Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
- Rechnernetzwerke

Grundlagen der Informatik

- Logische und mathematische Grundlagen
  - Binäre Informationsdarstellung
  - Elementare Logikoperationen
  - Zahlensysteme
  - Rechenoperationen als Logikoperationen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
- Rechnernetzwerke

Informationsdarstellung im Computer

Grundsätzliche Regel:

- Informationen können genau dann in Computer eingespeist werden, wenn Sie adäquat und vollständig als 0/1-Daten (*Binärdaten*) darstellbar sind.
- Verarbeitungsschritte auf Binärdaten sind genau dann durch den Computer ausführbar, wenn sie sich auf elementare Logik-operationen zurückführen lassen.

Bits and Bytes

- 1 bit: 1 binäre Entscheidung
  - 2 Möglichkeiten: 0 oder 1, wahr oder falsch, ...
- 1 Byte: eine Abfolge von 8 bits
  - $2^8 = 256$ Möglichkeiten
  - traditionell die kleinste Größeneinheit für Speicherung:
    - 8-bit: 1 Wort = 1 Byte
  - Neuere Rechner arbeiten mit größeren Einheiten
    - 16-bit Rechner: 1 Wort = 2 Bytes
    - 32-bit Rechner: 1 Wort = 4 Bytes
    - 64-bit Rechner: 1 Wort = 8 Bytes
- Weitere Einheiten
  - kilo: 1 kB = 1024 Bytes ($1024 = 2^{10}$)
  - Mega: 1 MB = 1024 kB
  - Giga: 1 GB = 1024 MB
  - Tera: 1 TB = 1024 GB

Elementare Logikoperationen

- Binärwert 1 wird als „wahr“, Binärwert 0 als „falsch“ interpretiert.
- *Logisches Und:* Für zwei Binärwerte *a* und *b* ist $\boldsymbol{a \wedge b = 1}$ genau dann, wenn $a = 1$ und $b = 1$ ist
- *Exklusiv-Oder:* $\boldsymbol{a \neq b = 1}$ genau dann, wenn $a = 1$ oder $b = 1$ gilt, aber nicht beides zugleich
- *Inklusiv-Oder:* $\boldsymbol{a \vee b = 1}$ genau dann, wenn $a = 1$ oder $b = 1$ oder beides zugleich zutrifft
- *Negation:* $\boldsymbol{\neg a = 1}$ genau dann, wenn $a = 0$

Wahrheitstafel

- logische Operationen lassen sich mit einer sogenannten Wahrheitstafel darstellen
  - enthält alle möglichen Eingabe-Kombinationen (links)
  - das Ergebnis der Berechnung für jede Kombination (rechts)
- Wahrheitstafel für die vorhergehenden logischen Operationen:

a	b	$a \wedge b$	$a \neq b$	$a \vee b$	$\neg a$	$\neg b$
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	0	1	0	0

Wichtige Rechenregeln

- Kommutativgesetz

$$a \wedge b = b \wedge a \qquad a \vee b = b \vee a$$

- Assoziativgesetz

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c \qquad a \vee (b \vee c) = (a \vee b) \vee c$$

- Distributivgesetz

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \qquad a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

- Verschmelzung

$$a \vee (a \wedge b) = a \qquad a \wedge (a \vee b) = a$$

- De Morgan's Gesetz

$$\neg(a \wedge b) = \neg a \vee \neg b \qquad \neg(a \vee b) = \neg a \wedge \neg b$$

Darstellung von anderen logischen Funktionen

- Es gibt insgesamt $2^4 = 16$ verschiedene Möglichkeiten, 2 Eingaben *a* und *b* miteinander zu kombinieren
- diese lassen sich alle durch elementare logische Operationen darstellen

a 0011 b 0101	
0000	$y_0 = 0$
0001	$y_1 = a \wedge b$
0010	$y_2 = a \wedge \neg b$
0011	$y_3 = a$
0100	$y_4 = \neg a \wedge b$
0101	$y_5 = b$
0110	$y_6 = a \neq b$
0111	$y_7 = a \vee b$

a 0011 b 0101	
1111	$y_{15} = 1$
1110	$y_{14} = \neg a \vee \neg b$
1101	$y_{13} = \neg a \vee b$
1100	$y_{12} = \neg a$
1011	$y_{11} = a \vee \neg b$
1010	$y_{10} = \neg b$
1001	$y_9 = a \equiv b$
1000	$y_8 = \neg a \wedge \neg b$

Logische Schaltungen

- Die elementaren logischen Operationen lassen sich unmittelbar auf logische Schaltungen abbilden
- Es läßt sich sogar zeigen, daß sich alle logischen Operationen durch *eine* der folgenden grundlegenden Operationen darstellen lassen:
  - NAND (Negated AND): $\neg(a \wedge b)$
  - NOR (Negated OR): $\neg(a \vee b)$
- Beispiele:
  - Negation: $\neg a = \neg(a \vee a) = \neg(a \wedge a)$
  - Konjunktion: $a \wedge b = \neg(\neg(a \wedge b)) = \neg(\neg a \vee \neg b)$
  - Disjunktion: $a \vee b = \neg(\neg a \wedge \neg b) = \neg(\neg(a \vee b))$

Zahlendarstellung

Erinnerung aus der Schule:

- Unser Zahlensystem („Zehnersystem“) ist nicht *das* Zahlensystem schlechthin
- Sondern eben „nur“ das Zahlensystem zur Basis „zehn“.
- Jede andere Zahl 2, 3, 4, 5, ... ließe sich ebenso als Basis hernehmen

