Informationssysteme

Inhalt

[Algorithmusbegriff (Kap. 2, p. 31f) 3](#_Toc408823136)

[Allgemein 3](#_Toc408823137)

[Von-Neumann- vs. Harvard-Architektur (Kap. 2, p. 36ff) 4](#_Toc408823138)

[Zahlendarstellung in ALU und FPU (Kap. 3, p. 48ff) 6](#_Toc408823139)

[Zahlensysteme 6](#_Toc408823140)

[Alu: 7](#_Toc408823141)

[Rechnungen: 7](#_Toc408823142)

[Wertebereiche: 8](#_Toc408823143)

[Gleitpunktzahlen 8](#_Toc408823144)

[Tabellen- und Zeichencodes (Kap. 3, p. 71ff) 10](#_Toc408823145)

[ACII 10](#_Toc408823146)

[Unicode 10](#_Toc408823147)

[UTF-Unicode Transformation Format 11](#_Toc408823148)

[Big und Little Endian 12](#_Toc408823149)

[Exkurs: Codegewinnung 12](#_Toc408823150)

[Diskretisierung 12](#_Toc408823151)

[Modulierung 12](#_Toc408823152)

[Harmonische Schwingung 13](#_Toc408823153)

[Fehlertolerante Codierung (Kap. 20, p. 734ff) 14](#_Toc408823154)

[Code-Belegung 14](#_Toc408823155)

[Fehlererkennung/Korrektur 14](#_Toc408823157)

[Hammingcode 15](#_Toc408823158)

[Paritybit 15](#_Toc408823159)

[Parity Bit Positions: 16](#_Toc408823160)

[Leave Space for PB : 16](#_Toc408823161)

[14.2 Fixing 16](#_Toc408823162)

[CRC(cyclic redundancy check)-Codierung 16](#_Toc408823163)

[Holzi 17](#_Toc408823164)

[17](#_Toc408823165)

[Boolesche Algebra (Kap. 4, p. 82ff) 17](#_Toc408823166)

[Transistor 18](#_Toc408823167)

[Hardware und Firmware (Kap 5, p. 92ff) 19](#_Toc408823168)

[Speichergrössen 19](#_Toc408823169)

[Speicher-Hierarchie 20](#_Toc408823170)

[Disks 21](#_Toc408823171)

[Miniaturisierung: 21](#_Toc408823172)

[RAID (Redundant Array of Inexpensive/Independet Disks): 22](#_Toc408823173)

[Programmentwicklung (Kap. 6, p. 136ff) 23](#_Toc408823174)

[Prozessmodelle: 23](#_Toc408823175)

[Schwergewichtige (klassische) 23](#_Toc408823176)

[Leichtgewichtige (Agile) 23](#_Toc408823177)

[Compilieren und Interpretieren 24](#_Toc408823178)

[Programmiersprachen, Algorithmen (Kap.7/8, p. 148ff) 24](#_Toc408823179)

[Prozedurale Programmierung 24](#_Toc408823180)

[Funktionale Programmierung 24](#_Toc408823181)

[Datenstrukturen (Kap. 7, p. 208ff, Kap. 8, p. 324ff) 26](#_Toc408823182)

[Sorts: 26](#_Toc408823183)

[Stack 28](#_Toc408823184)

[Bäume 28](#_Toc408823185)

[Binäre Suchbäume 29](#_Toc408823186)

# Algorithmusbegriff (Kap. 2, p. 31f)

## Allgemein

|  |  |
| --- | --- |
| Ein Algorithmus ist eine Verarbeitungsvorschrift, die von einem Mensch oder einer Maschine ausgeführt wird | Bsp. Euklidscher Algorithmus |

#### Unterschied Effektive vs. Effizienz

Effektive = Ziel erreichen, wie, egal --> Irgendwann kommst du zum Ziel

Effizienz = Ziel erreichen, in vorgegebener Zeit --> Möglichst wenig Aufwand, wenig Zeit, wenig Geld

Erreichen

Beschreiben eine Verarbeitungsvorschrift, dies kann sein:

Vorschrift zur Lösung einer Aufgabe

Vorschrift zur Lösung eines Problems

|  |  |
| --- | --- |
| **Begriff** | **Erklärung** |
| terminiert | Algorithmus ist nach n-Schritten beendet, hat er kein bestimmtes Ende,  spricht man einem nicht-terminiertem Algorithmus |
| Determinismus | Für alle Eingaben ist der Ablauf des Algorithmus eindeutig bestimmt,  anderenfalls heisst er nicht-deterministisch |
| Determiniertheit | Ein Algorithmus heißt determiniert, wenn er bei gleichen  zulässigen Eingabewerten stets das gleiche Ergebnis liefert. Andernfalls  heißt er nicht-determiniert. |

**Wichtige Aussagen:**

Ein deterministischer Algorithmus ist immer determiniert,

d. h. er liefert bei gleicher Eingabe immer die gleiche Ausgabe.

Die "Umkehrung" aber gilt nicht: So gibt es Algorithmen, die nicht-deterministisch,

aber trotzdem determiniert sind (d. h. das gleiche Ergebnis liefern).

#### Der Algorithmus

**endlich:** nach einer endlichen Zeit wird der Algorithmus beendet.

**deterministisch:** nur definierte und reproduzierbare Zustände treten auf d.h. bei gleicher Eingabe

folgt immer gleiche Ausgabe und zusätzlich wird die gleiche Folge von Zuständen durchlaufen. Zu

jedem Zeitpunkt ist der nachfolgende Abarbeitungsschritt des Algorithmus eindeutig festgelegt. Es

gibt auch "nicht-deterministische" Algorithmen!

**effektiv:** Grad (Mass) für die Zielerreichung: Es gibt Aufschluss darüber, wie nahe ein erzieltes

Ergebnis dem angestrebten Ergebnis gekommen ist. Wir erwarten bem Programmieren implizit meist

100% ige Zielerreichung!

**effizient:** Mass für die Wirtschaftleichkeit z.B. geringer Verbrauch an Ressourcen wie Speicherplatz

und Rechenzeit.

**Abstrahierung:** Ein Algorithmus löst eine ganze Klasse von gleichartigen Problemen. Die Wahl des

speziellen Problems erfolgt über Parameter.

**Finitheit statisch:** Die Beschreibung des Algorithmus selbst ist endlich. - Finitheit dynamisch: Ein in

Bearbeitung befindlicher Algorithmus hat zu jedem Zeitpunkt eine endliche Fülle von

Datenstrukturen und Zwischenergebnissen. Er belegt deshalb endlich viele Ressourcen im System.

**Sequenzialität:** Ein Algorithmus ist aus Einzelschritten aufgebaut. In jedem dieser Schritte wird eine

einfache Operation ausgeführt, wie z.B. eine Addition oder eine Zuweisung zu einer Variablen.

**Realisierbarkeit:** Die genannten Operationen müssen tatsächlich in der Praxis durchführbar sein. Die

Quadratur des Kreises oder die Division durch Null sind also nicht algorithmisch lösbar, ebenso wenig

wie die Bestimmung der Masse der Erde auf ein Gramm genau.

**Terminierung:** Normalerweise gehen wir davon aus, dass ein Algorithmus terminiert, das heisst, nach

einer absehbaren Zeit kontrolliert abbricht. Gewisse Algorithmen – und Programme – laufen

potenziell endlos wie z.B. Betriebssysteme, Prozessleitsysteme usw.

**Determinismus:** Ein Algorithmus ist dann deterministisch, wenn zu jedem Zeitpunkt nur eine

Möglichkeit des weiteren Ablaufs, oder des Abbruchs, besteht. Ist ein nichtdeterministischer

Algorithmus durch Wahrscheinlichkeiten oder Zufälle gesteuert, dann heisst er stochastisch.

**Determiniertheit:** Ein Algorithmus ist dann determiniert, wenn er bei gleichen Startparametern und

Eingabewerten auf gleiche Art terminiert und gleiche Ergebnisse liefert (dabei aber möglicherweise

nicht immer die gleiche Sequenz von Einzelschritten wählt).

## Von-Neumann- vs. Harvard-Architektur (Kap. 2, p. 36ff)

|  |  |
| --- | --- |
| **Von-Neumann Architektur:**   * Rechenwerk(ALU, Arithmetic Logical Unit, Ganzzahlrechner) * Leitwerk (CPU, Central Processing Unit) * Speicherwerk (Memory) * Ein-Ausgabewerk   **Architektonische Merkmale der Von-Neumann-Maschine sind:**   * Es handelt sich um einen SISD-Rechner. * Es gibt ein gemeinsames Leitungssystem für Befehle und Nutzdaten (Operanden, Ergebnisse), * den Bus. * Im Speicherwerk sind Befehle und Nutzdaten abgelegt, wobei keine vorgegebene Schichtung * existiert. * Die Befehle und Nutzdaten sind direkt adressier-, lösch- und änderbar. * Die Verarbeitungsbefehle sind in einem Speicherwerk als 1-dimensionaler Adressenraum * abgelegt. * Befehle werden sequenziell abgearbeitet. * Ein Befehlszeiger, der zu jedem Zeitpunkt auf den nächsten zu ladenden und bearbeitenden * Befehl zeigt, überwacht die Befehlsbearbeitung (oft: Befehlszähler). * Bedingte und unbedingte Sprünge erlauben die Abweichung von der Sequenzialität. * Sämtliche Datenflüsse müssen durch das Leitwerk koordiniert werden. Dies verlangsamt den * Fluss grosser Datenmengen zwischen Ein- oder Ausgabegeräten sowie dem Speicherwerk * (Von-Neumann-Flaschenhals und Direct Memory Access, DMA). * Bei steigender Leistungsfähigkeit des Prozessors bildet das Bussystem einen Engpass (VonNeumann-Flaschenhals * und Befehls- bzw. Daten-Caching). | Komponenten Von-Neumann Architektur |

|  |  |
| --- | --- |
| **Harvard Architektur:**   * Getrennte Speicherwerke für Befehle und Nutzdaten mit je eigenen 1-dimensionalen Adressenräumen. * Getrennte Leitungssysteme für Befehle und Nutzdaten. * Dadurch gleichzeitiges Laden eines Befehls und eines Operanden. | Komponenten Harvard Architektur |
| **Von-Neumann mit Cache (Mischform):**  Entspricht der Aktuellen Prozessorarchitektur. Verwendet ebenfalls getrennten Daten und Befehlsspeicher, Cached die Daten aber nur. Unten wird alles über den Universellen Speicherbus geleitet |  |

