

Codierung von Zahlen

► Natürliche Zahlen

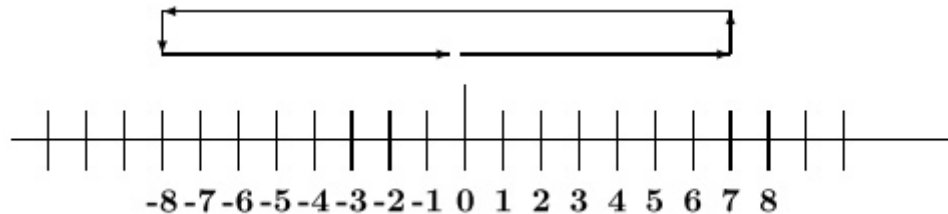
► Wertebereiche positiver ganzer Zahlen:

Anzahl Bits	Speichergröße	Wertebereich	Standardname
N = 2		0, ..., 3	
N = 3		0, ..., 7	
N = 4		0, ..., 15	
N = 8	1 Byte	0, ..., 255	unsigned shortint
N = 16	2 Byte	0, ..., 65.535	unsigned integer
N = 32	4 Byte	0, ..., 4.294.967.295	unsigned longint

- **Definition:** Bei der Darstellung natürlicher Zahlen im Rechner beschränkt man sich auf n–Bit Worte. Natürliche Zahlen 0, 1, 2, 3, werden durch die Binärdarstellung codiert. In der Binärdarstellung beschreibt die Folge von 0en und 1en die Koeffizienten der Potenzen im Stellenwertsystem zur Basis 2.

► Ganze Zahlen

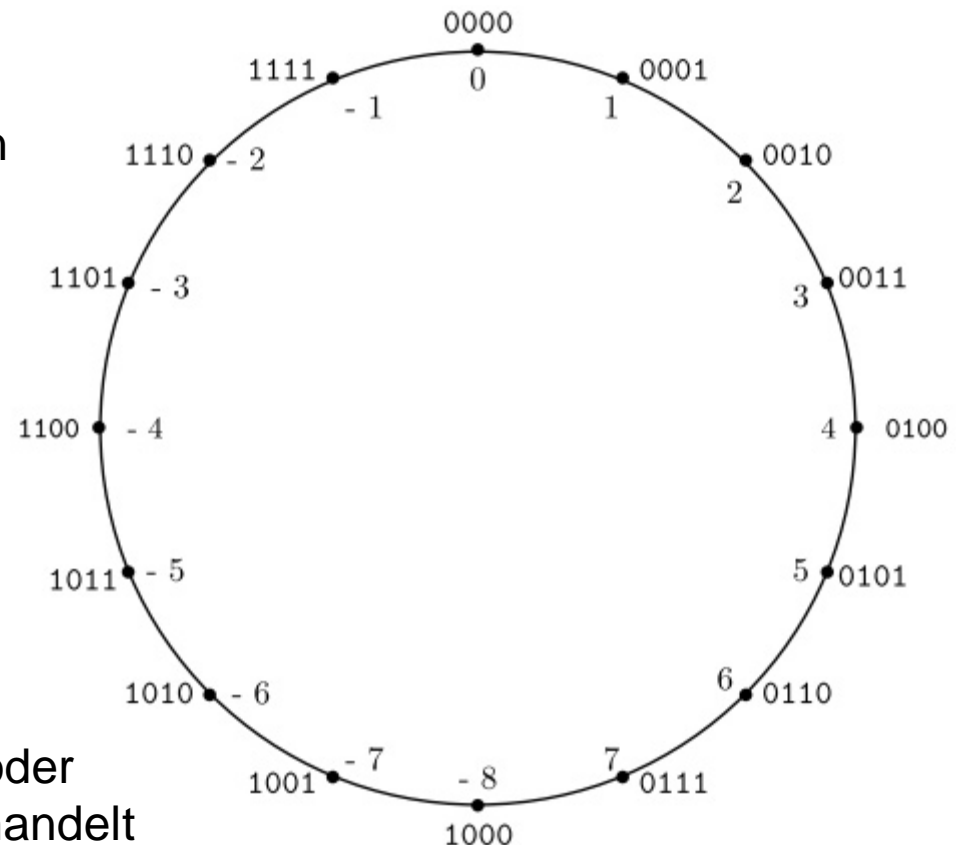
- Wie wird das Vorzeichen negativer Zahlen codiert?
 - Mit 4 Bits lassen sich 16 ganze Zahlen darstellen.
 - In der Praxis wird damit der Zahlenbereich von -8 bis +7 dargestellt.



