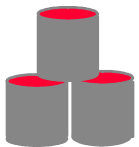
Grundlagen der Informatik
– Rechnergrundlagen –

Prof. Dr. Bernhard Schiefer

basierend auf Unterlagen von Prof. Dr. Duque-Antón
Abbildungen aus [Herold,Lurz,Wohlrab: GDI, 2012)]

bernhard.schiefer@fh-kl.de
http://www.fh-kl.de/~schiefer



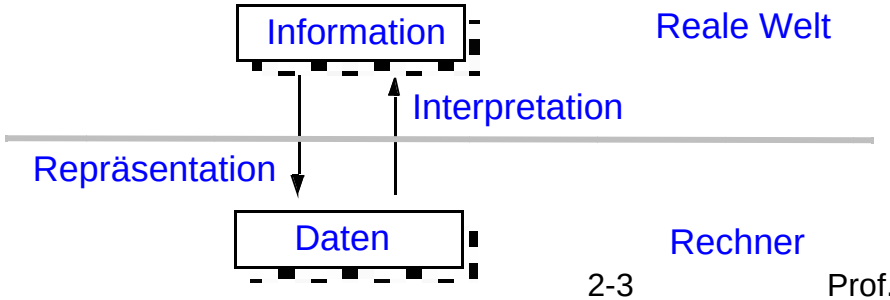
Inhalt

- Information und Daten
- Darstellung von Zahlen
- Darstellung von Texten und anderen Medien

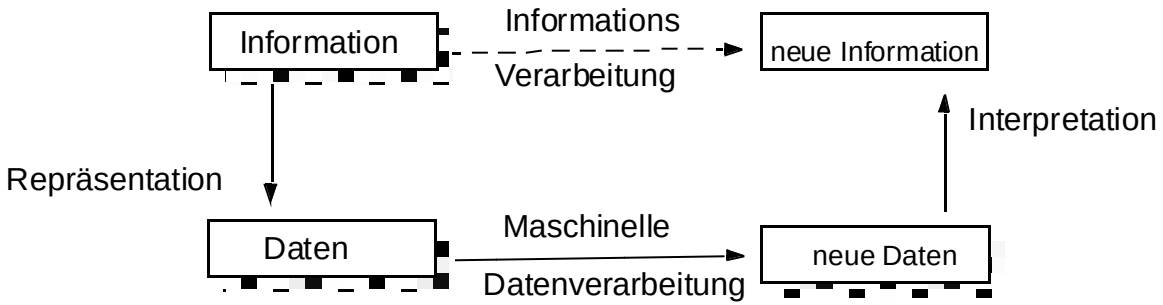
- Hardware
- Betriebssystem

Information und Daten

- Was tut eigentlich ein Computer?
  - ⇨ Umgangssprachlich soll er uns das Leben erleichtern, und zwar durch eine maschinelle, automatische Informationsverarbeitung!
  - ⇨ Konkret berechnen Computer Wettervorhersagen, steuern Raumfähren oder visualisieren medizinische Röntgenaufnahmen.
- Um seine Aufgabe zu erfüllen, muss die Eingangsinformation als Datum repräsentiert werden und die Ausgangsdaten wieder als Information interpretiert werden.
  - ⇨ Ein Computer kann auf Grund seiner Einfachheit nie Information im eigentlichen Sinne verarbeiten, sondern nur mit der in den Daten repräsentierten Information umgehen.



Maschinelle Verarbeitung



- Die Interpretation der Daten kann auch als Abstraktion aufgefasst werden.
- Die Interpretation der Daten ist immer abhängig vom Problem. Dasselbe digitale Codewort hat in einer anderen Umgebung (statt Wettervorhersage) eine ganz andere Bedeutung

Bits

- Ein Bit ist die kleinstmögliche Einheit der Information.
  - ⇨ Ein Bit ist die Informationsmenge in einer Antwort auf eine Frage, die nur zwei Möglichkeiten zulässt.
- Die Information in einem Bit kann also durch nur zwei Symbole in Daten codiert werden.
  - ⇨ Man benutzt dazu die Zeichen 0 und 1.
- Eine solche Codierung ist deswegen nötig, weil die Information technisch dargestellt werden muss. Man bedient sich dabei etwa
  - ⇨ elektrischer Ladungen (0 = ungeladen, 1 = geladen), oder
  - ⇨ elektrischer Spannungen (0 = 0 Volt, 1 = 5 Volt) oder
  - ⇨ Magnetisierungen (0 = unmagnetisiert, 1 = magnetisiert)
- Erst durch Interpretation werden Daten zu Informationen!