# Zahlendarstellung in ALU und FPU (Kap. 3, p. 48ff)

## Zahlensysteme

|  |  |
| --- | --- |
| **38 vom 10ner ins 5er System:** | **38 vom 5ner ins 10er System:** |
| 38 : 5 = 7 Rest 3  7 : 5 = 1 Rest 2  1 : 5= 0 Rest 1  Resultat = 123 |  |
| **45.0625 vom 10ner ins 6er System:** | **45.0625 vom 6er ins 10er System:** |
| **Vorkomma:**  45 : 6 = 7 Rest: 3  7 : 6 = 1 Rest: 1  1 : 6 = 0 Rest: 1  Resultat: 113  **Nachkomma:**  6 · 0,125 = 0,75 --> Ziffer: 0  6 · 0,75 = 4,5 --> Ziffer: 4  6 · 0,5 = 3 --> Ziffer: 3  Resultat: 0,043  **Gesamt Resultat: 113,043** | **Vorkomma:**  **Nachkomma:**  **Gesamt Resultat= 45.125** |

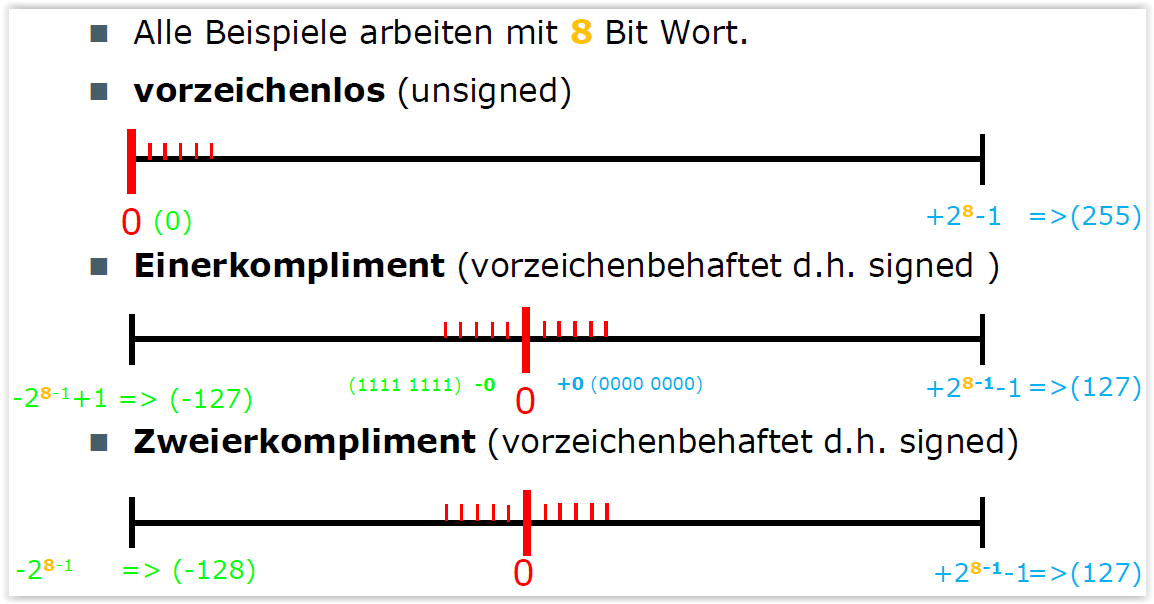
## Alu:

|  |  |
| --- | --- |
|  |  |

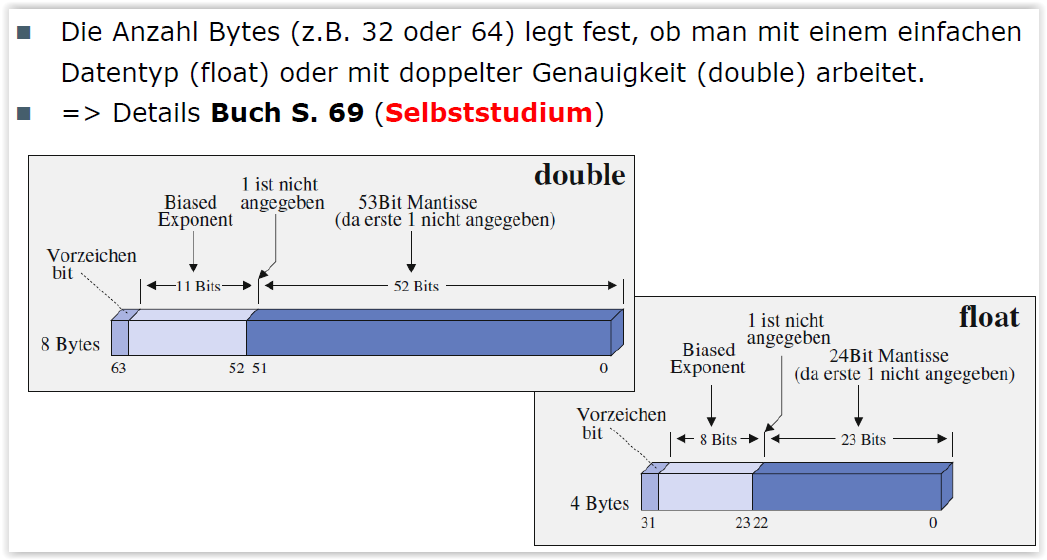
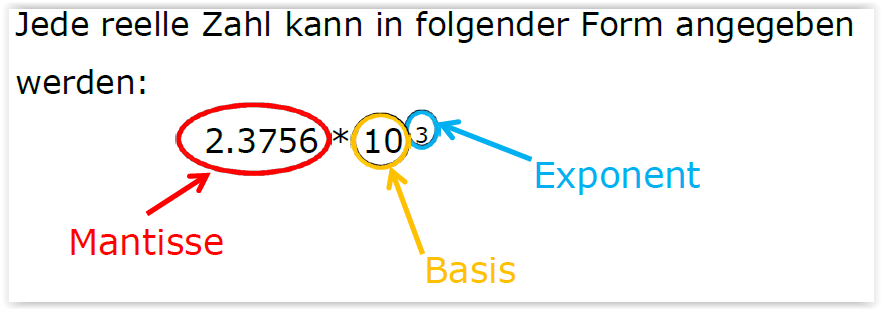
### Rechnungen:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Addition:**    Ist der Übertrag am Ende der Rechnung gibt es einen Überlauf. Z.B. 10 + 7 und wir haben nur 4 Bits, Ist das Resultat 1, da wir einen Überlauf haben (Bsp. Links) | Bsp. Überlauf   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 1 | 0 | 1 | 0 | | + | | | | | |  | 0 | 1 | 1 | 1 | | 1 | 0 | 0 | 0 | 1 | |
| **Subtraktion (Addition von negativen Werten):**  Um diese negativen Werte zu erhalten werden sogenannte Komplemente Gebildet. | |
| **Einerkomplement:**  Ist die erste Zahl MSB(Most Significant Bit) eine 1 ist die Zahl negativ.  Negative Zahlen sind Symmetrisch 🡪      Bekommt man bei der Subtraktion mit dem Einerkomplement, wie oben einen Überlauf, muss dem Resultat eins Addiert werden. |  |
| **Zweierkomplement:**  Ist die erste Zahl MSB(Most Significant Bit) eine 1 ist die Zahl negativ.  Zahlen sind Definiert 🡪  Um die Negative Zahl zu berechnen wird die Zahl invertiert und eins Addiert 🡪 |  |

## Wertebereiche:

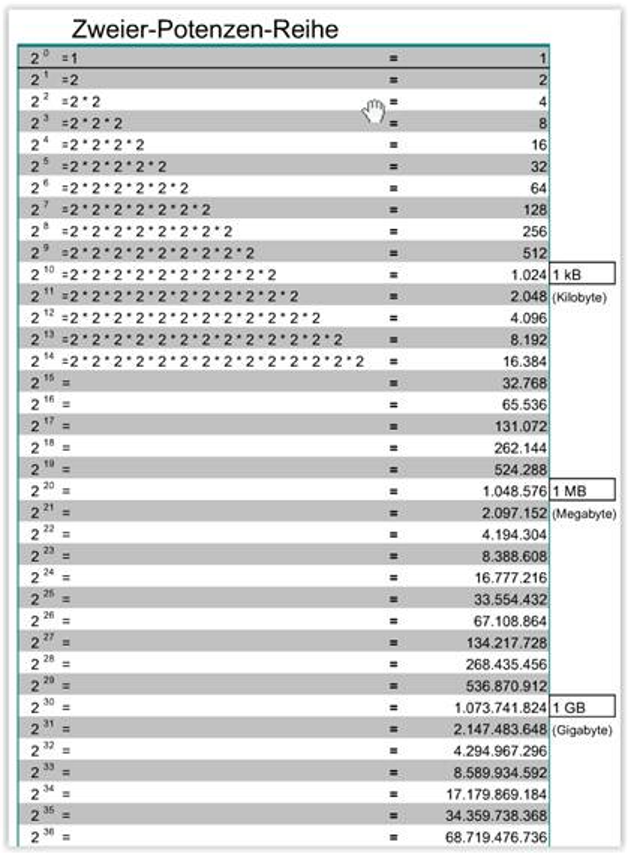


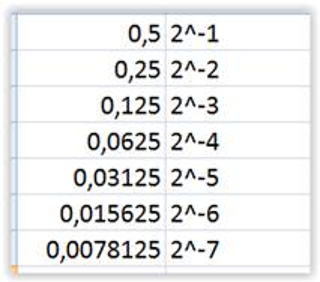
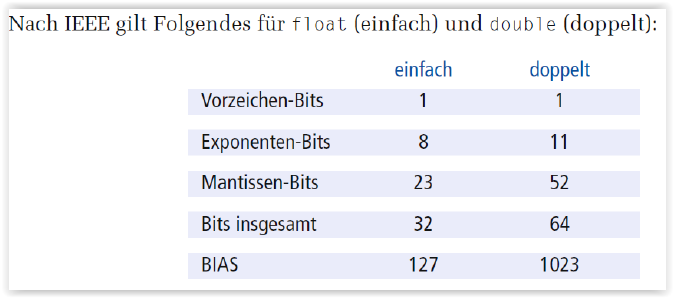
## Gleitpunktzahlen



**Single (float)** Biased exponent immer minus 127

**Doppelt (Double )** Biased exponent immer minus 1023

Bei der Gleitkommadarstellung muss normalisiert werden, da sonst eine Zahl mehrere Darstellungen hätte. Darum gibt es immer nur eine Stelle vor dem Komma und die ist immer 1 und wird deshalb auch nicht gespeichert.

