2er	1er
1	0	1

$$\begin{array}{c} 4 + 0 + 1 = 5 \\ \begin{array}{ccc} | & | & | \\ 1 & 0 & 1 \end{array} \end{array}$$



# Eine Binärzahl

---

1      0      1      1      0      1      1      1

128er	64er	32er	16er	8er	4er	2er	1er
1	0	1	1	0	1	1	1

$$128 + 0 + 32 + 16 + 0 + 4 + 2 + 1 = 183$$

$$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

# Zahlensysteme: Binär - Dezimal - Hexadezimal

**Alphabet<sub>10</sub> = {0,1,2,3,4,5,6,7,8,9}**

$$316_{10} = 3 * 10^2 + 1 * 10^1 + 6 * 10^0$$
$$300 \quad + 10 \quad + 6$$

**Alphabet<sub>2</sub> = {0,1}**

$$1010_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$
$$8 \quad + 0 \quad + 2 \quad + 0$$

**Alphabet<sub>16</sub> = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}**

$$4A_{16} = 4 * 16^1 + 10 * 16^0$$
$$64 \quad + 10$$

Dezimal	Dual	Hexadezimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14



# Zahlenumwandlung

# Binär nach Dezimal

---

Jede Stelle der Zahl hat den Wert der entsprechenden 2er-Potenz.

Nimm jede Ziffer mal der entsprechenden Potenz und summiere.

Gehe am besten von rechts nach links vor:

01001010

$$\begin{aligned} 0*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 \\ 0 + 64 + 0 + 0 + 8 + 0 + 2 + 0 = \underline{74} \end{aligned}$$

$$254 = 2^8$$

$$128 = 2^7$$

$$64 = 2^6$$

$$32 = 2^5$$

$$16 = 2^4$$

$$8 = 2^3$$

$$4 = 2^2$$

$$2 = 2^1$$

$$1 = 2^0$$

# Dezimal nach Binär

---

1. Die Zahl durch 2 dividieren
2. Den Rest der Division notieren.
3. Falls das Ergebnis nicht 0 ist, Schritt 1 und 2 wiederholen.
4. Die Restergebnisse von unten nach oben lesen

(manchmal gibts es vorangehende 0en, um auf ein Byte (8 Bit) aufzuschließen, für die Zahlendarstellung können sie aber ignoriert werden)

74	:	2		0	R
37	:	2		1	R
18	:	2		0	R
9	:	2		1	R
4	:	2		0	R
2	:	2		0	R
1	:	2		1	R



01001010

# Binär nach Hexadezimal

---

1. Unterteile die Binärzahl von rechts nach links in 4er-Päckchen.
2. Jede Ziffer auflösen (Blick auf Tabelle, Folie 7).
3. Wandle jedes Päckchen in die entsprechende Hexadezimalziffer um.  
Alphabet = {1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

01001010

0100

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

$$0 + 4 + 0 + 0 = 4$$

1010

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

$$8 + 0 + 2 + 0 = 10 = A$$

4A

# Hexadezimal nach Binär

---

Der umgekehrte Weg...

**F18**<sub>16</sub> (3864<sub>10</sub>)

**1111 0001 1000**<sub>2</sub>

Weil die Konversion zwischen Binär und Hexadezimal so leicht ist, wird Hex oft in der Informatik verwendet. Sozusagen als Kurzschreibweise für Binärfolgen, wie auch das Oktalsystem, mit 8 Ziffern (<https://de.wikipedia.org/wiki/Oktalsystem>).