Beispiel: Die Zahl 28 130 im Zehnersystem:

$$28130 = 2*10^4 + 8*10^3 + 1*10^2 + 3*10^1 + 0*10^0$$

Aber auch:

$$28130 = 1*4^7 + 2*4^6 + 3*4^5 + 1*4^4 + 3*4^3 + 2*4^2 + 0*4^1 + 2*4$$

→ Darstellung im Vierersystem: 12 313 202

Zahlensysteme

- Binärsystem:
  - Basis 2 → Ziffern 0, 1
- Oktalsystem:
  - Basis 8 → Ziffern 0,1,2,3,4,5,6,7
- Hexadezimalsystem:
  - Basis 16 → Ziffern 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

- Oktal und Hexadezimalsystem werden oft als Kurzschreibweise für Binärsystem verwendet
  - Oktal: $8 = 2^3$
    $11111001011 \rightarrow 011,111,001,011 \rightarrow 3,7,1,3 \rightarrow 3713$ oktal
  - Hexadezimal: $16 = 2^4$
    $11111001011 \rightarrow 0111,1100,1011 \rightarrow 7,12,11 \rightarrow 7CB$ hexadezimal

Beispiel Zahlenumwandlung (Restklassenverfahren)

1995 : 2 = 997 Rest 1	1995 : 8 = 249 Rest 3	1995 : 16 = 124 Rest 11
997 : 2 = 498 Rest 1	249 : 8 = 31 Rest 1	124 : 16 = 7 Rest 12
498 : 2 = 249 Rest 0	31 : 8 = 3 Rest 7	7 : 16 = 0 Rest 7
249 : 2 = 124 Rest 1	3 : 8 = 0 Rest 3	
124 : 2 = 62 Rest 0		
62 : 2 = 31 Rest 0		
31 : 2 = 15 Rest 1		
15 : 2 = 7 Rest 1		
7 : 2 = 3 Rest 1		
3 : 2 = 1 Rest 1		
1 : 2 = 0 Rest 1		
→ 11111001011 binär	→ 3713 oktal	→ 7CB hexadezimal

Rückrechnung: $\mathbf{3713}_{\mathbf{OCT}} = 3*8^3 + 7*8^2 + 1*8^1 + 3*8^0 =$

$= ((3*8 + 7)*8 + 1)*8 + 3 = \mathbf{1995}_{\mathbf{DEC}}$

- Warum verwechseln Informatiker gerne Halloween und Weihnachten?

$$31\ \mathrm{OCT} = 25\ \mathrm{DEC}$$

Addition und Multiplikation

- funktionieren genau wie im Dezimalsystem
- Addition:

101011010	532	15A
+ 11011100	+ 334	+ DC
Ü 1 1 0 1 1 0 0 0	Ü 1 0 0	Ü 1 1
1000110110	1066	236

- Multiplikation:

10101101 * 10110

10101101
00000000
10101101
10101101
00000000

111011011110

Arithmetische Operationen

- *Erinnerung:* Bekanntlich arbeiten Computer auf dem *Binärsystem*, das heißt dem Zahlensystem mit Basis 2.

- Wichtige Erkenntnis: Arithmetische Operationen im Binärsystem lassen sich aus rein logischen Operationen synthetisieren.
  → Sind daher durch den Computer ausführbar.

- *Einfachstes, grundlegendes Beispiel*: Zwei einstellige Binärzahlen *a* und *b* sind zu addieren.
  → Das Ergebnis ist ein- oder zweistellig.

- Wenn man zuläßt, dass Zahlen mit Nullen beginnen, kann man das Ergebnis in jedem Fall zweistellig aufschreiben.

Auswertungstabelle

a	b	$a + b$
0	0	**0 0**
0	1	**0 1**
1	0	**0 1**
1	1	**1 0**

a	b	$a \neq b$
0	0	0
0	1	1
1	0	1
1	1	0

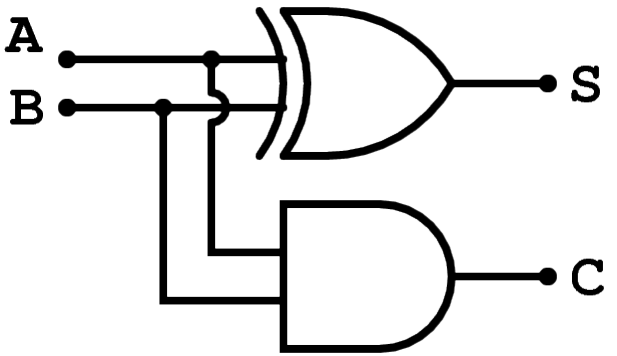
a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

Vergleich mit den Wahrheitstafeln:

- Die zweite (rechte) Stelle von $a + b$ ist gerade $a \leftrightarrow b$.
- Die erste (linke) Stelle von $a + b$ ist gerade $a \wedge b$.

Halbaddierwerk

a	b	$a + b$
0	0	**0 0**
0	1	**0 1**
1	0	**0 1**
1	1	**1 0**



Diese Schaltung/logische Verknüpfung nennt man *Halbaddierwerk*

- Die zweite (rechte) Stelle nennt man S (Sum).
- Die erste (linke) Stelle nennt man C (Carry, Übertrag).

→ Arithmetik auf Logik zurückgeführt.