- Bei der Zweierkomplementdarstellung stellt das höchste Bit somit das Vorzeichen dar.

► Ganze Zahlen

- Codierung von negativen Zahlen in Zweierkomplementdarstellung.
- Somit kommt ein Prozessor mit einem einzigen Addierwerk aus.
- Multiplikation & Division werden ebenfalls über wiederholte Addition durchgeführt.
- Ob es sich um eine binärcodierte oder Zweierkomplement-codierte Zahl handelt ist dabei unwichtig, addiert wird immer auf die gleiche Weise



► Ganze Zahlen

► Sonderfälle:

- **Überlauf:** Kein Problem, wird in der Regel ignoriert. **Beispiel:** $54 - 30 = 24$

$$\begin{array}{r} 011\ 0110_2 \\ +_2\ 110\ 0010_2 \\ = (1)001\ 1000_2 \end{array}$$

- **Bereichsüberschreitung:** tritt auf, falls das Ergebnis einer Addition zweier Zweierkomplementzahlen nicht innerhalb des Definitionsbereichs $[-2^{N-1}, +2^{N-1} - 1]$ dargestellt werden kann. **Beispiel:** $-4 - 5 = -9$

Heraus kommt ein unsinniges Ergebnis (+7).

$$\begin{array}{r} 1100 \\ +\ 1011 \\ = (1)0111 \end{array}$$

- **Lösung:** einfügen einer Schutzstelle:

$$-4 - 5 = -9$$

- Ein Prozessor besitzt einen Fehlererkennungsmechanismus, der automatisch eine Schutzstelle hinzufügt.

$$\begin{array}{r} 0\ 1100 \\ +\ 0\ 1011 \\ =\ 1\ 0111 \end{array}$$

► Ganze Zahlen

► Sonderfälle:

► Bereichsüberschreitung:

Fehlererkennung bei Bereichsüberschreitung mittels folgender KO-Kriterien:

► $a_n + b_n - c = -1$

► $a_n + b_n + c = +2$

► Am nebenstehenden Beispiel:

$$a_n + b_n - c = -1$$

$$1 + 1 - 1 \nless -1$$

$$a_n + b_n + c = +2$$

$$1 + 1 + 1 \nless +2$$

- 4 - 5 = - 9

a_n	→	1100
b_n	→ +	1011
<hr/>		
c	→	(1) 0111

- 4 - 5 = - 9

	0	1100
+	0	1011
<hr/>		
=	1	0111

► **Schlussfolgerung:** Bereichsüberschreitung → Schutzstelle einfügen

► Ganze Zahlen

► Sonderfälle:

► Bereichsüberschreitung:

Fehlererkennung bei Bereichsüberschreitung mittels folgender KO-Kriterien:

► $a_n + b_n - c = -1$

► $a_n + b_n + c = +2$

► Am nebenstehenden Beispiel:

$$a_n + b_n - c = -1$$

$$1 + 0 - 1 \neq -1$$

$$a_n + b_n + c = +2$$

$$\underline{1 + 0 + 1 == +2} \leftarrow \text{Treffer!}$$

► **Schlussfolgerung:** Keine Bereichsüberschreitung

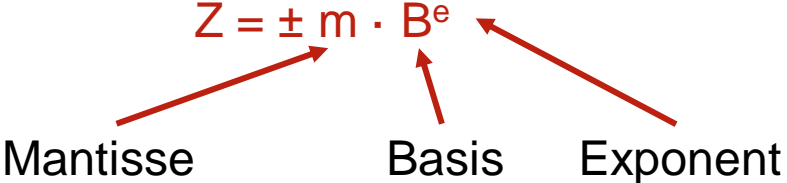
$$\begin{array}{r} -3 + 5 = +2 \\ a_n \xrightarrow{\text{cyan}} 1101 \\ b_n \xrightarrow{\text{red}} 0101 \\ \hline c \xrightarrow{\text{green}} (1)0010 \end{array}$$

$$\begin{array}{r} -3 + 5 = +2 \\ 1101 \\ + 0101 \\ \hline = (\cancel{1})0010 \end{array}$$