# Rechnen im Binärsystem

# Binäre Addition

-

# Subtraktion

Die Addition funktioniert wie die Addition von Dezimalzahlen:

$$\begin{array}{r} 0111_2 \quad 7_{10} \\ +0100_2 \quad 4_{10} \\ \hline 1011_2 \quad 11_{10} \end{array}$$

*“Wir borgen uns immer von der voran gestellten Position etwas aus, wenn wir 0 - 1 haben” ( $\rightarrow 0 - 1 = 1$  und Übertrag)*

$$\begin{array}{r} 1 \quad 0 \quad 0 \quad 0 \quad 1_2 \quad 17_{10} \\ - \quad 0 \quad 1 \quad 1 \quad 1_2 \quad 7_{10} \\ \hline \quad 1 \quad 0 \quad 1 \quad 0_2 \quad 10_{10} \end{array}$$

[https://www.mathematik-wissen.de/schriftlich\\_rechnen\\_im\\_binaersystem.htm](https://www.mathematik-wissen.de/schriftlich_rechnen_im_binaersystem.htm)



# Binäre Multiplikation

# Division

## Regeln

$$0 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 0 = 0$$

$$1 * 1 = 1$$

$$\underline{1001} * \underline{1011}$$

1001

0000

1001

1001

1100011

$$110101 : 1010 = 000101$$

1 //passt nicht in 1010

11 //nope

110 //nope

1101 //1101 - 1010

-1010

00110

1101 //1101 - 1010

-1010

0011 R



# Übung

---

Schreibe die Zahl 35 als Binärzahl an (manuelle Umwandlung). Wandle sie dann manuell in eine hexadezimale Zahl um.

Wandle in binäre Zahlen um und führe folgende arithmetischen Operationen durch und schreibe das Ergebnis auch als hexadezimale Zahl an.