Bitfolgen

- Mit einem Bit können $2^1 = 2$ verschiedene Möglichkeiten dargestellt werden. Um mehr Informationen zu codieren, müssen Bitfolgen verwendet werden.
  - ⇨ Dazu werden in einem geordneten n-Tupel einfach n-viele Bits hintereinander gehängt.
  - ⇨ Mit n Bits können 2^n Werte dargestellt werden.
- Auf Maschinenebene werden Bitfolgen verwendet um Daten und Befehle zu codieren.
  - ⇨ Die Daten stellen natürliche oder reelle Zahlen dar oder auch Felder von solchen Zahlen.
  - ⇨ Die (Maschinen-) Befehle führen Operationen auf den Daten aus. Dabei werden beispielsweise zwei Zahlen addiert oder multipliziert und als neues Datum ausgegeben.

Bytes und Worte

- Innerhalb eines Rechners werden Bits in Gruppen verarbeitet, jeweils als Vielfaches von 8 Bit: also 8 Bit, 16 Bit, 32 Bit oder 64 Bit.
  - ⇨ Ein **Byte** ist eine Bitfolge der Länge 8.
  - ⇨ Zwei Bytes bilden ein **Wort**,
  - ⇨ 4 Bytes bilden ein **Doppelwort**.
- In der Informatik werden i. d. R. 2er-Potenzen für die entsprechenden Vielfache verwendet.
  - ⇨ Daher ergeben sich abweichende Potenzen gegenüber dem Dezimalsystem:
  - ⇨ 1 KiloByte = 1 KB = 2^{10} Byte = 1024 Bytes.
  - ⇨ 1 MegaByte = 1 MB = 2^{20} Byte = 1.048.576 Bytes.
  - ⇨ Diese Festlegungen entsprechen nicht dem üblichen dekadischen System und sind somit auch nicht streng einheitlich

Gebräuchliche Maßeinheiten für Bytes

Tabelle 3.3

Maßeinheiten für Bytes

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

Darstellung von Informationen

- In Anlehnung an das Grundprinzip, wird in diesem Kapitel für verschiedene Arten von Informationen eine geeignete Repräsentation als Datum gefunden. Das betrifft:
  - ⇨ Wahrheitswerte
  - ⇨ Natürliche Zahlen
  - ⇨ Rationale und reelle Zahlen
  - ⇨ Text und
  - ⇨ andere Medien wie Audio oder Video-Daten.

Wahrheitswerte

- Wahrheitswerte bzw. Kombinationen davon können als 0,1 Folgen dargestellt werden.
  - ⇨ Interpretation legt fest, ob z.B. 0 als Falsch und 1 als Wahr zu lesen ist
- Wahrheitswerte werden in der Mathematik auch als „boolsche Werte“ bezeichnet
  - ⇨ benannt nach George Boole (1815-1864)
- boolesche Algebra
  - ⇨ eine spezielle algebraische Struktur
    mit logischen Operatoren: AND, OR, XOR, NOT
  - ⇨ mathematische Notation: $\wedge$, $\vee$, $\underline{\vee}$, $\neg$

Zweielementige boolsche Algabra

- elementare Rechenregeln

$\wedge$	**0**	**1**
0	0	0
1	0	1

$\vee$	**0**	**1**
0		
1		

$\neg$	
0	1
1	0

$\veebar$	**0**	**1**
0		
1		

Gesetze

- Folgende Gesetze gelten in der boolschen Algebra:

⇨ Kommutativgesetze	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
⇨ Assoziativgesetze	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	$(a \vee b) \vee c = a \vee (b \vee c)$
⇨ Idempotenzgesetze	$a \wedge a = a$	$a \vee a = a$
⇨ Distributivgesetze	$a \wedge (b \wedge c) = (a \wedge b) \vee (a \wedge c)$ $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	
⇨ Neutralitätsgesetze	$a \wedge 1 = a$	$a \vee 0 = a$
⇨ Extremalgesetze	$a \wedge 0 = 0$	$a \vee 1 = 1$
⇨ Doppelnegationsgesetz	$\neg(\neg)\, a = a$	
⇨ DeMorgansche Gesetze	$\neg(a \wedge b) = \neg a \vee \neg b$	$\neg(a \vee b) = \neg a \wedge \neg b$
⇨ Komplementärgesetze	$a \wedge \neg a = 0$	$a \vee \neg a = 1$
⇨ Dualitätsgesetze	$\neg 0 = 1$	$\neg 1 = 0$
⇨ Absorptionsgesetze	$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$

Natürliche Zahlen

- $\mathbb{N}$ bezeichnet die Menge der natürlichen Zahlen ohne die Null, $\mathbb{N}_0$ einschließlich der Null.
- Mit n Bits können 2^n verschiedene Zahlen dargestellt werden.
- Typischerweise beginnt man in der Informatik mit der 0
  - ⇨ Die mit n-Bits darstellbaren natürlichen Zahlen decken damit den Bereich von 0 bis $2^n - 1$ ab.
- Wie sieht dieser Bereich bei einem Byte, einem Wort bzw. einem Doppelwort aus?
- Rechner arbeiten stets mit einer endlichen Genauigkeit, die sich aus der Anzahl der zur Darstellung verwendeten Bits ergibt

Weitere Zahlensysteme

- Dualzahlen können gut vom Rechner dargestellt werden, aber schlecht von einem Menschen interpretiert werden.
- Aus diesem Grund werden „Zwischensysteme“ eingebaut:
  - ⇨ Das Oktalsysteme verwendet ein Alphabet mit 8 Zeichen {0, 1, 2, ... 7} .
  - ⇨ Hexalziffern erfordern 16 Zeichen {0, 1, ... 9, A, B, C, D, E, F}. Hexalziffern werden auch Halb-Bytes genannt.
  - ⇨ Die Anordnung der Bits in 4er Gruppen führt zu Hexadezimalziffern. Jeder Vierergruppe kann dann ein Zeichen aus dem Alphabet zugewiesen werden. Wie sieht diese binäre Tabelle aus?
- Aufgabe: Wandle 200_8 bzw. $B2A_{16}$ in eine duale Darstellung um!
  - ⇨ Wie kann man das systematisieren?
- Aufgabe: Wandle 200_{10} in eine hexadezimale Darstellung um!
  - ⇨ Wie kann man das systematisieren?

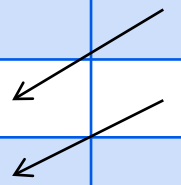
Umwandlung des Bezugssystems

- Mit Hilfe der Divisionsmethode kann die Umwandlung von Dezimalzahlen in Hexadezimalzahlen, Oktalzahlen oder auch Dualzahlen systematisch ermittelt werden.
  - ⇨ Dazu werden zwei spezielle Operationen benötigt:
    - die ganzzahlige Division (div b) und
    - der Divisionsrest (mod b)
  - ⇨ wobei b die Basis des Ziel-Zahlensystems darstellt.
- Für b = 2 gilt beispielweise:
  - ⇨ 101 div 2 = 50 und
  - ⇨ 101 mod 2 = 1

Die Divisionsmethode

- Den Algorithmus kann man sich gut mit Hilfe einer Tabelle veranschaulichen:

Zahl	div 2	mod 2	
6	3	0	2^0
3	1	1	2^1
1	0	1	2^2



- ⇨ Die Ausgangszahl z (oder genauer z_{10}) wird in einer Zeile mit dem Ergebnis der div-Operation und der mod-Operation geschrieben.
- ⇨ In der darauf folgenden Zeile wird das Ergebnis der div-Operation ganz links geschrieben und das ganze Wiederholt.
- ⇨ Das Verfahren endet, wenn die div-Operation 0 ergibt.
- ⇨ Das Ergebnis wird durch die rechte Spalte in umgekehrter Reihenfolge dargestellt.