- ▶ Gleitkommazahlen (engl.: floating point numbers)
 - ▶ Darstellung von Gleitkommazahlen (auch Gleitpunktzahlen oder Realzahlen) in Computern geschieht nach dem IEEE 754 Gleitkommaformat.
 - ▶ Ein Computer, der Gleitpunktzahlen durch eine Folge von 32 Bit (= 4 Bytes) darstellt, kann höchstens 2^{32} viele Werte exakt darstellen.
 - ▶ Schreibweise von Gleitpunktzahlen: $1,2345 \times 10^5$
 - ▶ Unterschied bei Computern zur oben dargestellten Schreibweise: Exponent wird nicht zur Basis 10 sondern zur **Basis 2** genommen.

► Gleitkommazahlen

- Grundlage des IEEE 754 Formats ist die Darstellung einer Gleitkommazahl Z in halblogarithmischer Schreibweise:

$$Z = \pm m \cdot B^e$$


Mantisse Basis Exponent

- Beispiel $0,5_{10}$:

$$\frac{1}{2} = 0.1_2 \cdot 2^0 = 1.0_2 \cdot 2^{-1} = 0.01_2 \cdot 2^1 = 0.0001_2 \cdot 2^3$$

- Beispiel zeigt, die Darstellung in dieser Form ist nicht eindeutig.
 - Um die Eindeutigkeit der Darstellung zu gewährleisten, werden Gleitkommazahlen in *binärer, normalisierter* Form ausgedrückt.

► Gleitkommazahlen

► *Binäre, normalisierte Form:*

- die Mantisse in Binärdarstellung hat die Form **$1.b_1 b_2 \dots b_n$** ;
- Das binäre Komma wird also so weit nach links verschoben - bei gleichzeitiger Anpassung des Exponenten - dass das Komma hinter der führenden 1 steht.
- Werden normalisierte Mantissen vorausgesetzt, dann ist das erste Bit immer eine 1. Da es unnötig ist, dieses Bit zu speichern, ist dieses Bit implizit - man sagt dazu auch **hidden bit**.
- **Beispiel $7,625_{10}$:**
$$\begin{aligned} &= 4 + 2 + 1 + 1/2 + 1/8 \\ &= 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 111.101 \\ &= \mathbf{1.11101 \times 2^2} \end{aligned}$$

► Gleitkommazahlen

- Im IEEE–Standard wird eine Gleitkommazahl also abgespeichert in der Form:
 - Vorzeichen–Bit
 - Mantissebits
 - und Exponent Potenz von 2.
- Eine normalisierte, nichtverschwindende Gleitkommazahl hat also die Form: $\pm 1. \mathbf{bbb} \dots \mathbf{b} \times 2^{\pm e}$, wobei b entweder 0 oder 1 ist.
- Zur Darstellung des Vorzeichens wird ein Vorzeichenbit verwendet:
 - Vorzeichen–Bit = 0 \rightarrow positiver Wert
 - Vorzeichen–Bit = 1 \rightarrow negativer Wert