- $3 + 7 = ?$
- $20 - 6 = ?$
- $12 / 3 = ?$
- $2 * 6 = ?$

# Übung in der Konsole (=Terminal)

---

## Binär nach Dezimal

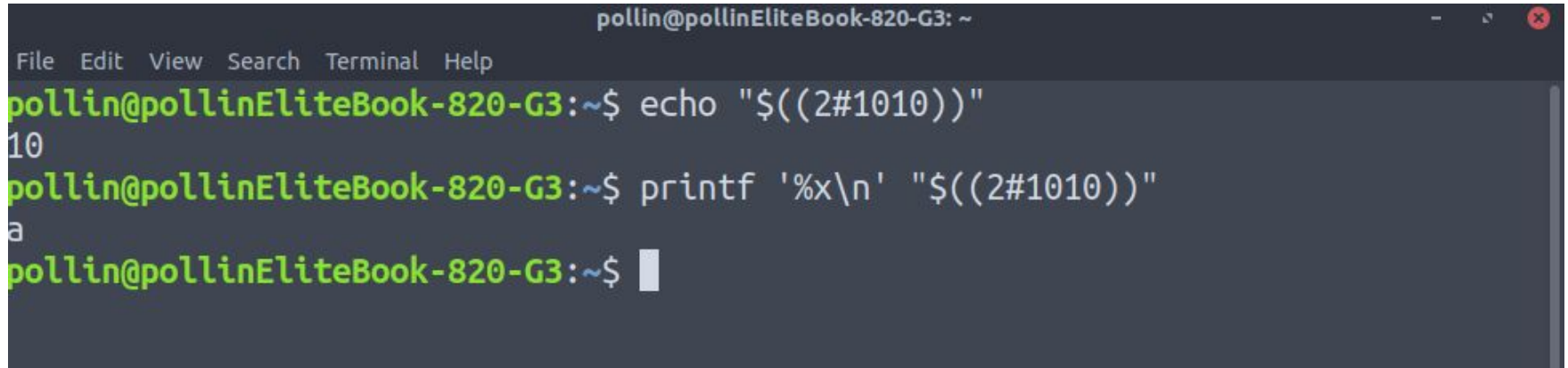
```
echo "$((2#1010))"
```

<https://www.geeksforgeeks.org/echo-command-in-linux-with-examples/>

## Binär nach Hexadezimal

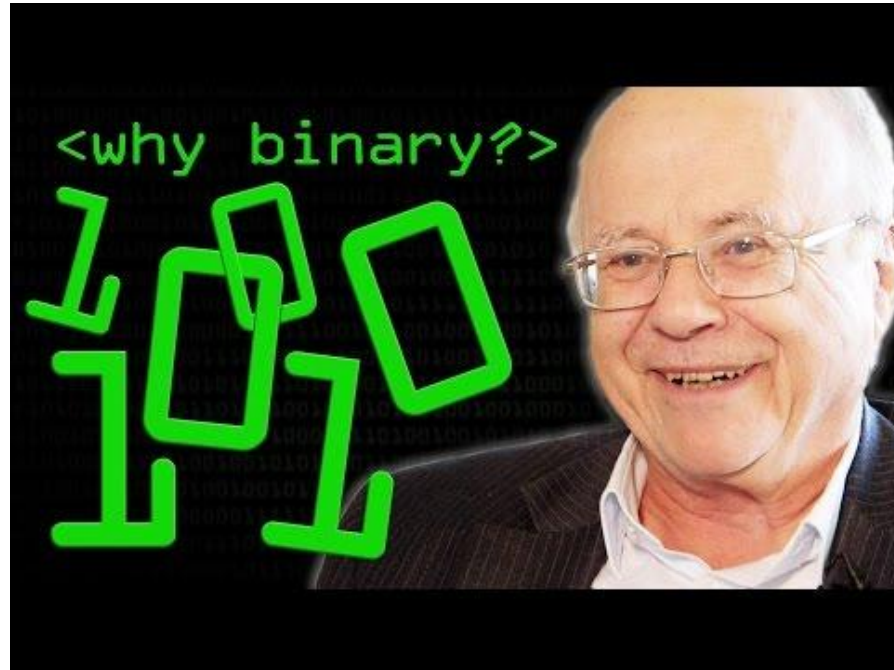
```
printf '%x\n' "$((2#1010))"
```

<https://www.computerhope.com/unix/uprintf.htm>



```
pollin@pollinEliteBook-820-G3: ~  
File Edit View Search Terminal Help  
pollin@pollinEliteBook-820-G3:~$ echo "$((2#1010))"  
10  
pollin@pollinEliteBook-820-G3:~$ printf '%x\n' "$((2#1010))"  
a  
pollin@pollinEliteBook-820-G3:~$
```

Es gibt natürlich auch Konverter im Web: <https://www.rapidtables.com/convert/number/index.html>



# Bit - **B**inary Digit

---

## Kleinstmögliche Einheit von Information

falsch - wahr

nein - ja

ungeladen - geladen

0 - 1

0 Süd

1 Nord

00 Süd

01 West

10 Nord

11 Ost

000 Süd

001 Südwest

010 West

011 Nordwest

100 Nord

101 Nordost

110 Ost

111 Südost

*Jedes zusätzliche Bit verdoppelt die Anzahl der Möglichkeiten Information zu repräsentieren.*

# Bit

Es gibt  $2^n$  verschiedene **Bitfolgen** der Länge  $n$ .

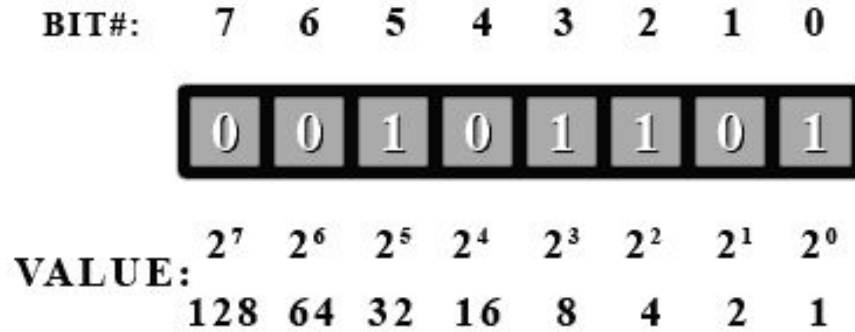
Es wächst logarithmisch zur Basis 2.