Bsp:

# Tabellen- und Zeichencodes (Kap. 3, p. 71ff)

Zeichensatz = Liste von Zeichen

Zeichenkodierung = Eindeutige Zuordnung von Zeichen zu einer Zah

## ACII

Computergenerierter Alternativtext:
Scan- ASCII zchen
code hex dez
Scan- ASCII
Zch.
code hex dez
Scan- ASCII
Zch.
code hexdez
Scan- ASCII
code hexdez C
00 0 NULA@
01 1 SOHAA
02 2 STXAB
03 3 ETXAC
04 4 EOTAD
05 5 ENQAE
06 6 ACKAF
07 7 BELAG
CE 088 BSAH
OF 09 9 TABAI
OA 10 LF AJ
OB 11 VT AK
OC 12 FF AL
1C OD 13 CRAM
OE 14 SO AN
OF 15 SI AQ
10 16 DLEAP
11 17 DC1AQ
12 18 DC2AR
13 19 DC3AS
14 20 DC4AT
15 21 NAKAU
16 22 SYNAV
17 23 ETBAW
18 24 CANAX
19 25 EM A
1A 26 SUBAZ
01 1B 27 EscA[
1C 28 FS A
1D 29 GS 9
1E 30 RS AA
1F 31 US
20 32
02 21 33
03 22 34
29 23 35
05 2436
06 25 37
07 2638
OD 27 39
09 2840
OA 2941
1B 2A42
1B 2B43
33 2C44
35 2D45
34 2E46
08 2F47
OB 3048
02 31 49
03 32 50
04 33 51
05 34 52
06 35 53
07 3654
08 37 55
09 38 56
OA 3957
34 3A58
33 3B59
2B 3C6O
OB 3D61
2B 3E62
OC 3F63
40 64
1E 41 65
30 42 66
2E 4367
20 4468
12 45 69
21 4670
22 4771
23 4872
17 49 73
24 4A74
25 4B75
26 4C76
32 4D77
31 4E78
18 4F79
19 50 80
10 51 81
13 52 82
1F 53 83
14 5484
16 55 85
2F 56 86
11 57 87
2D 58 88
2C 59 89
15 5A9O
5B 91
5C 92
5D 93
29 5E94
35 5F 95
sp
‚1
$
%
(
+
/
0
1
2
3
4
5
6
7
8
9
<
>
‘2
OD 6096
A 1E 6197 a
B 30 6298 b
C 2E 6399 c
D 20 64100 d
E 12 65101 e
F 21 66102 f
G 22 67103 g
H 23 68104 h
17 69105 i
J 24 6A106 j
K 25 6B107 k
L 26 6C108 1
M 32 6D109 m
N 31 6E110 n
O 18 6F111 o
P 19 70112 p
Q 10 71 113 q
R 13 72114 r
S 1F 73115 s
T 14 74116 t
U 16 75117 u
V 2F 76118 v
W 11 77 119 w
X 2D 78120 x
Y 2C 79121 y
Z 15 7A122
7B123 {
7C124 1
7D125 }
A 7E126 —
— 53 7F 127 DELComputergenerierter Alternativtext:
HTML- (Windows-1 252)
code hexdez (Zch.)
HTML- ISO-8859-1
code hex dez Zch.
HTML- ISO-8859-1
code hex dez Zch.
HTML- ISO-8859-1
code hexdezZch.
&#x80; 80 128
€
&nbsp;
A0 160
&Agrave;
Co 192
Ä
&agrave;
EO 224
&#x81; 81 129
&iexcl;
Al 161
&Aacute;
Cl 193
A
&aacute;
El 225
ä
&#x82; 82 130
&#x83; 83 131
&#x84; 84 132
‘
f
„
&ceni
&pound;
&curren;
A2 162
A3 163
A4 164
£
a
&Acirc;
&Atilde;
&AumI;
C2 194
C3 195
C4 196
A
A
Ä
&acirc;
&atilde;
&auml;
E2 226
E3 227
E4 228
ä
ä
ä
&#x85; 85 133
...
&yen;
A5 165
&Aring;
C5 197
Ä
&aring;
E5 229
&#x86; 86 134
t
&brvbar;
A6 166
&AEIig;
C6 198
?E
&aelig;
E6 230
&#x87; 87 135
1
&sect;
A7 167
§
&Ccedil;
C7 199
9
&ccedil;
E7 231
&#x88; 88 136
‘
&uml;
A8 168
&Egrave;
C8 200
E
&egrave;
E8 232
&#x89; 89 137
&#x8A; 8A 138
%
&copy;
&ordf;
A9 169
AA 170
©
a
&Eacute;
&Ecirc;
C9 201
CA 202
E
E
&eacute;
&ecirc;
E9 233
EA 234
&#x8B; 8B 139
&#x8C; 8C 140
<
E
&Iaquo;
&not;
AB 171
AC 172
«
-‘
&EumI;
&Igrave;
CB 203
CC 204
!
&euml;
&igrave;
EB 235
EC 236
i
&#x8D; 8D 141
&shy;
AD 173
&Iacute;
CD 205
1
&iacute;
ED 237
r
&#x8E; 8E 142
Z
&reg;
AE 174
®
&Icirc;
CE 206
1
&icirc;
EE 238
T
&#x8F; 8F 143
&macr;
AF 175
&IumI;
CF 207
1
&iuml;
EF 239
T
&#x90; 90 144
&deg;
B0 176
O
&ETH;
D0 208
D
&eth;
F0 240
Ö
&#x91; 91 145
&#x92; 92 146
‘
&plusmn;
&sup2;
B1 177
B2 178
±
2
&Ntilde;
&Ograve;
Dl 209
D2 210
N
0
&ntilde;
&ograve;
F1 241
F2 242
6
&#x93; 93 147
“
&sup3;
B3 179
&Qacute;
D3 211
0
&oacute;
F3 243
6
&#x94; 94 148
“
&acute;
B4 180
‘
&Qcirc;
D4 212
0
&ocirc;
F4 244
6
&#x95; 95 149
•
&micro;
B5 181
p
&Qtilde;
D5 213
0
&otilde;
F5 245
ö
&#x96; 96 150
—
&para;
B6 182
¶
&Ouml;
D6 214
0
&ouml;
F6 246
ö
&#x97; 97 151
—
&middot;
B7 183
&times;
D7 215
x
&divide;
F7 247
--
&#x98; 98 152
&cedil;
B8 184
‚
&Oslash;
D8 216
0
&oslash;
F8 248
ø
&#x99; 99 153
TM
&supl;
B9 185
&Ugrave;
D9 217
Y
&ugrave;
F9 249
ü
&#x9A; 9A 154
&ordm;
BA 186
O
&Uacute;
DA 218
Y
&uacute;
FA 250
ü
&#x9B; 9B 155
&raquo;
BB 187
»
&Ucirc;
DB 219
U
&ucirc;
FB 251
ü
&#x9C; 9C 156
&fracl4;
BC 188
%
&Uuml;
DC 220
Y
&uuml;
FC 252
ü
&#x9D; 9D 157
&fracl2;
BD 189
1,4
&Yacute;
DD 221
Y
&yacute;
FD 253
‘
&#x9E; 9E 158
&frac34;
BE 190
3/4
&THORN;
DE 222
&thorn;
FE 254
&#x9F; 9F 159
Y
&iquest;
BF 191
‚
&szlig;
DF 223
ß
&yuml;
FF 255
y 

## Unicode

Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende

Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein

digitaler Code festgelegt wird.

* Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in

verschiedenen Ländern oder Kulturkreisen zu beseitigen. U

* Unicode wird ständig um Zeichen weiterer Schriftsysteme ergänzt (heute in der Version 6.2)
* Die Nummerierung ist hexadezimal in der Schreibweise U+XXXXXXXX
* z.B. Beispiel: U+00B6 ist das "Pilcrow-Zeichen", wie wir es aus Word kennen: "¶".
* Üblich sind eine 4-/6- oder 8-stellige Hex-Schreibung (16/24/32 Bits) – Obiges ist ein Beispiel

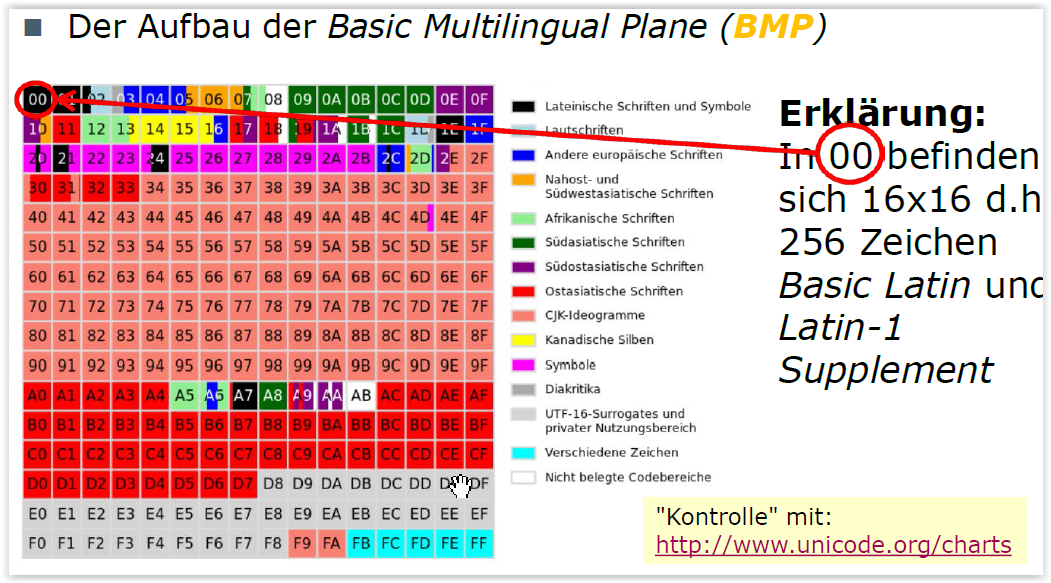
für 4 Stellen.