Die Substraktionsmethode

Darstellung ganzer Zahlen

- $\mathbb{Z}$ ist die Menge der Natürlichen Zahlen $\mathbb{N}$ vereinigt mit der Menge der negativen Natürlichen Zahlen und der Menge, die nur die Null enthält:
  - ⇨ $\mathbb{Z}$ = {0, +1, -1, +2, -2, . . .}
- Darstellung der positiven Zahlen sollte klar sein
- Wie kann nun das Vorzeichen im Rechner dargestellt werden?

Darstellung negativer Zahlen

- Denkbar: Wie im Dezimalsystem erste Stelle für das Vorzeichen nutzen
  - Im führende Bit (ganz links) das Vorzeichen codieren
    - ⇨ z.B. eine 0 zur Codierung einer positiven und eine 1 für negative Zahl.
- Beispiele:
  - ⇨ +3 = 0011 und -3 = 1011 (bei Betrachtung von Halb-Bytes)
- Im Rechner wird immer auf Wortgrenzen aufgefüllt und rechtsbündig abgespeichert. Bei der Nutzung von 2-Byte-Zahlen bedeutet das:
  - ⇨ $-3 = 10000000\ 00000011_2$
- Mit n Bit ist die Menge $\{-2^{n-1}, \ldots, +2^{n-1}\}$ darstellbar, also $2^n - 1$ ($= 2 * 2^{n-1} - 1$) Werte. Dabei treten einige Probleme auf:
  - ⇨ Es gibt zwei Darstellungen für die Null: 0000 = 0 = 1000 = - 0.
  - ⇨ Unsere gewohnte Arithmetik gilt nicht mehr!
    Bestimmen Sie für eine beliebige Zahl das Ergebnis von z + (-z)

Das Einerkomplement

- Bei der Einerkomplementdarstellung werden negative Zahlen als invertierte Form der positiven Zahl dargestellt.
  - ⇨ auch hier müssen negative Zahlen am ersten Bit erkannt werden können
  - ⇨ Schreibweise für Komplementdarstellung der Zahl z: ~z
- Beispiel:
  - ⇨ +3 = 0011 und -3 = 1100 (bei Betrachtung von Halb-Bytes)
- Bzw. bei der Nutzung von 2-Byte-Zahlen dann:
  - ⇨ -3 = 1111111 1111100_2
- Auch hiermit werden noch spezielle Additions- bzw. Substraktionsmethoden benötigt.

Darstellung im Zweierkomplement

- Mit Hilfe der Zweierkomplement-Darstellung können die erwähnten Probleme alle behoben werden.
  - ⇨ Subtraktion wird auf Addition mit entsprechenden Zweierkomplement zurückgeführt.
- Zur **Darstellung** einer beliebigen **ganzen Zahl z** unterscheiden:
  - ⇨ Falls **z positiv** ist, wird z direkt als Dualzahl dargestellt und rechtsbündig abspeichert und falls nötig mit Nullen aufgefüllt.
  - ⇨ Falls **z negativ** ist, dann wird zunächst –z in eine Dualzahl gewandelt und davon das Einerkomplement gebildet: **~z** (0 und 1 vertauscht) . Anschließend wird eine 1 dazu addiert.
- Beispiele:

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

Eigenschaften Zweierkomplement

- Das führende Bit gibt das Vorzeichen an (wie bei Einerkomplement)
- Mit n Bit kann man 2^n Zahlen darstellen, und zwar im Wertebereich von -2^{n-1} bis $+2^{n-1} -1$
- Es gibt nur eine Darstellung für die 0
- Es wird keine spezielle Subtraktionsmethode benötigt. Die Subtraktion kann effizient auf die Addition zurückgeführt werden
- Beispiel:
  - ⇨ $7_{10} - 3_{10} = 7_{10} + (-3_{10}) =$
  - ⇨ 0111 +
1101
=====
0100 = 4_{10}

Rückwandlung Zweierkomplement

- Interpretation einer binären Zahl in Zweierkomplement-Darstellung
  - ⇨ Dazu betrachten wir eine binäre Zahl mit n Bit in allgemeiner Form: $a_{n-1}\ a_{n-2}\ \ldots\ a_0$
- Falls die Zahl positiv ist, dann muss $a_{n-1} = 0$ gelten.
  - ⇨ Die letzten n-1 Bits können direkt interpretiert werden.
  - ⇨ Wert $z_{10} = \sum_{i=0}^{n-2} a_i * 2^i$
- Falls die Zahl negativ ist, dann muss $a_{n-1} = 1$ gelten
  - ⇨ Die letzten n-1 Bits müssen in der Zweierkomplement-Darstellung interpretiert werden und der resultierende Wert erhält ein negatives Vorzeichen.
  - ⇨ Wert $z_{10} = -\left(\sum_{i=0}^{n-2}(1 - a_i) * 2^i + 1\right)$