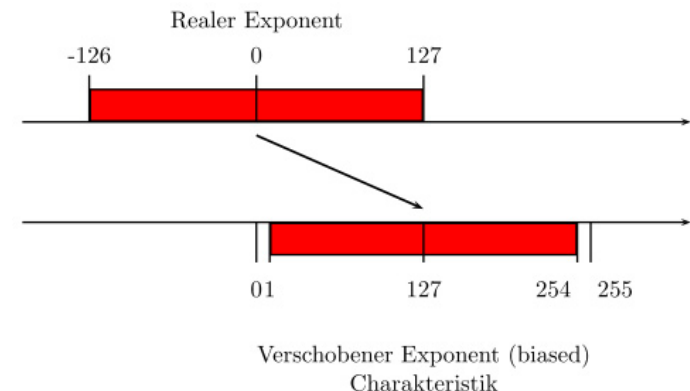
► Gleitkommazahlen

- Die Mantisse wird binär dargestellt.
- Da Exponenten von Gleitkommazahlen positiv oder negativ sein können, wird der Exponent als ganze Zahl in der sogenannten **Biasdarstellung** gespeichert.
 - Das Vorzeichen des Exponenten wird in der Biasdarstellung dadurch codiert, dass der Zahlenbereich um einen Basiswert (bias) verschoben wird.
 - Den verschobenen Exponent nennt man **Charakteristik** oder **biased exponent**.
 - Der Grund für die Verwendung dieser Codierung - und nicht der Zweierkomplementdarstellung - liegt darin, dass Exponenten für Gleitkommaarithmetik auf diese Art schnell verglichen werden können.
 - Für die Umrechnung gilt also:
$$\text{Charakteristik} = \text{Biased Exponent} = \text{Exponent} + \text{Bias}$$
- Kleinster und der größter Wert bei Charakteristik sind für Ersatzdarstellungen reserviert.

► Gleitkommazahlen

► Beispiel: Darstellung des Exponenten mit $n = 8$ Bit

- Der reale Exponent kann damit die 256 ganzzahligen Werte von -127 bis +128 annehmen.
- Dieser Zahlenbereich wird nun um den Betrag 127 in den positiven Zahlenbereich verschoben. Die Charakteristik ist damit eine Zahl im ganzzahligen Bereich [1, 254].
- Die daraus resultierenden Randwerte 0 und 255, binär 0000 0000 und 1111 1111 haben eine besondere Bedeutung, sie codieren die 0 bzw. $+\infty$.
- Ist der Exponent 45, dann ist:
Charakteristik = $45 + 127$
= 172 \rightarrow 10101100
- Ist der Exponent -33, dann ist:
Charakteristik = $-33 + 127$
= 94 \rightarrow 01011110



► Gleitkommazahlen

► Beispiel *float*:



- Das *float*-Format der IEEE 754 Gleitkommadarstellung sieht vor, dass eine Gleitkommazahl in einem 32-Bit Format abgespeichert wird.
- Dieses Format wird folgendermaßen aufgeteilt:
 - 1 Bit für das Vorzeichen,
 - 8 Bits für den Exponenten
 - die restlichen 23 Bits sind für die Mantisse reserviert.
- Für die Speicherung des Exponenten stehen also 8 Bits als Speicherplatz zur Verfügung. Da der Exponent in der Biasdarstellung deklariert ist, kann dieser Teil der Gleitkommazahl die Werte -126 bis +127 annehmen

► Gleitkommazahlen

► Beispiel $x = 472,56640625$:

$$\begin{aligned}x &= 472,56640625_{10} \\&= 111\ 011\ 000.100\ 100\ 01_2 \\&= 1.110\ 110\ 001\ 001\ 000\ 1_2 \times 2^8\end{aligned}$$

► Damit wird diese Zahl im *float*-Format folgendermaßen dargestellt:

- Das Vorzeichen 1 Bit \rightarrow 0
- Der Exponent ist dezimal 8 \rightarrow die Biasdarstellung ist $8 + 127 = 135$
 $= 1000\ 0111$
- Die signifikanten Nachkommastellen = 11011000100100010000000
- **Damit: $472,56640625 = 0\ 10000111\ 11011000100100010000000$**

► Gleitkommazahlen

- Die IEEE 754 Darstellung ermöglicht keine Darstellung der Zahl 0.0 oder ∞ . Die Gleitkommadarstellung reserviert jedoch spezielle Bitmuster für diese Zahlen.
- Es gilt die Konvention:
 - Vorzeichen: 0 oder 1
 - Biased Exponent: 0000 0000
 - Mantissenbits: Alle 0
 - **Gleitkommazahl: 0.0**
- und:
 - Vorzeichen: 0 oder 1
 - Biased Exponent: 1111 1111
 - Mantissenbits: Alle 0
 - **Gleitkommazahl: $\pm \infty$**