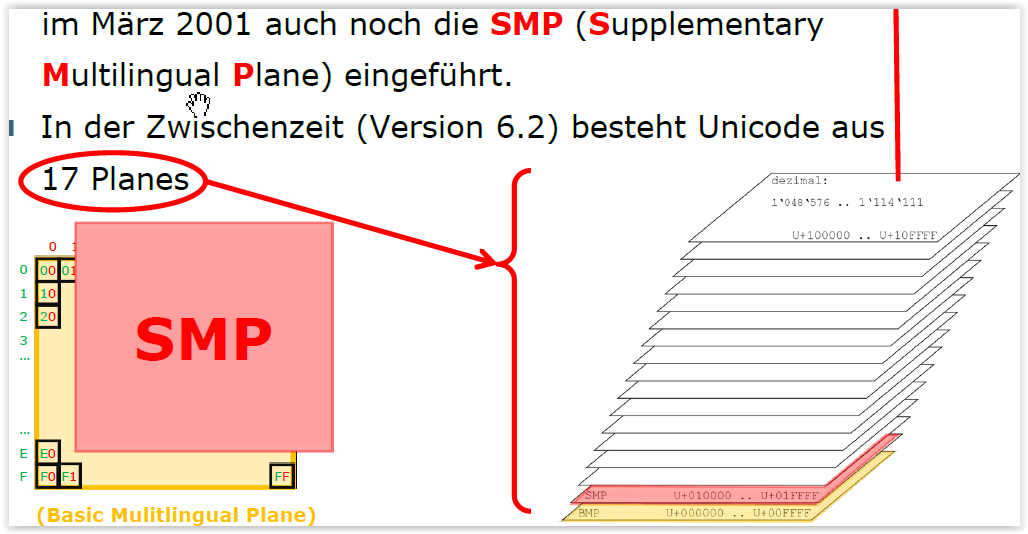
* Führende Nullen können in dieser Schreibung paarweise weggelassen werden.
* Die Identifikation durch eine Bezeichnung ist definiert und kann übersetzt werden

Die Zeichen des Unicode sind in so genannten Planes organisiert. Eine Plane ist eine quadratische

Tabelle von 256 Zeilen zu 256 Spalten, also 216 = 65'536 Feldern.

Dies sind die Code Points und werden innerhalb der Plane fortlaufend nummeriert.





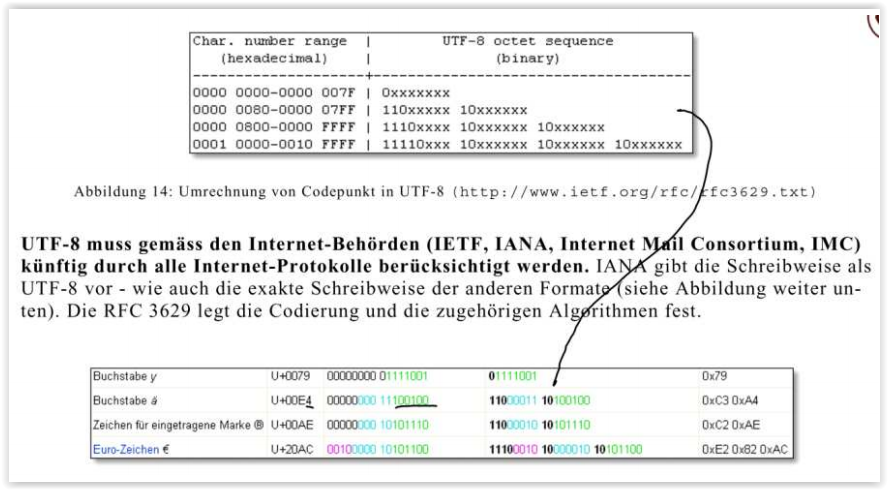
## UTF-Unicode Transformation Format

UTF ist also eine Methode, Unicode-Zeichen auf Folgen von Bits abzubilden.

Für die Repräsentation der Unicode-Zeichen zum Zweck der elektronischen Datenverarbeitung gibt

es verschiedene UTFs.UTF-8 empfiehlt sich für Texte mit vorwiegend lateinischen Buchstaben. UTF-16 empfiehlt sich für Texte mit z.B. asiatischen Zeichen. UTF-32 empfiehlt sich künftig für höchste Performanz (bei grösstem Speicherbedarf)

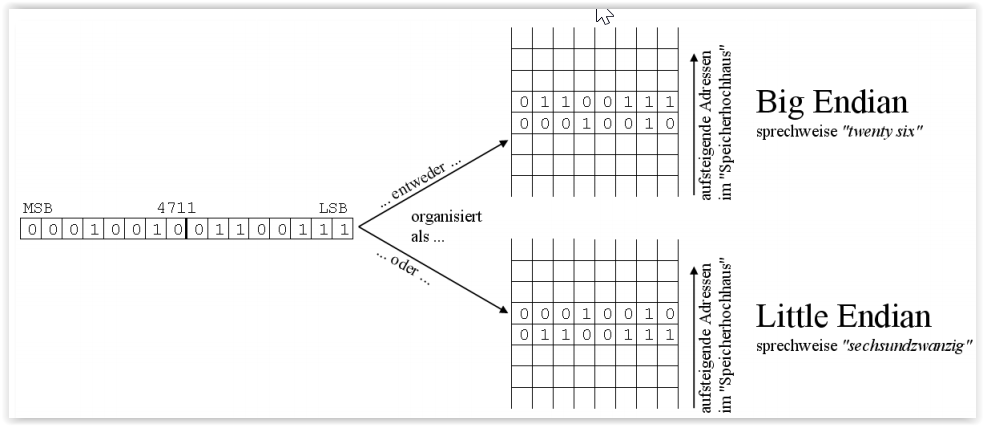
Alles Kann in UTF-8 Dargestellt werden muss aber wenn zu gross wie unten umgewandelt werden(die x Werden von hinten nach vorne aufgefüllt, bleiben übrig wird mit 0 aufgefüllt). UTF-16 und 32 Übernehmen den Binären Wert eins zu eins.



## Big und Little Endian

• Womit arbeiten MAC / Windows?

|  |  |  |
| --- | --- | --- |
| OS | Endian | Was ist besser |
| Windows | Little Endian | Also, because of the 1:1 relationship between address offset and  byte number (offset 0 is byte 0), multiple precision math routines  are correspondingly easy to write. |
| Linux | Big Endian | The numbers are also stored in the order in which they are  printed out, so binary to decimal routines are particularly efficient |



# Exkurs: Codegewinnung

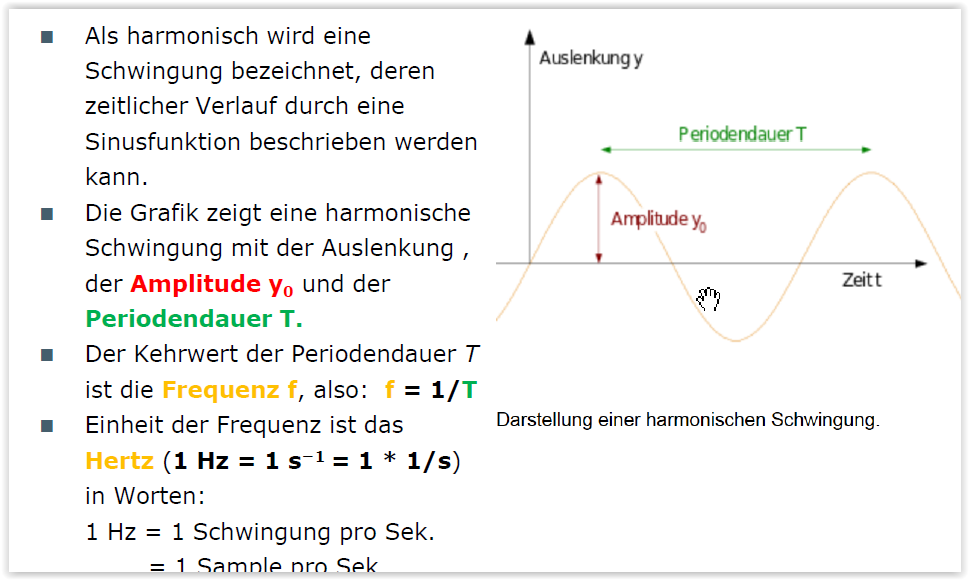
## Diskretisierung

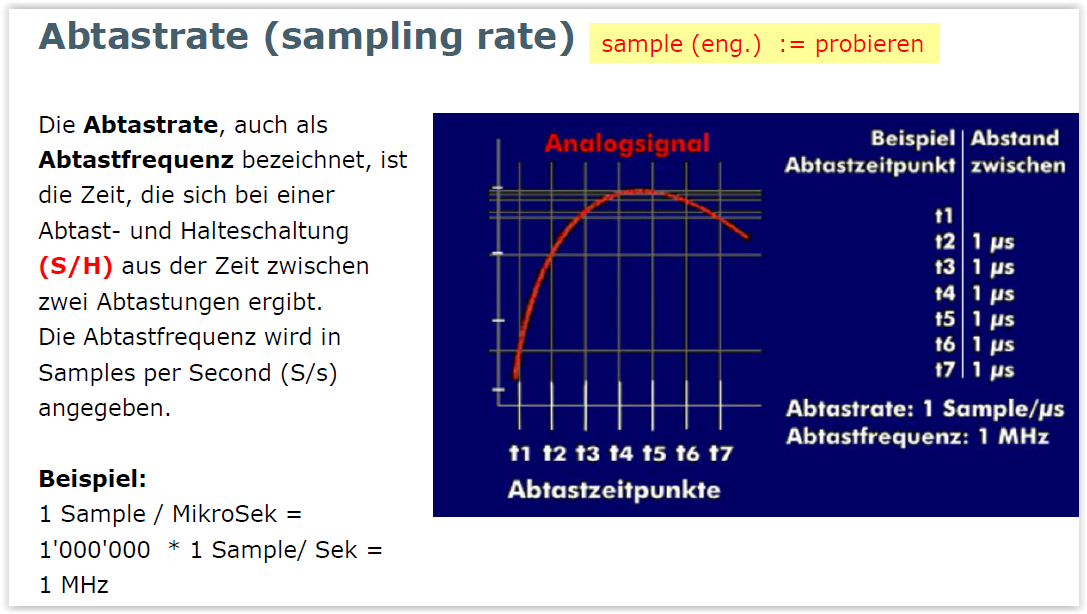
|  |  |
| --- | --- |
| Bei der Diskretisierung eines Kontinuierlichen (Analogen) Signals wird regelmässig gemessen und dann auf die darunterliegende diskrete Stufe abgerundet. Der Wert wird gehalten bis zur nächsten Messung. |  |
| Um eine Analoges Signal in einen Bitstrom umzuwandeln muss zuerst Diskretisiert werden, wie oben beschrieben. Anschliessend wird jeder Diskreten Stufe ein binäres Wort zugewiesen. Im letzten Schritt werden alle Binären Worte aneinandergereiht. Daraus ergibt sich der Bitstrom |  |