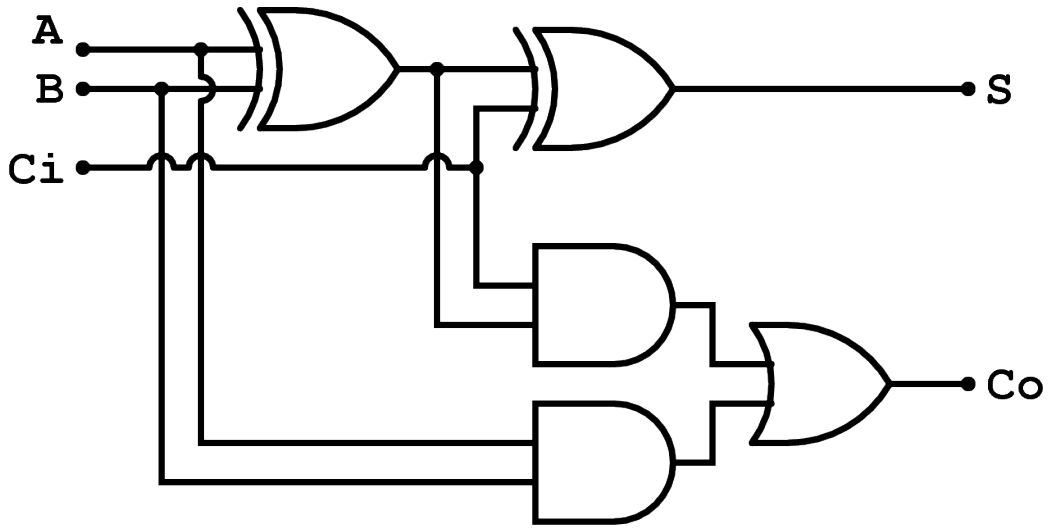
Volladdierwerk

- Um eine Addition über mehrere Stellen durchführen zu können, braucht man außer den Input-Signalen a und b noch ein Eingangssignal für das letzte Carry-Bit
- Ausgabe ist dann die Summe, und das neue Carry-Bit

a	b	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

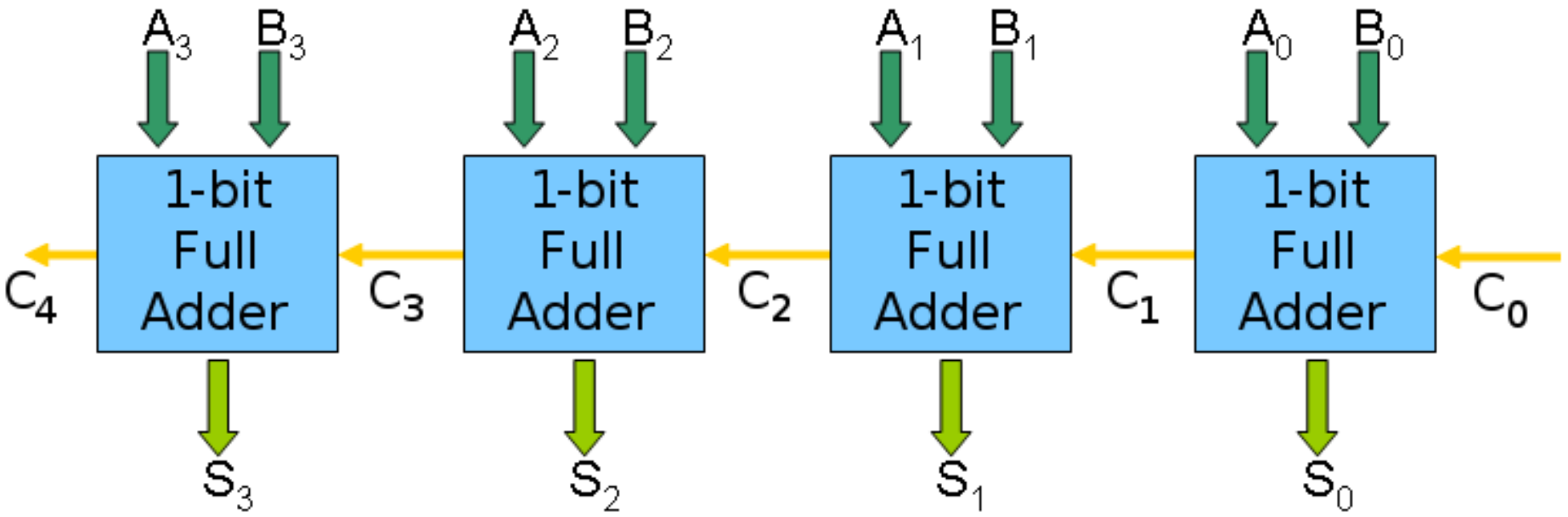
$$s = (a \Leftrightarrow b) \Leftrightarrow c_i$$

$$c_o = (a \wedge b) \vee (c_i \wedge (a \neq b)) = (a \wedge b) \vee (b \wedge c_i) \vee (c_i \wedge a)$$



Addition von ganzen Zahlen

- Die Summe zweier Zahlen aus beliebig vielen Bits kann nun durch Aneinander-Reihung mehrerer Volladdierer berechnet werden.