► Übung

- 1. Wandeln Sie die folgenden Zahlen in die binäre, normalisierte Gleitkommaschreibweise um:
 - a) 109_{10}
 - b) $-18,4_{10}$
 - c) $3FD.A_{16}$
 - d) $-5D_{16}$
 - e) 1001010101010101_2
 - f) 0.78515625_{10}
 - g) 101.1101_2

Zeichensätze

- ▶ Generell unterscheidet man zwischen 4 Arten von Zeichen:
 - ▶ Buchstaben: A, B, C, ... , Z, a, b, c, ... , z
 - ▶ Ziffern: 0, 1, 2, ... , B-1 (B = Basis)
 - ▶ Sonderzeichen:
 - ▶ Für arithmetische Operationen: +, -, *, /
 - ▶ Interpunktionszeichen: ; , : . ! ? usw.
 - ▶ Sonstige: #, %, &, | usw.
 - ▶ Landesspezifische Zeichen, wie ä, ö, ü, ß, ø, ç, œ, usw.
 - ▶ Nicht darstellbare Zeichen
 - ▶ Auch Steuerzeichen oder nicht-druckbare Zeichen genannt
 - ▶ Beispiele: CR = Carriage Return (Wagenrücklauf)
LF = Line Feed (Zeilenvorschub)
- ▶ Koppelt man einzelne Zeichen zur Darstellung irgendwelcher Informationen, dann entsteht eine Zeichenfolge, ein sogenanntes **Wort** (engl. word).

- ▶ Beispiele für Zeichenvorräte:
 - ▶ Die Menge der Dezimalziffern $D := \{ 0, 1, 2, \dots, 9 \}$
 - ▶ Die Menge der Zeichen auf einer PC-Tastatur
 - ▶ Die Buchstaben des deutschen Alphabets samt Umlauten
 - ▶ Die Menge der griechischen Buchstaben $\alpha, \beta, \gamma, \delta, \dots$,
 - ▶ Die binären Zeichenvorräte wie $B = \{ 0, 1 \}$ oder $\{ \text{TRUE}, \text{FALSE} \}$
 - ▶ Die Menge der Ampelfarben $\{ \text{rot}, \text{gelb}, \text{grün} \}$
 - ▶ Der Morse Code $\{ \bullet, - \}$, wobei das Zeichen \bullet für '*kurz*' und das Zeichen $-$ für '*lang*' steht.

- ▶ Ein Alphabet ist ein Zeichenvorrat mit einer darauf definierten totalen Ordnung. Üblicherweise wird in der Praxis nicht streng zwischen Zeichenvorrat und Alphabet unterschieden.

Codierung von Zeichen

- ▶ Informationen, die für Menschen verständlich sind, werden durch Texte oder Zahlen dargestellt.
- ▶ Die zu verarbeitenden Informationen sind in Computern in Form von Strings von Bits codiert. Jedes Bit solch eines Bitstrings nimmt dabei entweder den Wert 0 oder 1 an.
- ▶ Bits sind in Form klassischer physikalischer Zustände realisiert:
 - ▶ Übertragung auf einer Busleitung:
 - ▶ Spannung (0 Volt) → logische 0
 - ▶ Spannung (5 Volt) → logische 1
 - ▶ Übertragung in Glasfasern:
 - ▶ Licht aus → logische 0
 - ▶ Licht an → logische 1
 - ▶ Speichern in RAM oder ROM Chip:
 - ▶ 'Kondensator' nicht geladen → logische 0
 - ▶ 'Kondensator' geladen → logische 1
 - ▶ Speichern auf CD-ROM:
 - ▶ Land → logische 0
 - ▶ Pit → logische 1
 - ▶ Speichern auf Festplatte:
 - ▶ Bereich nicht magnetisiert → logische 0
 - ▶ Bereich magnetisiert → logische 1