## Modulierung

|  |  |
| --- | --- |
| Mit einer Bit-Übertragungsfolge sind kontinuierliche und analoge Signale bereit für den Transport und /oder die Speicherung in der Digitaltechnik.  Vorher müssen aber die Bits in elektronische Pegel umgewandelt werden. Das nennt sich Modelierung |  |
| Als Beispiel hier die Manchester Codierung.  Bei der Manchester Codierung wird die 1 jeweils als abfallende Flanke dargestellt und die 0 als aufsteigende oder wie man im Beispiel rechts gut sehen kann, ist die linke Hälfte innerhalb des Taktes hoch ist es eine 1 und wenn die rechte Seite hoch ist eine 0 |  |

## Harmonische Schwingung





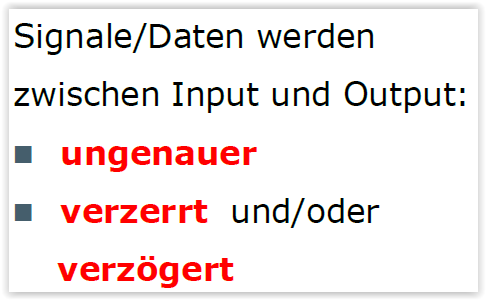
# Fehlertolerante Codierung (Kap. 20, p. 734ff)

## Code-Belegung

## 

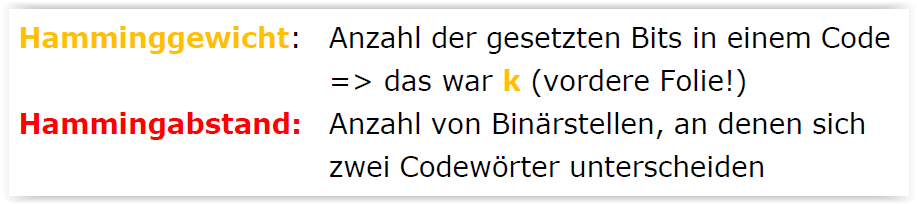
## Fehlererkennung/Korrektur

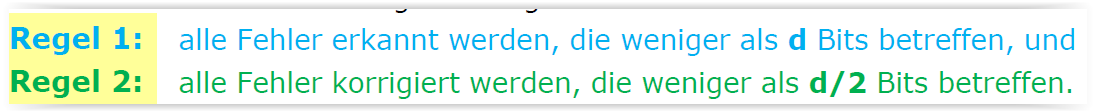
Dämpfung = Signal verliert an Genauigkeit während es gesendet wird. Bsp. Je weiter ich von jemandem weg bin desto schlechter versteht er mich.

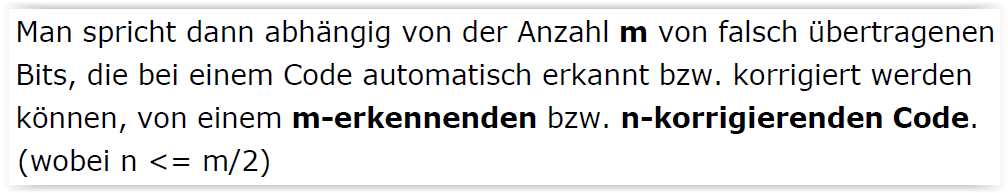


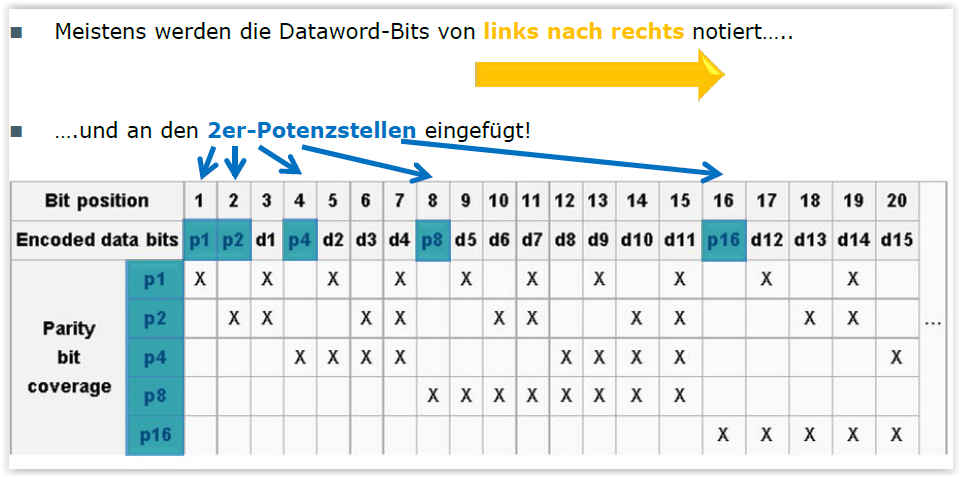
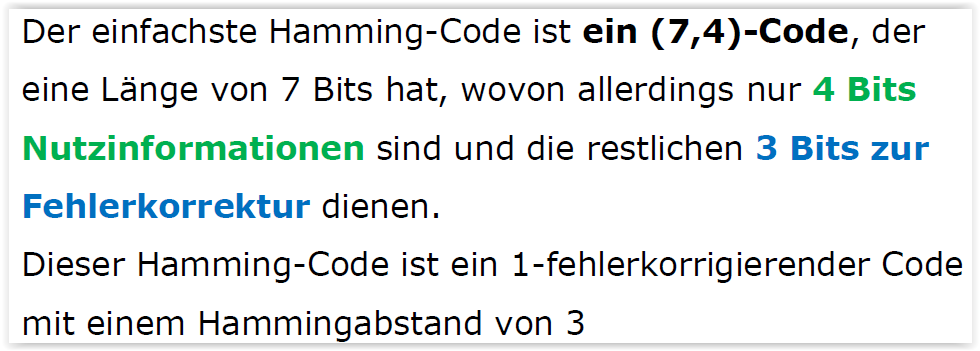
|  |  |
| --- | --- |
|  |  |
|  |  |