Mehr Bit, mehr Information... so auch besser Videos:



Anzahl	
der Bits	der Zustände
0	1
1	2
2	4
3	8
4	16
...	...
8	256
10	1024
12	4096
14	16.384
16	65.536
...	...
32	4.294.967.296

# Byte



- 1 Byte = 8 bit
- 1 Kilobyte (KB) = 1024 Byte
- 1 MB =  
1024 KB =  
1.048.576 Byte
- 1 GB =  
1024 MB) =  
1.048.576 KB =  
**1.073.741.824 Byte**

Dezimalpräfixe		
Name	Symbol	Anzahl Bytes <sup>[G 1]</sup>
Kilobyte	kB <sup>[G 2]</sup>	1 000 = $10^3$
Megabyte	MB	1 000 000 = $10^6$
Gigabyte	GB	1 000 000 000 = $10^9$
Terabyte	TB	1 000 000 000 000 = $10^{12}$
Petabyte	PB	1 000 000 000 000 000 = $10^{15}$
Exabyte	EB	1 000 000 000 000 000 000 = $10^{18}$
Zettabyte	ZB	1 000 000 000 000 000 000 000 = $10^{21}$
Yottabyte	YB	1 000 000 000 000 000 000 000 000 = $10^{24}$



# 32-bit System und 64-bit System

---

- Dies operieren mit 32 bzw. 64 Bits.
- Mit 32-bit ist die größte Zahl die repräsentieren werden kann:

**4 294 967 294**

damit kann man schon recht schöne Bilder, Videos, Spiele etc. machen

- Mit 64-bit ist die größte Zahl die repräsentieren werden kann:

**9 223 372 036 854 775 807**

damit kann man schon recht schöne Bilder, Videos, Spiele etc. machen



-214623719 Views ?



# Overflow

---

Zahlen sind durch ihre Länge beschränkt. Zahlen liegen an einem beliebigen Ort im Speicher.

$$\begin{array}{rclcl} 1111\ 1111_2 & = & 255_{10} & = & FF_{16} \\ +0000\ 0001_2 & = & 1_{10} & = & 1_{16} \\ \hline & & & & \end{array}$$

ist 0?

# Zahlendarstellung negativer Zahlen

Wie werden negative Zahlen im Rechner dargestellt?

$$111_2 = 7_{10}$$

$$- \quad 1 | 111 = -7$$

Signed Bit

$$+ \quad 0 | 111 = +7$$

$$0 | 111 \quad 7 + (-3) = 4$$

$$\begin{array}{r} +1 | 011 \\ \hline \end{array}$$

$$\begin{array}{r} 1 | 010 \quad -2 !? \\ \hline \end{array}$$



# Einer- und Zweierkomplement

---

## Das Einerkomplement

Jede Zahl wird durch ihr Gegenteil ersetzt:

01011010	NOT
10100101	

## Das Zweierkomplement

Zum Einerkomplement wird zusätzlich  
noch 00000001 addiert:

01011010	NOT
10100101	
<u>00000001</u>	+1
10100110	

**Die Subtraktion von 2 Zahlen erfolgt durch die Addition des Zweierkomplementes der zweiten Zahl!**

$$\begin{array}{r} 0111 \quad 7 - 3 = 4 \\ -0011 \\ \hline \end{array}$$

1 Kompl.: 1100 | NOT  
2 Kompl.: 1101 | +1

$$\begin{array}{r} 0111 \quad 7 + (-3) = 4 \\ +1100 \\ \hline 0100 \end{array}$$

1 der erste fliegt raus

Jetzt schreiben wir hier ohne das Signed Bit, weil das ja im Einer- bzw. Zweierkomplement steckt.