Multiplikation mit 2 Stellen

a	b	$a * b$
$a_1 a_2$	$b_1 b_2$	$c_1 c_2 c_3 c_4$
0 0	0 0	0 0 0 0
0 0	0 1	0 0 0 0
0 0	1 0	0 0 0 0
0 0	1 1	0 0 0 0
0 1	0 0	0 0 0 0
0 1	0 1	0 0 0 1
0 1	1 0	0 0 1 0
0 1	1 1	0 0 1 1
1 0	0 0	0 0 0 0
1 0	0 1	0 0 1 0
1 0	1 0	0 1 0 0
1 0	1 1	0 1 1 0
1 1	0 0	0 0 0 0
1 1	0 1	0 0 1 1
1 1	1 0	0 1 1 0
1 1	1 1	1 0 0 1

Umsetzung in Logik:

- $c_4 = a_2 \wedge b_{2,}$
- $c_3 = \neg(a_1 \wedge a_2 \wedge b_1 \wedge b_2) \wedge ((a_1 \wedge b_2)) \vee (a_2 \wedge b_1))$
- $c_2 = a_1 \wedge b_1 \wedge (\neg a_2 \vee \neg b_2)$
- $c_1 = a_1 \wedge a_2 \wedge b_1 \wedge b_2$

→ Prüfen Sie es nach!

Fundamentale Einsicht

Alle anderen arithmetischen Operationen auf Zahlen in Binärdarstellung lassen sich ebenfalls auf die elementaren logischen Operationen zurückführen.

→ Hier nicht weiter ausgeführt.

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
  - Zahlen
  - Zeichen
  - Texte
  - Farben
  - Bilder
- Computerprogramme als Binärdaten
- Betriebssysteme
- Rechnernetzwerke

Digitale Daten

- Konsequenz aus den letzten Folien: Alle Datenmanipulationen,
  - ◊ die sich als arithmetische Operationen auf natürlichen Zahlen formulieren lassen,
  - ◊ lassen sich auch allein mit Hilfe von logischen Operationen auf binären Wahrheitswerten formulieren
  - ◊ und lassen sich daher durch Computer erledigen
- In der Folge werden wir verschiedene Arten von Daten beispielhaft betrachten:

  Zahlen, Zeichen, Texte, Farben, Bilder

Darstellung ganzer Zahlen

- In der Regel werden alle ganzen Zahlen mit einer festen Stellenzahl *m* abgespeichert.

→ Von links ggf. mit Nullen aufgefüllt

- Einfache Möglichkeit zur Unterscheidung von positiven und negativen ganzen Zahlen: Das 1. Bit speichert das Vorzeichen:

  → 0 ≡ ”+“
  → 1 ≡ ”–“

(Ist in realen Computern aus gewissen Gründen nicht ganz so simpel realisiert → Zweierkomplement-Darstellung)

- *Konsequenz*: Die Zahlenmenge

$$[-(2^{m-1}-1), +2^{m-1}-1]$$

kann dargestellt werden.

Konkrete Umsetzung

111...111	$\equiv$	$-2^{m-1}+1$	
111...110	$\equiv$	$-2^{m-1}+2$	
...		...	
100...010	$\equiv$	-2	
100...001	$\equiv$	-1	
100...000	$\equiv$	-0	$\longleftarrow$!!!
000...000	$\equiv$	$+0$	$\longleftarrow$!!!
000...001	$\equiv$	$+1$	
000...010	$\equiv$	$+2$	
...		...	
011...110	$\equiv$	$+2^{m-1}-2$	
011...111	$\equiv$	$+2^{m-1}-1$	

Zahlenbereiche für ganze Zahlen

# Bits	*natürliche Zahlen*	*ganze Zahlen*
8	[0,255]	[-128,127]
16	[0,65535]	[-32768,32767]
32	[0,4294967295]	[-2147483648,2147483647]
64	[0,18446744073709551615]	[-9223372036854775808,9223372036854775807]

Reelle Zahlen: Gleitkommadarstellung

- Grundidee:

$$\text{Zahlenwert} = \pm m \cdot b^{e}$$

m: Mantisse
b: Basis = 2
e: Exponent

  - abgespeichert werden müssen nur der Exponent und die Mantisse (z.B. als Integer-Zahlen)

- Beispiel für Basis 10:
  - $\pi = 3.141593 = 3141593 \times 10^{-6}$
  - Abgespeichet wird das Zahlenpaar (+3141593,-6)

- In der Praxis etwas komplexer
  - andere Basis, Mantisse normiert auf 1.xxxx, etc.

Gleitkomma-Arithmetik

- Addition:
  - Angleichen der Exponenten (durch Verschieben der Stellen um die Differenz zwischen den Exponenten)
  - Dann Addition der Mantissen
  - Beispiel:
    - $1234 * 10^{2} + 1234 * 10^{-1} = 1234000 * 10^{-1} + 1234 * 10^{-1} = 1235234 * 10^{-1}$
- Multiplikation:
  - Multiplikation der Mantissen und Addition der Exponenten
  - Beispiel:
    - $1234 * 10^{2} * 1234 * 10^{-1} = 1234^{2} * 10^{(2 + (-1))} = 1522756 * 10^{1}$
- Rundung:
  - In der Praxis ist die Genauigkeit der Ergebnisse durch die Anzahl der Stellen, die für die Mantisse vorgesehen sind, begrenzt (Abschneiden der hinteren Stellen, Anpassen des Exponenten)

Darstellung von Zeichen

- Klein- und Großbuchstaben
- Ziffern
- Interpunktionszeichen
- Sonstige

Grundsätzlicher Ansatz:

Jedem Zeichen wird derart eine natürliche Zahl zugeordnet, dass je zwei Zeichen unterschiedliche Zahlen zugeordnet sind.

ASCII-Zeichen

- Auf den allermeisten Computern (auch bei uns) werden Zeichen nach dem *ASCII–Standard* kodiert.

- *ASCII*: American National Standard Code for Information Interchange (sprich ”Aas-kih“).

Jedem Zeichen wird eine Bitfolge aus sieben Bits zugeordnet.