## Hammingcode

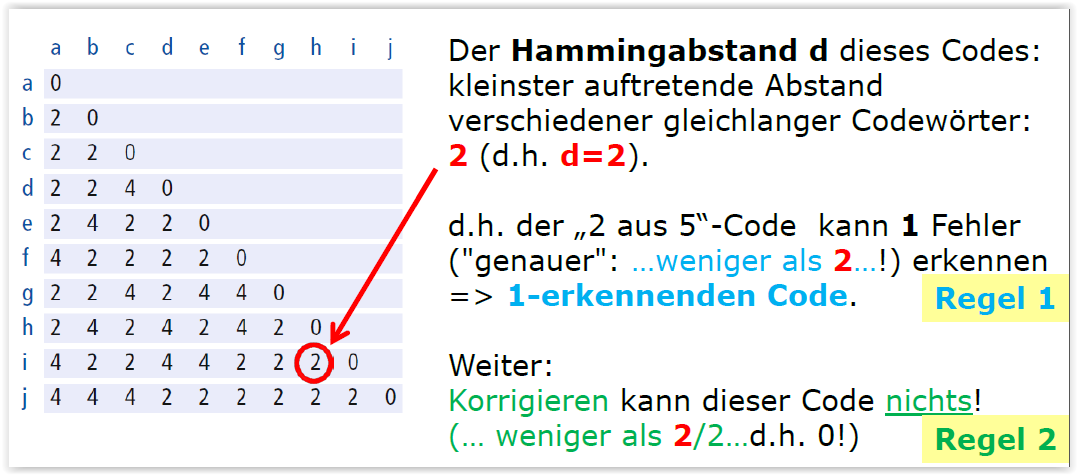




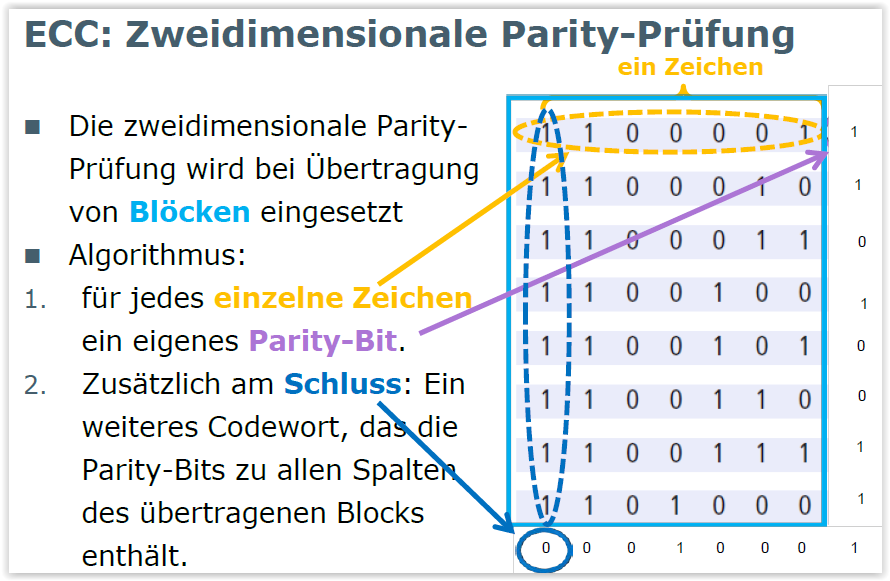
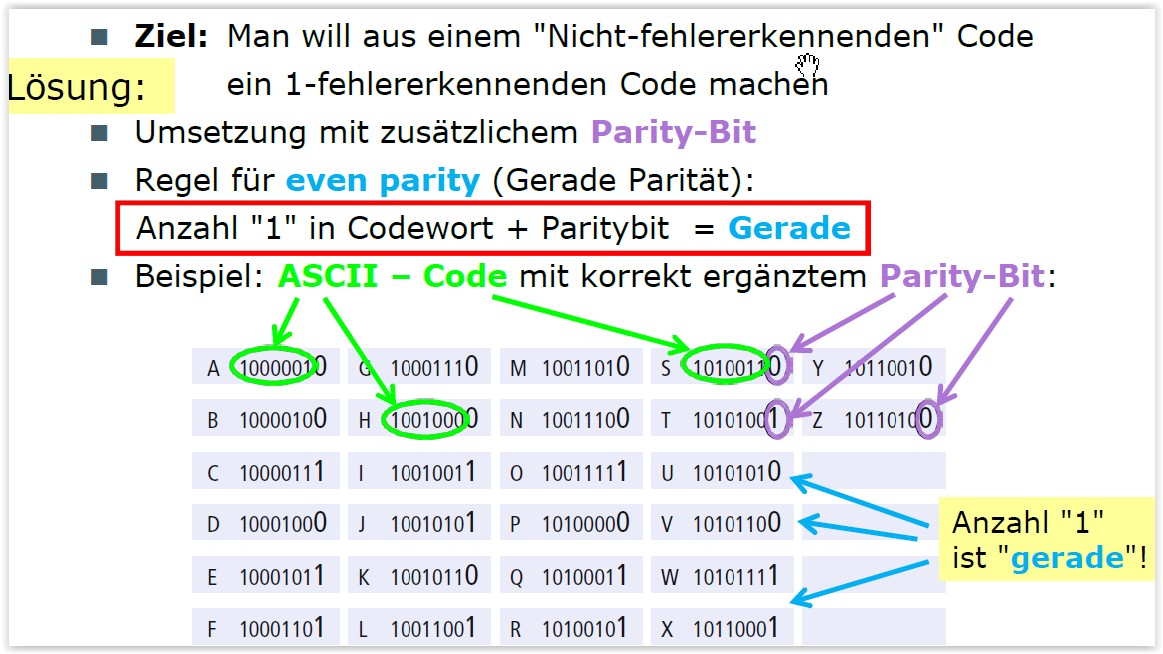




Beispiel:



## Paritybit



Bei der Zweidimensionalen Parity-Prüfung kann ein Fehler erkannt und korrigiert werden. 2-3 Fehler können nur erkannt werden.

### Parity Bit Positions:

1,2,4,8,16,32,64 etc.

### Leave Space for PB :

\_ \_ 1 \_ 001 \_ 1010

• Position 1 checks bits 1,3,5,7,9,11:

? \_ 1 \_ 0 0 1 \_ 1 0 1 0. Even parity so set position 1 to a

0: 0 \_ 1 \_ 0 0 1 \_ 1 0 1 0

• Position 2 checks bits 2,3,6,7,10,11:

0 ? 1 \_ 0 0 1 \_ 1 0 1 0. Odd parity so set position 2 to a 1: 0 1 1 \_ 0 0 1 \_ 1 0

1 0

• Position 4 checks bits 4,5,6,7,12:

0 1 1 ? 0 0 1 \_ 1 0 1 0. Odd parity so set position 4 to a 1: 0 1 1 1 0 0 1 \_ 1 0

1 0

• Position 8 checks bits 8,9,10,11,12:

0 1 1 1 0 0 1 ? 1 0 1 0. Even parity so set position 8 to a 0: 0 1 1 1 0 0 1 0 1 0

1 0

• Code word: 011100101010.

### 14.2 Fixing

• **Received:** 011100101110 Nach Prüfung Fehler (Parity Bit nicht even)

an Position 2 + 8 = 10 Fehler an der Position 10

**Beweis:**

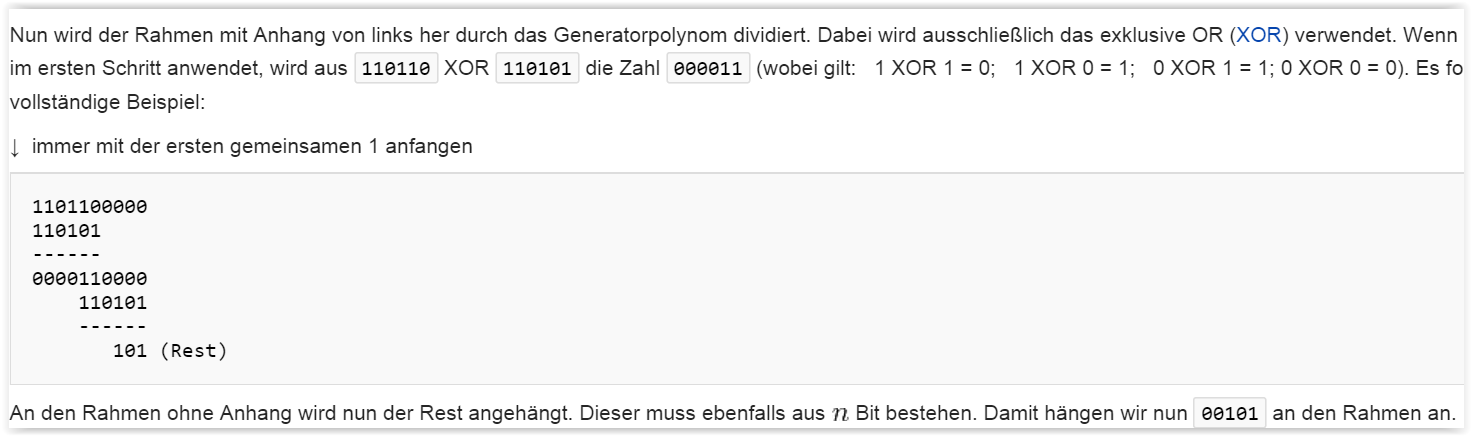
Code-Wort: 011100101010 (So wäre es richtig)

Received : 011100101110 (Falsches bit)

## CRC(cyclic redundancy check)-Codierung

Bei der CRC-codierung wird eine Polynomdivison des Werts vorgenommen.



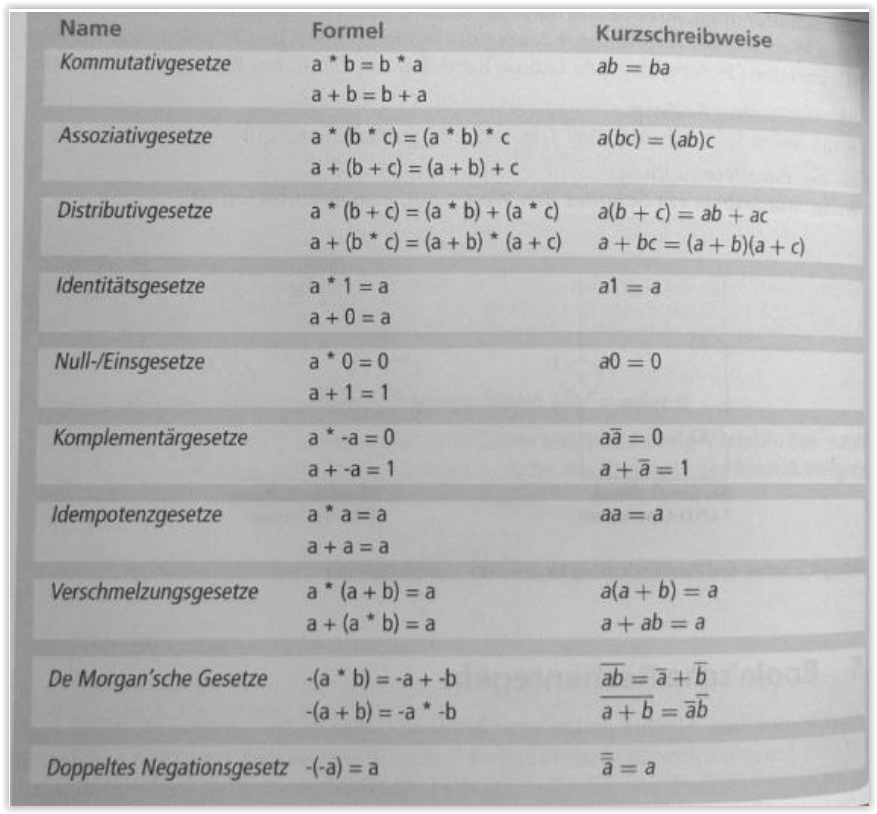


# Holzi

# 

# Boolesche Algebra (Kap. 4, p. 82ff)

**Boolesche Algebra:**



OR = a + b oder a v b

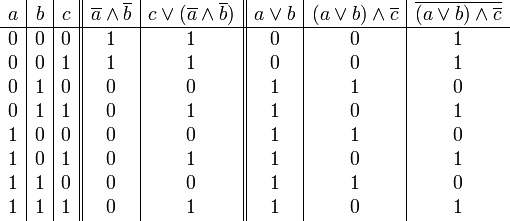
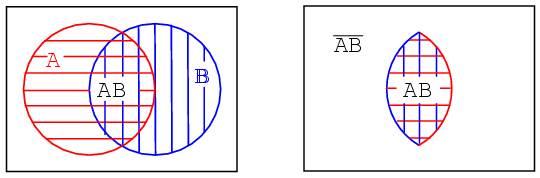
AND = a\*b oder a^b

NOT = ¬ a oder a darüber -

Computergenerierter Alternativtext:
Wahrheitstabelle OR: Symbol OR:
L’ 1bt (JR 1)1
I()t0I ()
blu i I
11101 1 I
lilul i
Wahrheitstabelle AND: Symbol AND:
‘lilla AND bi
0oI o
d. IiI  I
I ûI o
Ii i i I
Wahrheitstabeile NOT: Symbol NOT:
IaINOT al a ã
mi  i
t 1•11 oComputergenerierter Alternativtext:
Hochschule Lusern
Wichtige Gesetze für Boolesche Algebra
De Morgan Gesetze I Verschmelzungsgesetz
. De Morqan ___
(a* b)=-a+-b ab =+S
a+b=b
. Verschmelzungsgesetz
a*(a+b)=a a(a+b)=a
a+(a*b)=a a+ab=a
a*(ã+b)=ab a(ã+b)=ab
a+(ã*b)=a+b a+ãb=a+b

Kann bewiesen werden mit Venn-Diagramm oder einer Wahrheitstabelle.