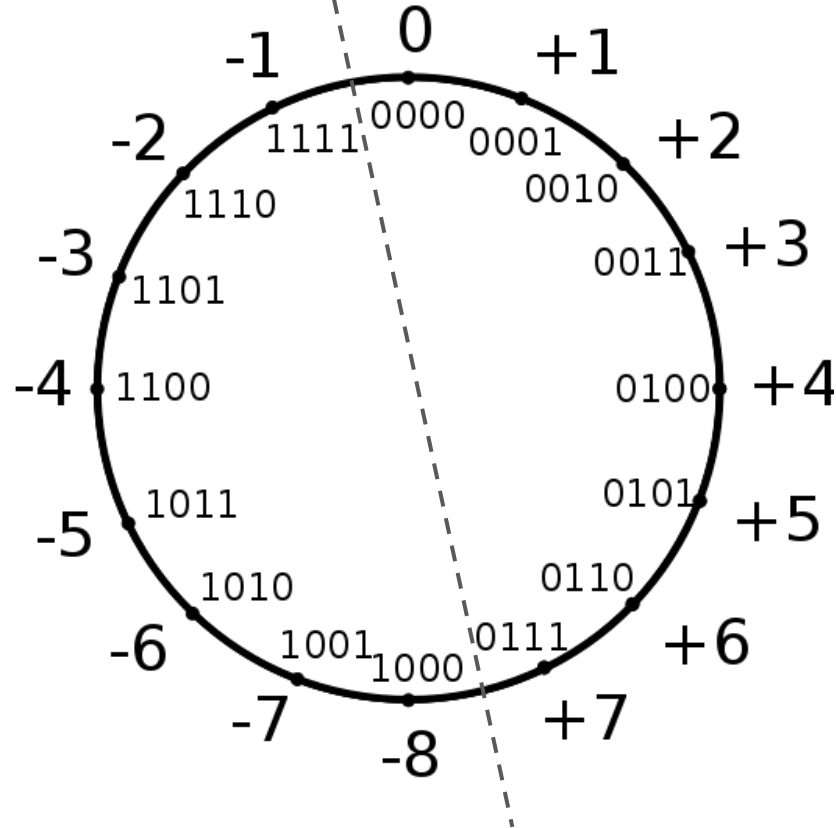
“Ver

wende das Signed Bit wie ein normales Bit.”

Übertrag ganz links wird einfach ignoriert!

# Signed Bits im Zweierkomplement

Jetzt gibts es  
auch nur  
noch eine 0!



# Binäre Subtraktion und negative Zahlen

---

$$\begin{array}{rcl} 1010_2 & = & 10_{10} \\ - 1110_2 & = & 14_{10} \end{array}$$

Jetzt wollen wir das Zweierkomplement von 1111 mit 1010 addieren um zu subtrahieren.

Einerkomplement

$$\begin{array}{r} 1110_2 \quad | \text{NOT} \\ 0001_2 \end{array}$$

Zweierkomplement

$$\begin{array}{r} 0001_2 \\ + 0001_2 \\ \hline 0010_2 \end{array}$$

$$\begin{array}{r} 1010_2 \\ + 0010_2 \\ \hline 1100_2 \end{array}$$

$$1100_2 = 12_{10} ?$$

→ erstes Bit ist jetzt das **Signed Bit** und definiert das Vorzeichen,  $1|100$  und  $100_2 = 4_{10}$

# Gleitkommadarstellung



**625,9 kann man auch schreiben als  $0,6259 * 10^3$**

Screenshot aus <https://www.youtube.com/watch?v=1GSjbWt0c9M&list=PLaHADNRco7n0KyC3U61AYOQKIGa5S-yUL&index=10&t=181s>

[https://en.wikipedia.org/wiki/Single-precision\\_floating-point\\_format](https://en.wikipedia.org/wiki/Single-precision_floating-point_format)