→ 7 bits = Zahlen 0 ... 2^7–1 = 0 ... 127

- Beispiele:

’a’...’z’	≡	97 ...122
’A’...’Z’	≡	65 ... 90
’0’...’9’	≡	48 ... 57
’?’	≡	63
’%’	≡	37
’/’	≡	47
’&’	≡	38

Sonderzeichen

- ASCII kodiert auch alle gängigen Interpunktionszeichen.
- Auch beispielsweise typisch amerikanische Zeichen wie ’@’ (Nr. 64).
- Das Leerzeichen (auch *Space* oder *Blank* genannt), hat ASCII–Wert 32 (= 20 Hex).
- Es gibt auch ASCII–Werte, die

  - keinem Zeichen entsprechen,
  - sondern eine *Funktion* haben.
    z.B. newline

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

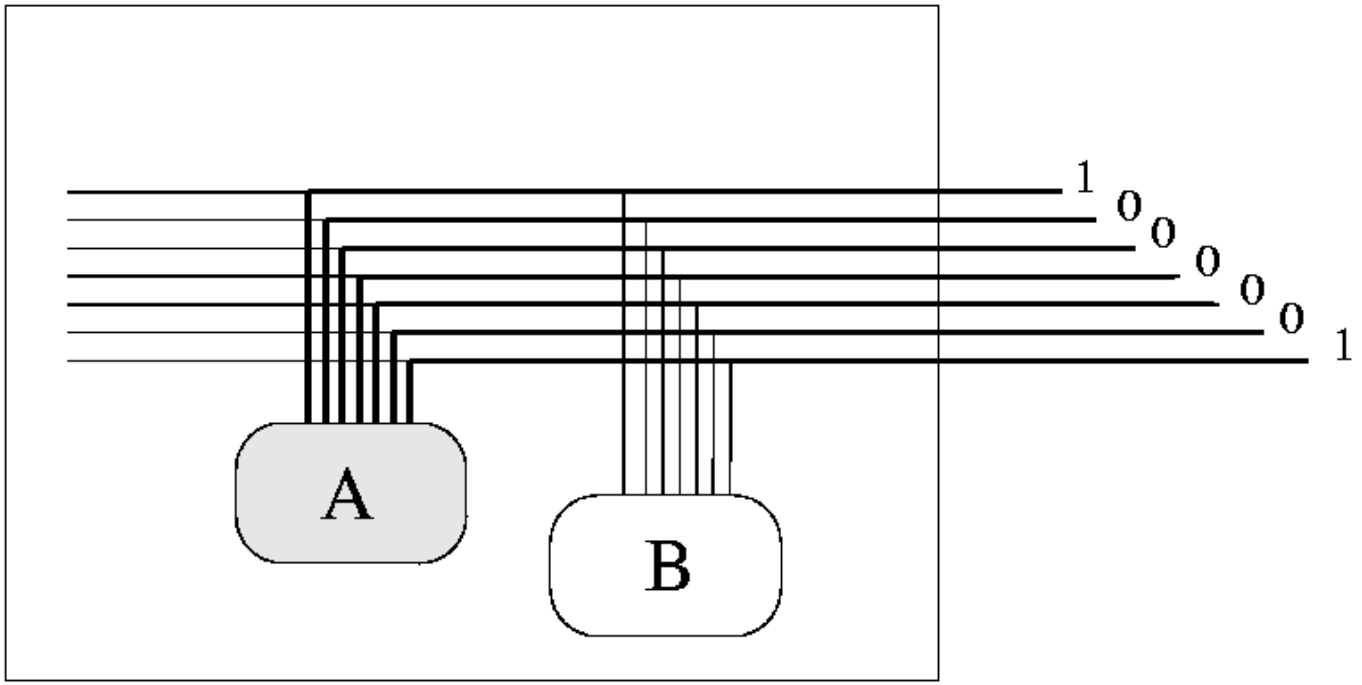
Source: www.asciitable.com

Tastatureingabe von Zeichen

Der Computer speichert Zeichen praktisch ausschließlich als ASCII–Bitmuster.

Frage: Wie kommt ein Zeichen bei der Eingabe zu seinem ASCII–Wert?

Antwort: Zum Beispiel bei Eingabe des Zeichens ’A’ per Tastatur durch Stromfluss in Leitung Nr. 7 und 1 (’A’ ≡ 65 ≡ 1000001).

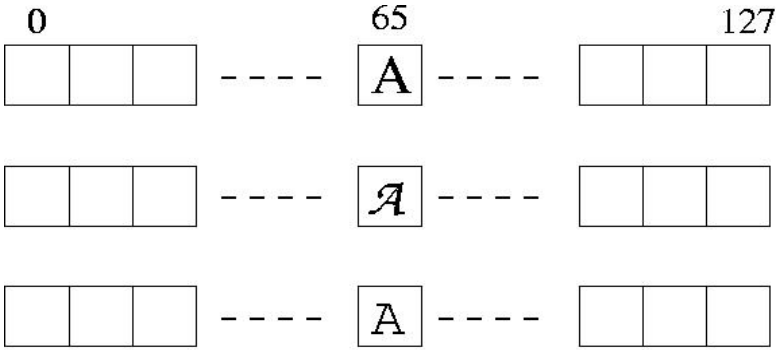


Tastatur

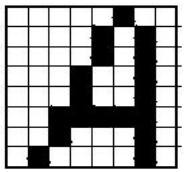
Bildschirmanzeige von Zeichen

Rückwandlung eines ASCII–Wertes in eine bildliche Darstellung des Zeichens (vereinfacht):

- Für diverse Schriftarten (*Fonts*) sind Tabellen gespeichert, in denen jedem ASCII–Wert ein Bild zugeordnet ist.