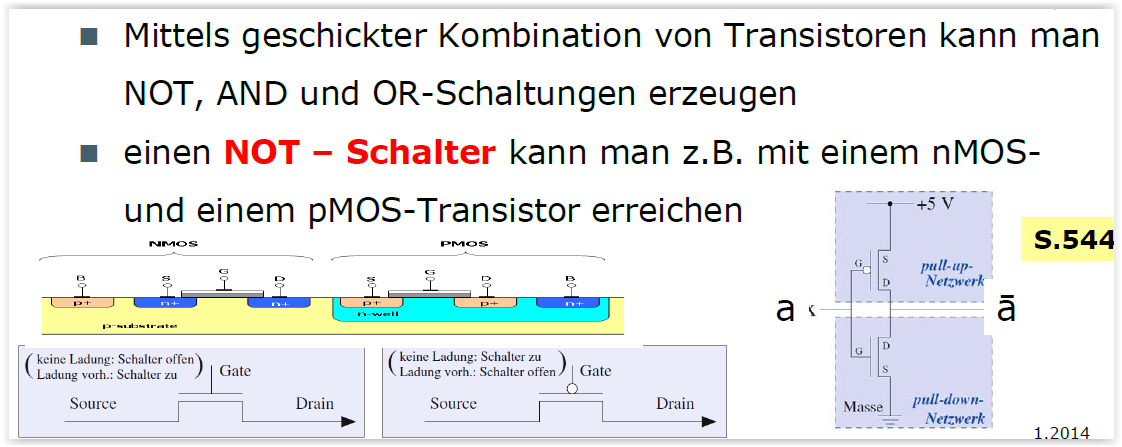
Venn-Diagramm: Wahrheitstabelle:



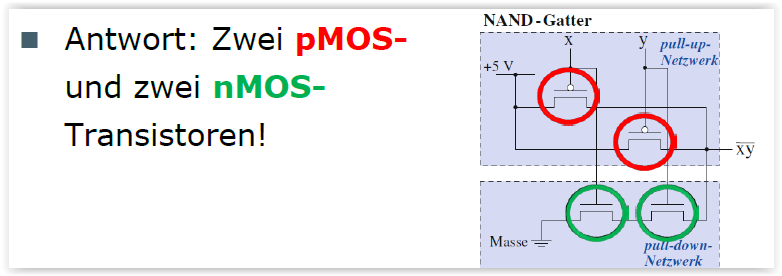
Zweistellige Boolesche Funktionen Buch Seite 87!

## Transistor

Ein Transistor entspricht einem Schalter ohne Mechanik. Früher waren das Relais.

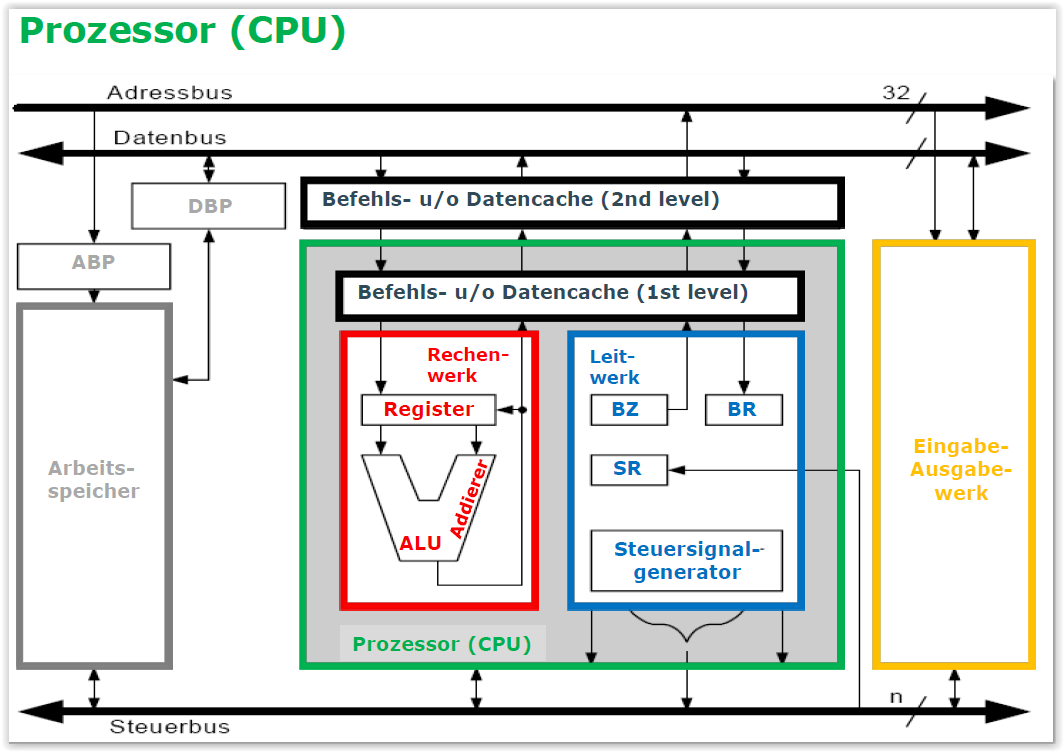
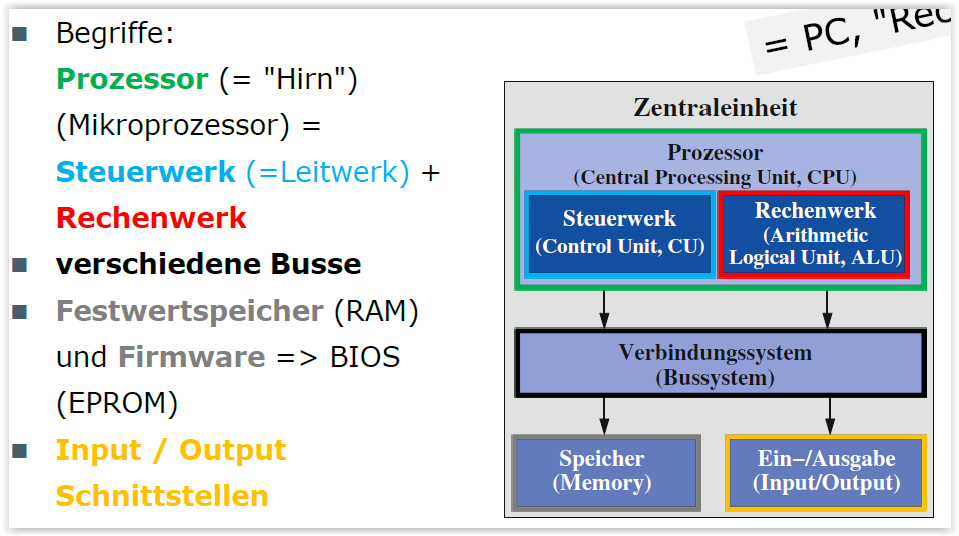


Computergenerierter Alternativtext:
Hochichuic Luztin
W rtschaft
AND- und OR-Schaltung (Gatter) als Symbole
AND I
o o o a
::: b
i i i
http:/Thyperphysics.phy-astr.gsu.edu/hbase/electi-onic/and.htrnl
IDa-
o o o
a+b
i o i
i i i
http:/fhyperphysics.phy-aszr.gs .edu/hbase/electronic/or.html
NAND- und NOR-Schaltung (Gatter) als Symbole
D
ab
a
b
NAND
o o i
o i i
1E__ab
i i o
a
b
[I] 
o o i
a+b
i o o
i i o
http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/nand.html http:ffhyp.er-physic_:H u.duhase/eectronic/nc-,h:rrI
, ¿risuwt für Wiitschaftsinformatik IWI
ruile o 