- ▶ *Ein Bit ist die minimale Informationseinheit, die ein Computer verarbeiten kann.*
- ▶ **Zu lösendes Problem:** Wie können die für Menschen verständlichen Zeichen wie Buchstaben, Zahlen, usw. in Zeichen abgebildet werden, die ein Computer speichern und verarbeiten kann?
- ▶ **Lösung:** die Zeichen des lateinischen Alphabets, Ziffern usw. müssen durch das binäre Alphabet $B = \{ 0, 1 \}$ geeignet dargestellt oder codiert werden. Eine solche zweiwertige Codierung nennt man **Binär-Codierung**.
- ▶ **Codierung nach DIN 44300:**
 - ▶ 1. Eine Vorschrift für die eindeutige Zuordnung (Codierung) der Zeichen eines Zeichenvorrates zu denjenigen eines anderen Zeichenvorrates (Bildmenge).
 - ▶ 2. Als Code bezeichnet man auch den bei der Codierung als Bildmenge auftretenden Zeichenvorrat.

- ▶ Beispiele für Binärcodierungen:
 - ▶ BCD-Code (Binary coded decimal) = 6 Bit
 - ▶ ASCII-Code (American Standard Code for Information Interchange)
 - ▶ ASCII-Code = 7 Bit
 - ▶ Erweiterter ASCII-Code = 8 Bit
 - ▶ EBCDIC (extended binary coded decimal interchange code) = 8 Bit
 - ▶ findet in IBM Großrechenanlagen Verwendung
 - ▶ Unicode = ursprünglich 16-Bit-Kodierung

- ▶ Erweiterter ASCII-Code
 - ▶ Lange Zeit Standard in der Computerwelt
 - ▶ Zuerst 1968 in seiner ursprünglichen 7-Bit-Form durch ANSI freigegeben
 - ▶ tabellarische Zuordnungsregel welche die Darstellung von Zeichen in Form binärer Zahlen vorschreibt
 - ▶ 8-Bit Länge = 1 Byte = 256 darstellbare Zeichen

1.3 Zeichensätze

Codierung von Zeichen

► ASCII-Code:

Regular ASCII Chart (character codes 0 - 127)															
000	{nul}	016	► (dle)	032	sp	048	0	064	ø	080	P	096	`	112	p
001	⓪ (soh)	017	◄ (dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	Ⓢ (stx)	018	↑ (dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	▼ (etx)	019	!! (dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	Ⓡ (eot)	020	¶ (dc4)	036	\$	052	4	068	D	084	T	100	d	116	t
005	Ⓜ (enq)	021	§ (nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	Ⓜ (ack)	022	— (syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	• (bel)	023	↓ (etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	▣ (bs)	024	↑ (can)	040	(056	8	072	H	088	X	104	h	120	x
009	(tab)	025	↓ (em)	041)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	␣ (vt)	027	← (esc)	043	+	059	;	075	K	091	[107	k	123	{
012	Ⓡ (np)	028	L (fs)	044	,	060	<	076	L	092	\	108	l	124	
013	(cr)	029	→ (gs)	045	-	061	=	077	M	093]	109	m	125	}
014	Ⓢ (so)	030	▲ (rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	Ⓢ (si)	031	▼ (us)	047	/	063	?	079	O	095	_	111	o	127	o

Extended ASCII Chart (character codes 128 - 255)															
128	Ç	143	Š	158	Ž	172	Ł	186		200	Ł	214	Œ	228	Σ
129	ù	144	É	159	f	173	;	187	}]	201	Œ	215	Œ	229	σ
130	é	145	æ	160	á	174	«	188	}]	202	Œ	216	Œ	230	μ
131	â	146	Æ	161	í	175	»	189	}]	203	Œ	217	Œ	231	τ
132	ä	147	ó	162	ó	176	█	190	}]	204	Œ	218	Œ	232	Φ
133	à	148	ö	163	ú	177	█	191	}]	205	=	219	█	233	©
134	å	149	ò	164	ñ	178	█	192	}]	206	Œ	220	█	234	Ω
135	ç	150	û	165	Ñ	179	█	193	}]	207	=	221	█	235	δ
136	ê	151	ù	166	ª	180	█	194	}]	208	Œ	222	█	236	∞
137	ë	152	ÿ	167	º	181	█	195	}]	209	Œ	223	█	237	φ
138	è	153	Ö	168	¿	182	█	196	—	210	Œ	224	α	238	ε
139	ï	154	Ü	169	¬	183	█	197	—	211	Œ	225	ß	239	π
140	î	155	ó	170	¬	184	█	198	—	212	Œ	226	Γ	240	≡
141	ì	156	£	171	½	185	█	199	—	213	Œ	227	π	241	±
142	Ä	157	¥											255	