# Zeichenkodierung

Schei❖ encoding

Kodierungen und Ähnliches



# American Standard Code for Information Interchange - ASCII

7-Bit Zeichenkodierung

most significant Bits  
000 0000 = 0 = null

least significant Bits  
111 1111 = 127 = DEL

A = 65 → 100 0001

B = 66 → 100 0010

C = 67 → 100 0011

a = 97 → 110 0001

b = 98 → 110 0010

b = 99 → 110 0011

Scan-code	ASCII hex dez	Zeichen	Scan-code	ASCII hex dez	Zch.	Scan-code	ASCII hex dez	Zch.	Scan-code	ASCII hex dez	Zch.
	00 0	NUL ^@		20 32	SP		40 64	@	0D	60 96	`
	01 1	SOH ^A	02	21 33	!	1E	41 65	A	1E	61 97	a
	02 2	STX ^B	03	22 34	"	30	42 66	B	30	62 98	b
	03 3	ETX ^C	29	23 35	#	2E	43 67	C	2E	63 99	c
	04 4	EOT ^D	05	24 36	\$	20	44 68	D	20	64 100	d
	05 5	ENQ ^E	06	25 37	%	12	45 69	E	12	65 101	e
	06 6	ACK ^F	07	26 38	&	21	46 70	F	21	66 102	f
	07 7	BEL ^G	0D	27 39	'	22	47 71	G	22	67 103	g
0E	08 8	BS ^H	09	28 40	(	23	48 72	H	23	68 104	h
0F	09 9	TAB ^I	0A	29 41	)	17	49 73	I	17	69 105	i
	0A 10	LF ^J	1B	2A 42	*	24	4A 74	J	24	6A 106	j
	0B 11	VT ^K	1B	2B 43	+	25	4B 75	K	25	6B 107	k
	0C 12	FF ^L	33	2C 44	,	26	4C 76	L	26	6C 108	l
1C	0D 13	CR ^M	35	2D 45	-	32	4D 77	M	32	6D 109	m
	0E 14	SO ^N	34	2E 46	.	31	4E 78	N	31	6E 110	n
	0F 15	SI ^O	08	2F 47	/	18	4F 79	O	18	6F 111	o
	10 16	DLE ^P	0B	30 48	0	19	50 80	P	19	70 112	p
	11 17	DC1 ^Q	02	31 49	1	10	51 81	Q	10	71 113	q
	12 18	DC2 ^R	03	32 50	2	13	52 82	R	13	72 114	r
	13 19	DC3 ^S	04	33 51	3	1F	53 83	S	1F	73 115	s
	14 20	DC4 ^T	05	34 52	4	14	54 84	T	14	74 116	t
	15 21	NAK ^U	06	35 53	5	16	55 85	U	16	75 117	u
	16 22	SYN ^V	07	36 54	6	2F	56 86	V	2F	76 118	v
	17 23	ETB ^W	08	37 55	7	11	57 87	W	11	77 119	w
	18 24	CAN ^X	09	38 56	8	2D	58 88	X	2D	78 120	x
	19 25	EM ^Y	0A	39 57	9	2C	59 89	Y	2C	79 121	y
	1A 26	SUB ^Z	34	3A 58	:	15	5A 90	Z	15	7A 122	z
01	1B 27	Esc ^[	33	3B 59	;		5B 91	[		7B 123	{
	1C 28	FS ^\	2B	3C 60	<		5C 92	\		7C 124	
	1D 29	GS ^]	0B	3D 61	=		5D 93	]		7D 125	}
	1E 30	RS ^^	2B	3E 62	>	29	5E 94	^		7E 126	~
	1F 31	US ^_	0C	3F 63	?	35	5F 95	_	53	7F 127	DEL

# Welches Wort ist hier kodiert?

---

01001010 01101111 01101110 00100000 01010011 01101110 01101111 01110111

74

111

110

32

83

110

111

119

J

o

n

\w

S

n

o

w

# UNICODE : “One format to rule them all”

---

Unicode-Standard kodierte elementare Zeichen mittels *Code Points* (*Unicode Number*).

<https://unicode-table.com>

Kodierung mit <https://www.branah.com/unicode-converter>: Jon Snow

## UTF-32

**0000004a0000006f0000006e 000000530000006e0000006f00000077**

1 Zeichen wird mittels 4 Byte kodiert.

2.147.483.647 unterschiedliche Zeichen

Nachteil: Sehr viel Speicherplatz; Vorteil: Direktzugriff da Code Point

## UTF-8

**\x4a\x6f\x6e \x53\x6e\x6f\x77**

variabler Länge zugeordnet

auch 4 [Byte](#)

alle Unicode-Zeichen abbildbar, weniger Speicher, da **Algorithmus(!)**

(siehe Computerphile: <https://www.youtube.com/watch?v=MijmeoH9LT4&t=334s>)

# “UTF-8 Hack”

- ASCII, Rückwärtskompatibel
- 0000 0000 kann null sein  
<https://unicode-table.com/en/#0000>
- Speicherplatz, kein Index