Ganze Zahlen in C/C++ und Java

- Typische Werte bei 32-Bit Architekturen

Tabelle 3.4

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	$1.2 \cdot 10^{-38} \ldots 3.4 \cdot 10^{38}$
`double`	64	$2.2 \cdot 10^{-308} \ldots 1.8 \cdot 10^{308}$
`long double`	96	$3.4 \cdot 10^{-4\,932} \ldots 1.1 \cdot 10^{4\,932}$

Rationale und Reelle Zahlen

- Rationale Zahlen können als Bruch von zwei ganzen Zahlen dargestellt werden
- Rationale Zahlen werden in einem Rechner wie reelle Zahlen dargestellt, obwohl diese im Grunde genau wie die natürlichen oder ganzen Zahlen abzählbar sind (reellen Zahlensind überabzählbar)
  - ⇨ Reelle Zahlen können in einem Rechner daher nur näherungsweise angegeben werden.
- Gesucht wird eine Darstellung, mit der die Genauigkeit den gestellten Anforderungen angepaßt werden kann.
  - ⇨ Also möglichst ein großes Intervall der reellen Zahlen umfassen und
  - ⇨ die Genauigkeit bei kleinen Zahlen sehr hoch, bei großen Zahlen eher niedriger halten.

Darstellung von Fließkommazahlen

- Darstellung gebrochener Zahlen als Dualbrüche
- Die Gleitpunktdarstellung (floating point) :
  - ⇨ Mantisse M (Ziffernfolge) wird normiert dargestellt als 1.M
    so muss die 1 vor dem Punkt nicht mehr gespeichert werden (spart Platz). Die Mantisse besteht aus Binärziffern m_1 bis m_n.
  - ⇨ Exponent E zur Basis b (hier wird meist b=2 als Basiszahl verwendet)
  - ⇨ Vorzeichen V

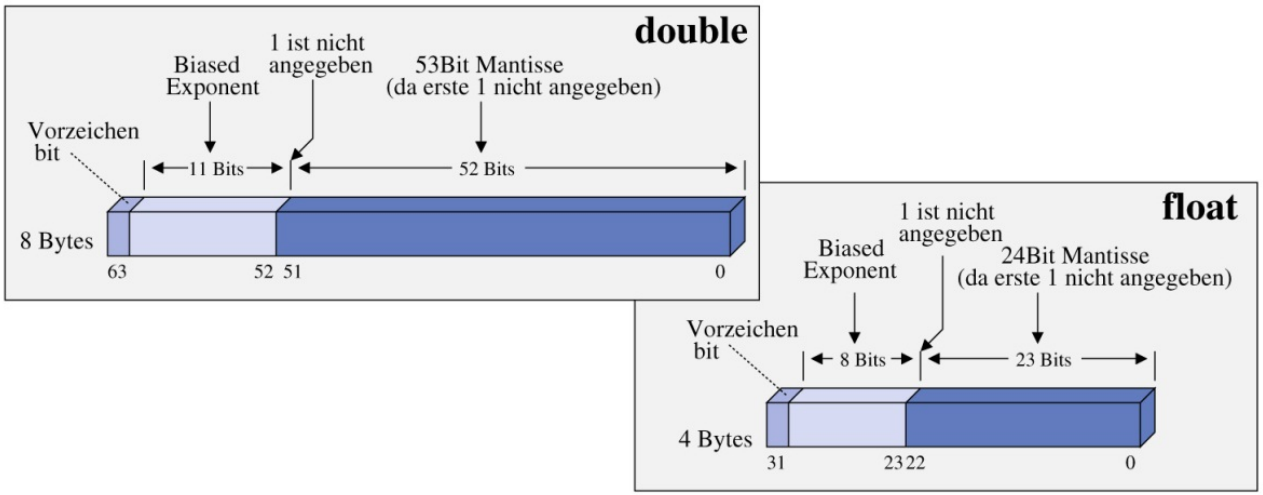


Abbildung 3.4: IEEE-Format für `float` und `double`

Beispiele

- Lichtgeschwindigkeit in der Gleitkommaschreibweise für Dezimalsysteme (Basis b=10):
  - ⇨ $c = 3 * 10^8 = 3.0E+08$
- Anzahl der Einwohner in Deutschland:
  - ⇨ 83 000 000 = 8.3E+07
- Konstante in einem technischen Ablauf:
  - ⇨ 0.000 123 456 = 0.123456E-03