- Üblicherweise (aber nicht ausschließlich) sind die Bilder als matrixartig angeordnete Sequenz von Bits abgelegt (1=gehört zum Zeichen, 0=Hintergrund).



Beispiel: Zahlen

Beachte: Der ASCII–Wert einer Dezimalziffer ist nicht identisch mit ihrem dezimalen Zahlenwert (*Anm:* das gilt nur für die unteren 4 bits)

Beispiel:

- Die Zeichenfolge ”1234“ wird als ASCII–Folge (49,50,51,52) abgespeichert.

- Um die ASCII–Folge (49,50,51,52) in eine Dezimalzahl umzuwandeln, muss ein Umrechnungsalgorithmus auf die ASCII–Folge angewandt werden:

  - Ziehe von jeder ASCII–Nummer die ASCII–Nummer von 0 (also 48) ab.
    → (1; 2; 3; 4)
  - Multipliziere die vorletzte Ziffer mit 10^1, die drittletzte Ziffer mit 10^2 usw.
    → (1000; 200; 30; 4)
  - Addiere die vier Zahlen.

- Zur Ausgabe einer Dezimalzahl als Sequenz von ASCII–Zeichen im Dezimalsystem wird ein dazu inverser Umrechnungsalgorithmus angewandt.

ISO-Latin-1

- Die meisten Computer und Programme können Zeichen-kodierungen mit acht Bits verarbeiten.
- Es gibt verschiedene standardisierte Auswahlen von Zeichen für die zusätzlich verfügbaren 128 Code–Nummern.
- *Standard* in Mitteleuropa:
  eine standardisierte Auswahl namens *ISO–Latin–1*.
- Beispiele für weitere Zeichen in ISO–Latin–1:
  - ◊ ”Scharfes s“: ’ß’ ≡ 223.
  - ◊ Deutsche Umlaute (z.B. ’Ä’ ≡ 196).
  - ◊ Vokale mit Akzenten aus romanischen Sprachen (z.B. ’Ã’ ≡ 195).
  - ◊ ’$’ ≡ 163.

	000	001	002	003	004	005	006	007
0	NUL	DLE	SP	0	@	P	`	p
1	STX	DC1	!	1	A	Q	a	q
2	SOT	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

	008	009	00A	00B	00C	00D	00E	00F
0	XXX	DCS	NB SP	°	À	Đ	à	ð
1	XXX	PU1	¡	±	Á	Ñ	á	ñ
2	BPH	PU2	¢	2	Â	Ò	â	ò
3	NBH	STS	£	3	Ã	Ó	ã	ó
4	IND	CCH	¤	´	Ä	Ô	ä	ô
5	NEL	MW	¥	µ	Å	Õ	å	õ
6	SSA	SPA	¦	¶	Æ	Ö	æ	ö
7	ESA	EPA	§	·	Ç	×	ç	÷
8	HTS	SOS	¨	¸	È	Ø	è	ø
9	HTJ	XXX	©	1	É	Ù	é	ù
A	VTS	SCI	a	º	Ê	Ú	ê	ú
B	PLD	CSI	«	»	Ë	Û	ë	û
C	PLU	ST	¬	¼	Ì	Ü	ì	ü
D	RI	OSC	SHY	½	Í	Ý	í	ý
E	SS2	PM	®	¾	Î	Þ	î	þ
F	SS3	APC	¯	¿	Ï	ß	ï	ÿ

Source: http://www.chebucto.ns.ca/~af380/htmlchars2.html

ASCII-Darstellung von ISO-Latin-1

- Für jedes dieser Zeichen gibt es eine Umschreibung in reinen ASCII–Zeichen der Form "`&...;`".

- *Beispiele*:
  - 'ß' → "`&szlig;`" (s–z–Ligatur)
  - 'Ä' → "`&Auml;`" (A–Umlaut)
  - 'ä' → "`&auml;`" (a–Umlaut)
  - 'Ö' → "`&Ouml;`" (O–Umlaut)
  - 'ö' → "`&ouml;`" (o–Umlaut)
  - '&' → "`&`" (engl. Ampersand)

- *Konsequenz*:
  - Wenn eine solche Sequenz (z.B. "`&szlig;`") wörtlich dastehen soll: Schreib einfach hin "`&szlig;`".
  - Dank "`&`" kann also jeder beliebige Text in ISO–Latin–1 mit reinen ASCII–Zeichen umschrieben werden.

Beispiel: HTML

- HTML ist die ”Sprache“, in der WWW–Seiten geschrieben sind
- HTML–Seiten mit beliebigen ISO–Latin–1–Texten können in ASCII erstellt werden.
- Insbesondere kann der Inhalt von HTML–Seiten mit normaler Tastatur eingegeben werden.

  → Auch wenn die Tastatur keine Umlaute usw. hat.
- Die WWW–Browser (Netscape, Explorer...) interpretieren solche Umschreibungen, indem sie die entsprechenden Zeichen auf dem Bildschirm darstellen.

Unicode

Weiterentwicklung: Unicode–Standard mit 16 Bits, also 2^{16} = 65 536 möglichen Zeichen.

- Tatsächlich festgelegt in Unicode ist nur die Bedeutung von knapp 40 000 16–Bit–Werten.