110x ... definiert Beginn und  
wie viele Bytes noch kommen

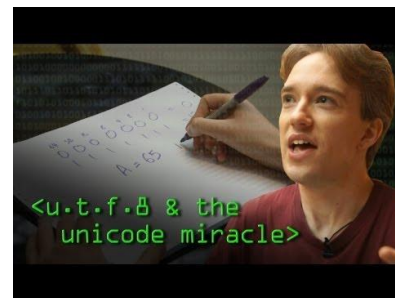
10 ... Beginn eines nächsten Byte

x ... Platz für das eigentliche Zeichen

110x xxxx 10xx xxxx

1110 xxxx 10xx xxxx 10xx xxxx

1111 11xx 10xx xxxx 10xx xxxx ...



UTF-8 Algorithmus: 6:40

# UTF-8

---

- **Empfehlung:** Verwende, wo immer es geht, UTF-8!
- Unicode bietet benötigte Freiheit im Web
- alle Zeichen
- Server, Browser, Datenbanken, Programmiersprachen, Editoren.

# Zusammenfassung

---

- Zahlensysteme: Binär, Dezimal, Hexadezimal
- Zahlenumwandlung
- Bits und Bytes
- Rechnen mit Binärzahlen
- Overflow und negative Zahlen
- Zeichenkodierung ASCII, UNICODE, UTF

# In der Konsole

---

ls	... zeigt uns an wo wir sind
cd	... so können wir einen Ordner weiter gehen
cd ..	... einen Ordner zurück
mkdir	... einen neuen Ordner erstellen ( mkdir MeinGDIOrdner )
touch	... ein neues Textfile erstellen ( touch encoding.txt)
>>	... echo Hier ist ein Text >> encoding.txt
file -i	... zeigt uns das Encoding einer Datei an (file -i encoding.txt)
hexdump	... wandelt text in ihre Hex-Kodierung um (hexdump encoding.txt.)

# Übung

---

Wenden wir das an auf

- eine .txt Datei die leere und noch nie bearbeitet wurde
- eine .txt Datei in der “hallo” steht
- eine .txt Datei in der “hallo Öl” steht.
- verwenden wir hexdump auf das .txt.

Was verändert sich und warum?

<https://www.tecmint.com/convert-files-to-utf-8-encoding-in-linux/>

<https://www.geeksforgeeks.org/hexdump-command-in-linux-with-examples/>



# Assignment 1 (Auswahl bis 24.10 10:00)

---

1. Kodiere deinen Vornamen (Text) in hexadezimal, dezimal und in binär. Nutze dazu die ASCII Tabelle. Zeige für einen Wert wie die manuelle Umrechnung (binär->hex; binär -> dezimal) funktioniert.

Text: Christopher

Hex: 43 68 ...

Binär: 01000011 01101000 ...

Dezimal: ...

2. Wandle die dezimal Zahlen 12 und 8 in Binärzahlen um und führe folgende arithmetischen Operationen durch:

2.1)  $12 + 8 = ?$

2.2)  $12 - 8 = ?$

2.3)  $8 - 12 = ?$

2.4)  $5 * 10 = ?$

2.5)  $9 : 3 = ?$

3. Wie wird die Zeichenkette "123" gespeichert, wenn die UTF8-Kodierung verwendet wird? Welche Codepoints sind das? Wie sieht die binäre Form davon aus.
4. Kann die Zahl 123 in einem Byte als Binärzahl gespeichert werden? Auch im 2-er Komplement?

# Assignment 1

---

5. Kopiere das tschechische Wort *Žluté* in einem beliebigen Texteditor und speichere es ab. Verwende den Befehl *hexdump* auf dieses file. Nimm das Ergebnis das herauskommt und konvertiere es nach Binär (<https://www.rapidtables.com/convert/number/hex-to-binary.html>) und dieses Ergebnis nach ASCII <https://www.rapidtables.com/convert/number/binary-to-ascii.html>.

Welches Ergebnis bekommst du und warum?