Angabe eines Bias

- Der Exponent kann negativ oder positiv sein
  - ⇨ Negative Exponenten würden wieder eine Wandlung ins 2er Komplement erfordern
- Zur Vereinfachung werden häufig nur positive Exponenten gespeichert
  - ⇨ Es wird ein konstanter Wert (z.B. 127 bei 8 Bit Exponenten) immer zum Exponent addiert
  - ⇨ Man spricht dann von einem *Biased Exponent*
  - ⇨ Das Ergebnis wird dann als vorzeichenlose 8 Bit Zahl bzw. 11 Bit Zahl abgespeichert.

Duale Fließkommazahl umwandeln

- Aufgabe:
  - ⇨ Gegeben sei die folgende Dualzahl
    - .1011E+10
  - ⇨ Welcher Zahle im Dezimalsystem entspricht das ?
    (Ohne Berücksichtigung eines Bias.)

Vorsicht: Rundungsproblematik

- Für Gleitpunktzahlen, die im Dezimalsystem exakt dargestellt werden können, gilt dies nicht mehr bei Wandlung der Basis ins Dualsystem!
- Eine Überprüfung auf Gleichheit bei *float* und *double* Werten führt sehr leicht zu Fehlern
  - ⇨
```
double   d = 0.7 + 0.2;
 if (d  ==  0.9)
    System.out.println(„Alles ok!");
else
    System.out.println("Wie kommt das?");
```

Darstellung von Texten

- Ein Text wird typischerweise zeichenweise dargestellt (codiert).
- Es gibt viele verschiedene binäre Codierungen für Zeichen
  - ⇨ Es gibt auch sehr viele verschiedene Zeichen in Texten
  - ⇨ Wie viele kennen Sie?
- Verbreitete Codierungen
  - ⇨ ASCII

ASCII Codierung

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- Bereits 1963 in den USA zum Datenaustausch zwischen Computern bzw. für den Fernschreibverkehr genormt
- Für jedes Zeichen wird 1 Byte verwendet
  - ⇨ In jedem Zeichen war ursprünglich 1 Bit als Prüf- Bit verwendet
  - ⇨ Es stehen so nur 7 Bit zur Verfügung.
- Damit werden 128 Zeichen codiert.
  - ⇨ Das reicht für vieles aus, enthält aber z.B. keine Umlaute
- Da aktuelle Datenübertragungen sicherer sind, verwenden ASCII-Erweiterungen alle 8 Bit zur Darstellung von Zeichen.

ASCII Codierung (2)

- Systematiken:
  - ⇨ Die Kleinbuchstaben sind in alphabetischer Reihenfolge durchnumeriert.
  - ⇨ Die Großbuchstaben sind in alphabetischer Reihenfolge durchnumeriert.
  - ⇨ Die Ziffern 0 bis 9 stehen in der natürlichen Reihenfolge.
  - ⇨ Es können 33 druckbare Sonderzeichen wie beispielsweise @ dargestellt werden.
  - ⇨ Es können 33 nicht druckbare Sonderzeichen wie *Carriage Return* repräsentiert werden.
- Es gibt zahlreiche Erweiterung, um die vollen 8-Bit auszunutzen. Beispielsweise wird in häufig Westeuropa Latin-1/Latin-9 verwendet.
- ASCII-basierte Codierungen stellen immer noch die am meisten verbreitete Repräsentation von Textinformation dar.