- *Inhalte*:
  - Sonderzeichen aus diversen Sprachen (einschl. chinesisches Alphabet),
  - diverse technische Piktogramme,
  - diverse einfache geometrische Formen,
  - ...

- Unicode ist der Standardzeichensatz moderner Programmiersprachen wie Java.

Kompatibilität

- Bei Computerprogrammen, die nur ”reines“ 7–Bit–ASCII verarbeiten können, kann es zu textuellen Entstellungen bei der Verwendung von Umlauten u. Ä. Kommen.
- Zum Beispiel beim Verschicken von Email können Email–Verwaltungsprogramme auf dazwischenliegenden Internet–Knoten (*Routern*) dieses Manko immer noch haben.
- *Typisches Ergebnis*: Wenn auch nur ein einzelner solcher Router ”auf dem Weg“ liegt,
  - ◊ wird das führende Bit einfach auf 0 gesetzt und
  - ◊ der nächste Router, der mit einem 8–Bit–Zeichensatz arbeitet, behält diese Setzung für diese Bits bei.

  → Woher soll der Router auch wissen, ob das ursprünglich eine 0 oder 1 war?
- *Beispiel*: ’Ä’ ≡ 196 wird zu 196 - 128 = 68 ≡ ’D’.

Operationen auf Zeichen

- Test, ob ein ASCII–Wert x f ür einen Kleinbuchstaben steht:

  x steht für einen Kleinbuchstaben

  $\leftrightarrow$

  $x \geq 97$ und $x \leq 122$

- Umwandlung eines Kleinbuchstabens in einen Großbuchstaben:

  Falls x der ASCII–Wert eines Kleinbuchstaben ist, dann ist
  x + 'A' − 'a' = x + 65 − 97 = x − 32
  der ASCII–Wert des entsprechenden Großbuchstabens.

Konsequenz:
Solche Textmanipulationen lassen sich arithmetisch formulieren und daher mit Computern automatisch durchführen.

Einfacheres Rechnen mit ASCII

- Die ASCII–Werte sind nicht willkürlich zugeordnet, sondern so, dass bestimmte Operationen möglichst effizient sind.
- Insbesondere gilt das für Groß- und Kleinbuchstaben und Ziffern.
- Bisher haben wir schon ausgenutzt:

  Die Zuordnung unmittelbar aufeinanderfolgender ASCII–Werte jeweils für ’a’...’z’, ’A’...’Z’ bzw. ’0’...’9’.

Weiteres Beispiel

Die ASCII–Wert eines Großbuchstabens und seines zugehörigen Kleinbuchstabens unterscheiden sich nur im Bit Nr. 6:

→ Klein- und Großbuchstaben können einfach durch Überschreiben des Bits Nr. 6 ineinander umgewandelt werden.

'A'	65	1000001
'B'	66	1000010
...	...	...
'Z'	90	1011011
...	...	...
'a'	97	1100001
'b'	98	1100010
...	...	...
'z'	122	1111011

Texte

- Texte sind im Prinzip Sequenzen von Zeichen, die aufeinanderfolgend im Speicher des Rechners abgelegt werden.
- Im Speicher eines Rechners ist es aber notwendig, das Ende eines Textes irgendwie zu markieren.
- *Idee*:
  - Ein ASCII–Wert wird reserviert, der keinem Zeichen entspricht und der auch keine sonstige Funktion hat.
  - Dieser Wert wird hinter das Ende jedes Textes gesetzt, um anzuzeigen, dass der Text hier zu Ende ist.
- Reservierter Wert: 0.
- *Erinnerung*: Das ist nicht der ASCII– Wert des Zeichens ’0’.

Newline

- Der ASCII–Wert Nr. 10 ist für ”Newline“ reserviert.
- Das ist ein ASCII–Wert, der
  - nicht einem Zeichen entspricht,
  - sondern eine *Funktion* hat.
- *Konkrete Funktion*: Damit werden Zeilenumbrüche in Files angezeigt.
- Editoren und andere Programme zum Anzeigen von Files
  - geben solche Zeichen nicht auf Bildschirm oder Drucker aus (in welcher Form auch???),
  - sondern verarbeiten jedes solche Zeichen, indem sie mit der Anzeige des restlichen Textes auf der nächsten Zeile fortfahren.
- *Achtung*: Unterschiedliche Betriebssysteme haben ähnliche, aber nicht identische Konventionen für Zeilenumbruch!

HTML (Hypertext Markup Language)

- In dieser Sprache werden Seiten im WWW beschrieben.
- Idee: Beschreibung der Struktur von Texten.
- Herausforderung: Darstellung vieler Dinge, die sich in ASCII nicht direkt darstellen lassen.
- Lösung: HTML verwendet so genannte Tags, um die Struktur des Textes zu markieren. Verwendung:

```
<tag>Vom Tag betroffener Text</tag>
```

- Tags können ineinander geschachtelt werden, d.h. innerhalb des Anwendungsbereichs eines Tags können sich weitere Tags befinden.