▶ Beispiel:

- ▶ Zeichenkette: **DHBW Mannheim**
- ▶ Dezimal: 068 072 066 087 032 077 097 110 110 104 101 105 109
- ▶ Binär: 01000100 01001000 01000010 01010111 00100000 01001101
01100001 01101110 01101110 01101000 01100101 01101001 01101101

▶ Anmerkung:

- ▶ Little endian Codierung:
 - ▶ Bei PCs und allen verwandten und somit gängigen Computern verwendet
 - ▶ Bit ganz rechts = niedrigstwertiges Bit
 - ▶ Bit ganz links = höchstwertiges Bit
- ▶ Big endian Codierung:
 - ▶ Exakt entgegengesetzt zur Little endian Codierung
 - ▶ Findet u.a. bei IBM Großrechnern Verwendung

► Unicode

- Codierung von Zeichen über sogenannte **Codepunkte**
- Der Codepunkt ist zunächst nicht die Art und Weise, wie das Zeichen im Rechner dargestellt wird, sondern ein Verfahren, alle möglichen Zeichen, die es in den versch. Sprachen weltweit gibt, systematisch aufzulisten.
- Ursprünglich 16-Bit codiert (65 536 Codepunkte), stellt er heute 1.114.112 Codepunkte bereit und wird stetig weiterentwickelt.
- Beispiel: **Hello**
 - Codepunkte: U+ 0048 U+ 0065 U+ 006C U+ 006C U+ 006F
 - Dies ist zunächst nur eine Kette von Zahlen, also eine Reihe von Codepunkten. Es ist bisher noch keine Aussage darüber gemacht, wie dies gespeichert wird oder in einer E-Mail dargestellt wird.
 - Dazu wurden verschiedene Codierungsformate definiert, wie UTF-8, UCS-2, UTF-16, UCS-4 und UTF-32.
 - UTF = Unicode Standard; UCS = ISO Standard

1.3 Zeichensätze

Codierung von Zeichen

► Interpretation von Daten

```
FA 98 47 96 36 43 16 DA 14 BB 8D AD 62 88 AA 8C
3B 8E ED 90 32 07 F9 EF 57 E1 47 2A A5 51 9B 71
3C 82 01 C7 6C 74 C8 C1 35 0C 2D 18 6C 2A 9C 64
F2 07 5C 67 FF 00 AF 52 7E E0 29 03 1C ED 38 04
F5 1D 7F 1E 7D 0F F3 C5 21 36 D2 B0 FB 7B 9F 22
12 1A D3 73 16 C9 75 1F 7B 93 8C 93 ED 51 80 43
39 74 0A CC C4 85 1C 8F 6E 7E 94 45 6A 2E 64 F3
65 C4 D0 A7 EE D6 31 CA 9F 7E 38 EB FD 6A 09 2D
D6 CE E9 AD C4 99 5C 6F 01 D8 8D BE DC 50 EF 60
8A 57 D0 1E 2C 28 08 37 A6 3E 75 6C 01 9F 62 7F
```

Bytefolge

```
•ÿGû6C. r.ŋ i; bē~i
;ÄÝÉ2.``'WBG*ÑQøç
<é.Ältll5.-.1*£d
=. \g .»R~Ó) ..Ý8.
$.□.} .%†!6Ê||: {f"
..Ës.ŕu. {ôïôÝQÇC
9t. †-à.Än~öEj. d%
e-ð°-Í1lf~8Û* j.-
í†Ú;-Ö\o. ïiŕ. P'`
èWð., (.7*>ul.fb□
```

ASCII-
Interpretation



jpg-
Interpretation

► Übung:

- Codieren Sie die folgende Zeichenfolge mithilfe der ASCII-Tabelle zuerst als Dezimalzahlen, und danach in die Binärschreibweise:

“Frohe Weihnachten! ;-P”