ASCII Codierung (3)

Hex	00	10	20	30	40	50	60	70
0	*nul*	*dle*		0	@	P	‘	p
1	*soh*	*dc1*	!	1	A	Q	a	q
2	*stx*	*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*

- Beispiele
  - ⇨ **A** = 41_{16} = 65_{10} = $0100\ 0001_2$
  - ⇨ **0** = 30_{16} = 48_{10} = $0011\ 0000_2$

ISO 8859

- Der ISO 8859 Standard umfasst 15 Codierungen, die die Darstellung verschiedener nationaler Sonderzeichen erlauben
- ISO8859-15 enthält zusätzlich zu ISO8859-1 noch das Euro Zeichen €
- Die deutschen Sonderzeichen (Ä = $C4_{hex}$, ä = $E4_{hex}$, Ö = $D6_{hex}$, ö = $F6_{hex}$, Ü = DC_{hex}, ü = FC_{hex} und ß = DF_{hex}) sind in allen Latin-*x*-Zeichensätzen (also *nicht* in den Teilnormen -5, -6, -7, -8 und -11) unter jeweils demselben Code vorhanden.

-1	*Latin-1*, Westeuropäisch
-2	*Latin-2*, Mitteleuropäisch
-3	*Latin-3*, Südeuropäisch
-4	*Latin-4*, Nordeuropäisch
-5	Kyrillisch
-6	Arabisch
-7	Griechisch
-8	Hebräisch
-9	*Latin-5*, Türkisch
-10	*Latin-6*, Nordisch
-11	Thai
-13	*Latin-7*, Baltisch
-14	*Latin-8*, Keltisch
-15	*Latin-9*, Westeuropäisch
-16	*Latin-10*, Südosteuropäisch

Unicode Codierungen

- Es existieren verschiedene Varianten von Codierungen, die mehr als 1 Byte verwenden, um auch mehrsprachige Texte korrekt abbilden zu können.
  - ⇨ Die wichtigsten sind: UTF-8, UTF-16, UTF-32
- Die Unicode-Codierung UTF-16 verwendet 2 Byte zur Codierung und kann somit 65.536 Zeichen darstellen.
  - ⇨ Damit können viele relevante internationale Zeichen (auch viele asiatische Sprachen) dargestellt werden.
  - ⇨ Java unterstützt direkt UTF-16.
- UTF-8 ist eine variable lange Codierung
  - ⇨ Die ersten 128 Zeichen sind mit dem ASCII-Code identisch!
  - ⇨ Seltene Zeichen benötigen bis zu 4 Byte

Codierung von Grafik

- Es gibt zahlreiche Möglichkeiten, Grafiken (effizient) darzustellen.
  - ⇨ Die einfachste Variante verwendet eine Pixelgrafik (Pixel = Bildpunkt).
  - ⇨ Im einfachsten Fall (Schwarzweißbild) wird eine Bit für die Information verwendet, ob der Bildpunkt weiß oder schwarz ist.
  - ⇨ Zur Codierung von Farbgrafiken, bieten sich das RGB-Verfahren an, welche die Farbe eines Bildpunktes in die drei Spektralfarben Rot, Grün und Blau zerlegt. Dabei werden die einzelnen Helligkeitswerte jeweils mit einer Genauigkeit von 8 Bit dargestellt (True Color: 3 * 8 Bit = 24 Bit = 16.777.217 Farben).
- Um die Übertragungskapazität bzw. das Speichermedium zu schonen, werden typischerweise zusätzlich Komprimierungsverfahren verwendet.
  - ⇨ Dabei muss man zwischen verlustfreien und verlustbehafteten unterscheiden
- Als Beispiele seien hier JPEG oder MPEG genannt.

Codierung von Audiosignalen

- Bei der Darstellung von Audio-Information spielt das Abtast-Theorem eine zentrale Rolle.
  - ⇨ Vereinfachend werden analoge Schallwellen durch periodisches Abtasten digital codiert.
- Beispiele:
  - ⇨ Die ISDN-Telefonie tastet mit 8 kHz ab
    und stellt jeden Abtastwert mit 8 Bit dar.
    Auf diese Weise ergibt sich ein digitaler Telefonkanal von 64 kBit/sec.
  - ⇨ Eine CD kommt auf 795 kBit/sec pro Stereokanal, da hier mit 44,1 kHz abgetastet wird und 16 Bit pro Abtastwert verwendet werden.
- Auch hier spielen Kompressionsverfahren eine wichtige Rolle.
  - ⇨ Beispiele: MP3, MPEG-2, …