Struktur eines HTML-Dokuments

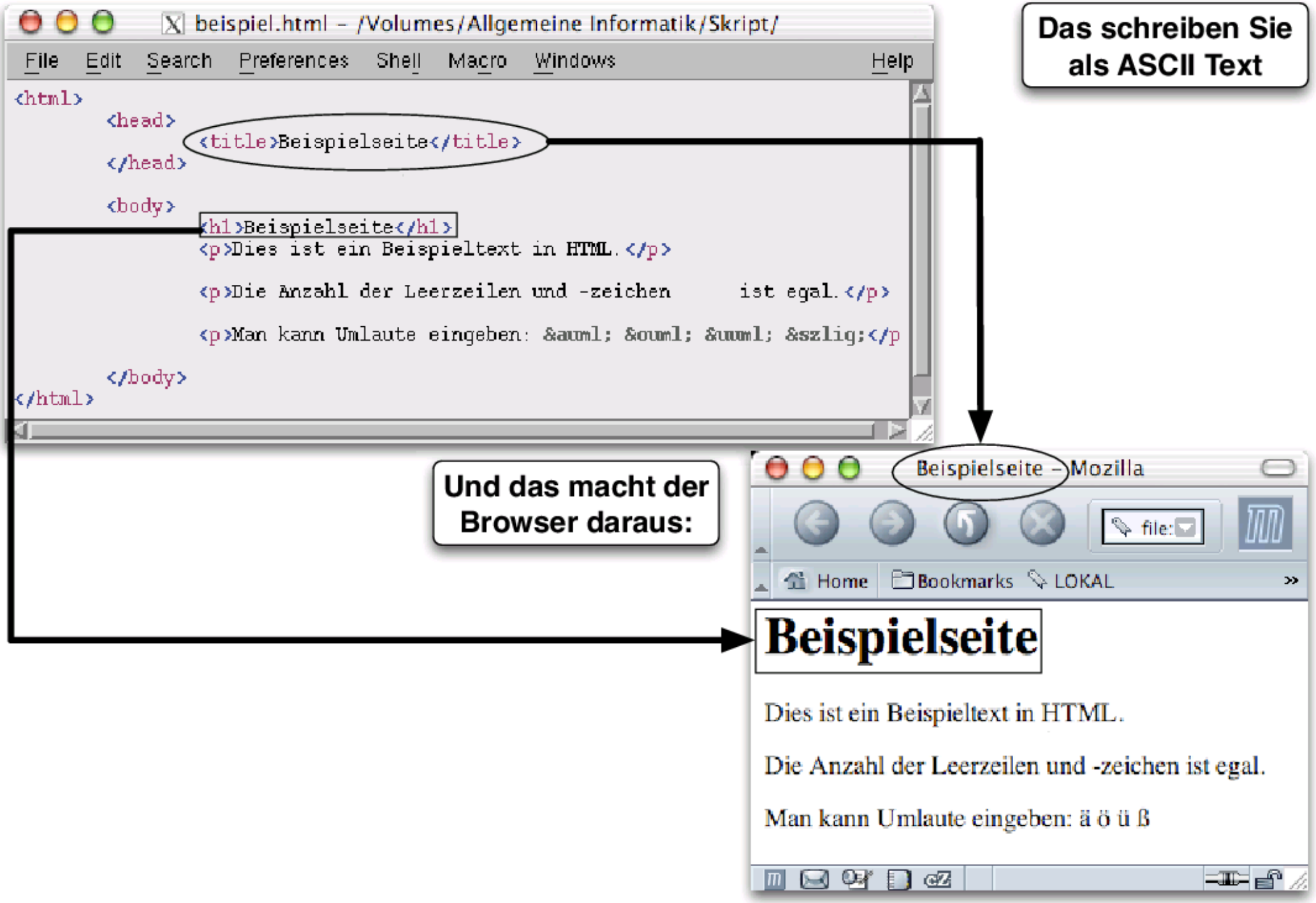
- HTML kennt ein oberstes Tag: `<html>`
  Dadurch wird ein HTML-Dokument gekennzeichnet.
- Direkt innerhalb des `<html>`-Tags befinden sich i.d.R. zwei weitere Tags:

  `<head>` Hier befinden sich Informationen über das Dokument.
  `<body>` umschließt den eigentlichen Inhalt des Dokumentes.

In `<body>` sind z.B. die folgenden Tags bekannt:

`<h1>` Dies bezeichnet eine Überschrift der ersten Ordnung.
Entsprechend gibt es `<h2>`, `<h3>`, etc.

`<p>` Der mit diesem tag umschlossene Text stellt einen Absatz dar.


beispiel.html – /Volumes/Allgemeine Informatik/Skript/
File Edit Search Preferences Shell Macro Windows Help
<html>
<head>
<title>Beispielseite</title>
</head>
<body>
<h1>Beispielseite</h1>
Dies ist ein Beispieltext in HTML.
Die Anzahl der Leerzeilen und -zeichen ist egal.
Man kann Umlaute eingeben: ä ö ü ß</p
</body>
</html>
Das schreiben Sie als ASCII Text
Und das macht der Browser daraus:
Beispielseite – Mozilla
file:
Home
Bookmarks
LOKAL
Beispielseite
Dies ist ein Beispieltext in HTML.
Die Anzahl der Leerzeilen und -zeichen ist egal.
Man kann Umlaute eingeben: ä ö ü ß

Wichtige HTML-Tags

- Formatierung
  - <b> Fettdruck
  - <i> Kursiv
  - <u> Unterstreichen
  -
 neue Zeile
  - <hr> Trennlinie
- Überschriften
  - <h1> Wichtigkeit 1
  - <h6> Wichtigkeit 6
- Listen
  - geordnete Liste (numeriert)
  - ungeordnete Liste (bullets)
  - Element der Liste
- Verlinkung
  - … Link auf eine andere Seite
  - <img src= „…“> Bild /Grafik einfügen
- Tabellen
  - <table> umschließt Tabelle
  - <tr> Tabellenzeile
  - <td> Tabellespalte
- Meta-Information (im head)
  - <title> Titel des Dokuments (wird als Titel des Browsers bzw. als Bookmark angezeigt)