## Hardware und Firmware (Kap 5, p. 92ff)

EVA Prinzip = Eingabe Verarbeitung Ausgabe



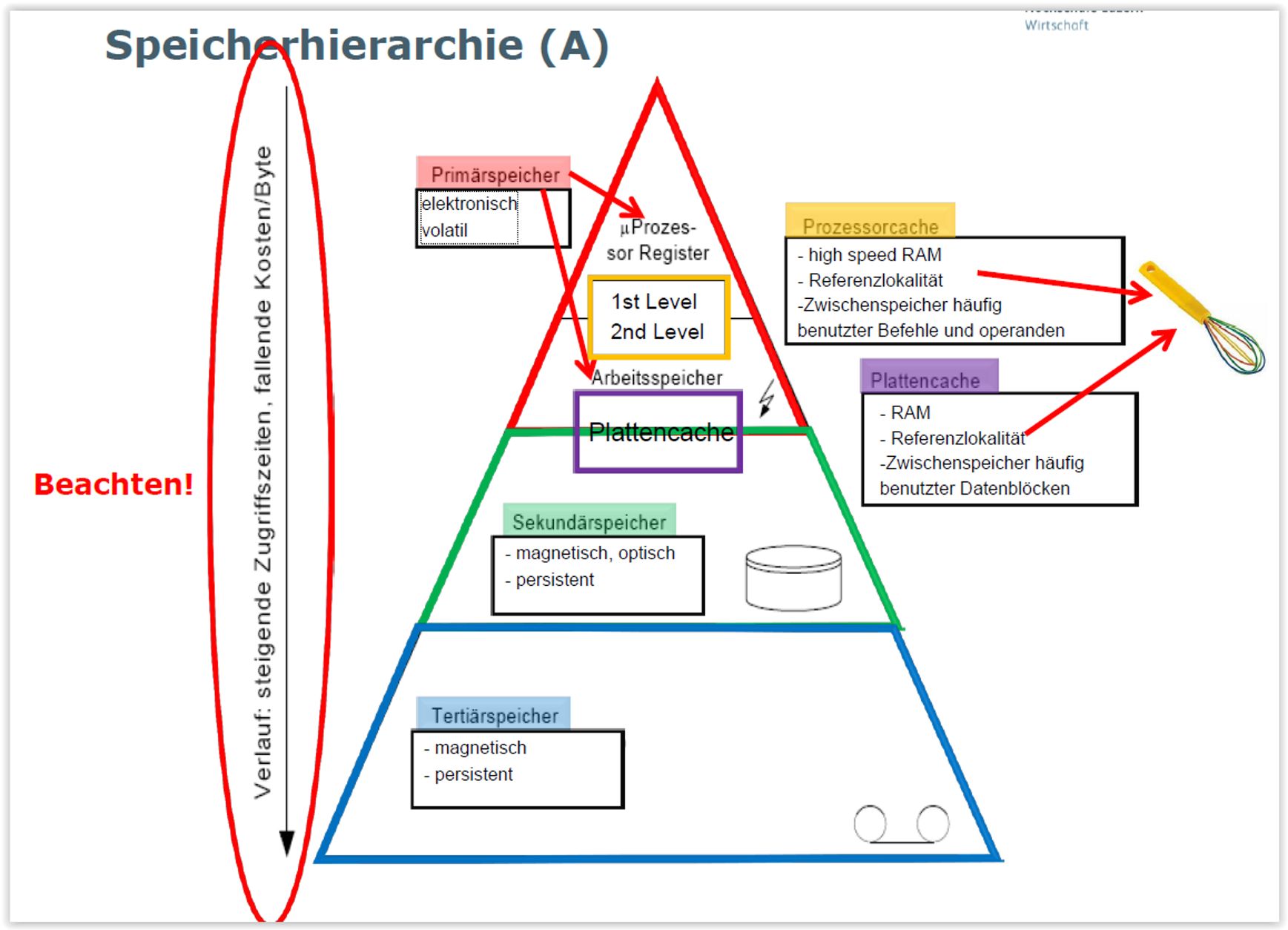
## Speichergrössen

Computergenerierter Alternativtext:
MiB vs MB oder GIB vs GB etc

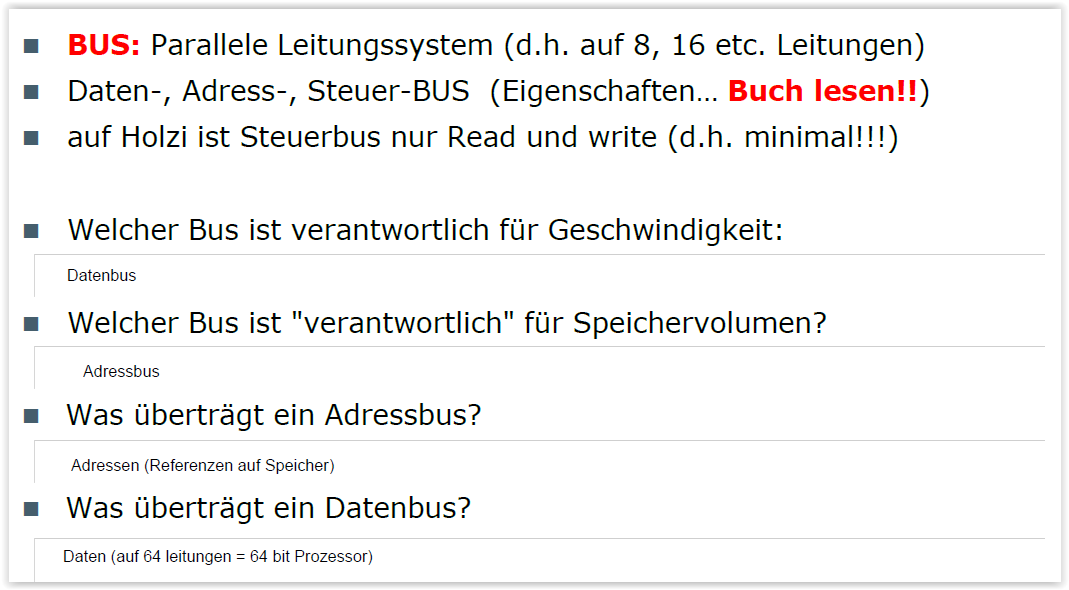
Computergenerierter Alternativtext:
1024 = 210 ca. 1O = 1000Computergenerierter Alternativtext:
Basis 2: statt 1ù:
kibi KiB KB
mebi MiB L
GihiByte
tebi TiB
pebi PiB
exbi EIB

## Speicher-Hierarchie

Unterschied von Zugriff auf Ram zu Festplatte = 1:25

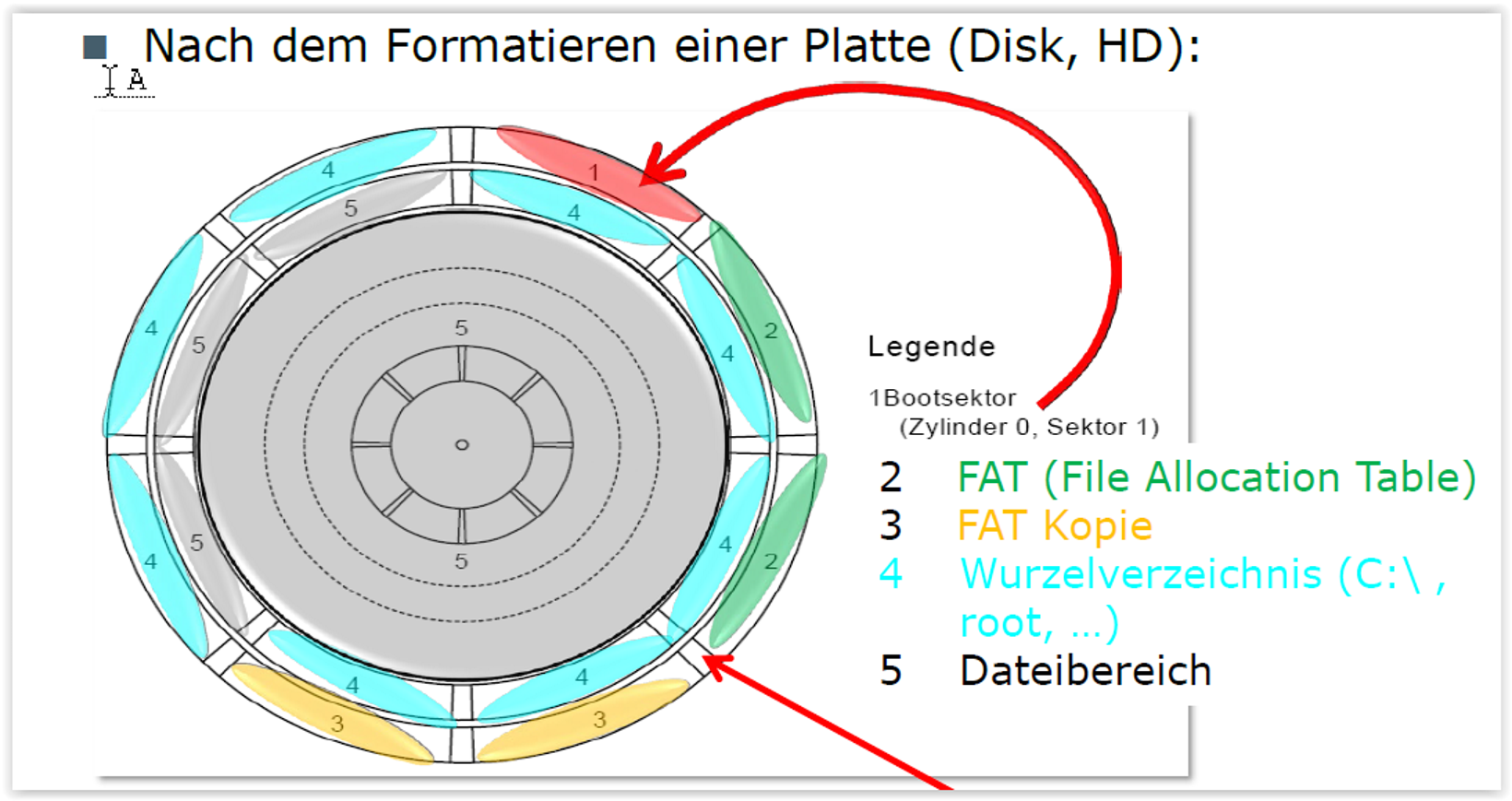


Primärspeicher ist weg sobald die Stromversorgung unterbrochen wird.

Computergenerierter Alternativtext:
gleiche Prinzip...;-)...!
]
PROM
E PR Q M
evtl. flash
• BIOS: Ein Chip der die
Firmware enthàì
• heute:
[
EFI Extensible [
Firmware Interface) E
Kopie als 1 
UEFI (Universal EFI) ab RAMBIos[
Parameter [
1998 (Intel) aIsBIOS ,-
RAMm 1 [
SRAM mit
ist aber immer das Batterie-
stütze
power on
self test
Befehl zum laden des OS auf
einem Speichermedium
I boot
loader
]
]
]
]
]-
]
Grundlegende I/O
Operationen
diverse Einstellungen wie
¡z.B. Bootreihenfolge ii 

Computergenerierter Alternativtext:
. Es gibt also mindestens 2 verschieden
Möglichkeiten:
n(cr )
___________I___________
Interrupt causes current
pI’OCessin k) SfO
Return frnn interrupt
Normal proces’thuz
‘esu’nes
>
‘V
I.
Con;inu )
Polling (Busiy Waiting, Round Robin)
Interrupt Requesting
1ssu reid or write
request to disk
Do othcr roccsing,
iinil disk icucs an
interrupt
IILIlSfCL’ Litita betwthn
dkk ¿iiicl memory
Yes

## Disks

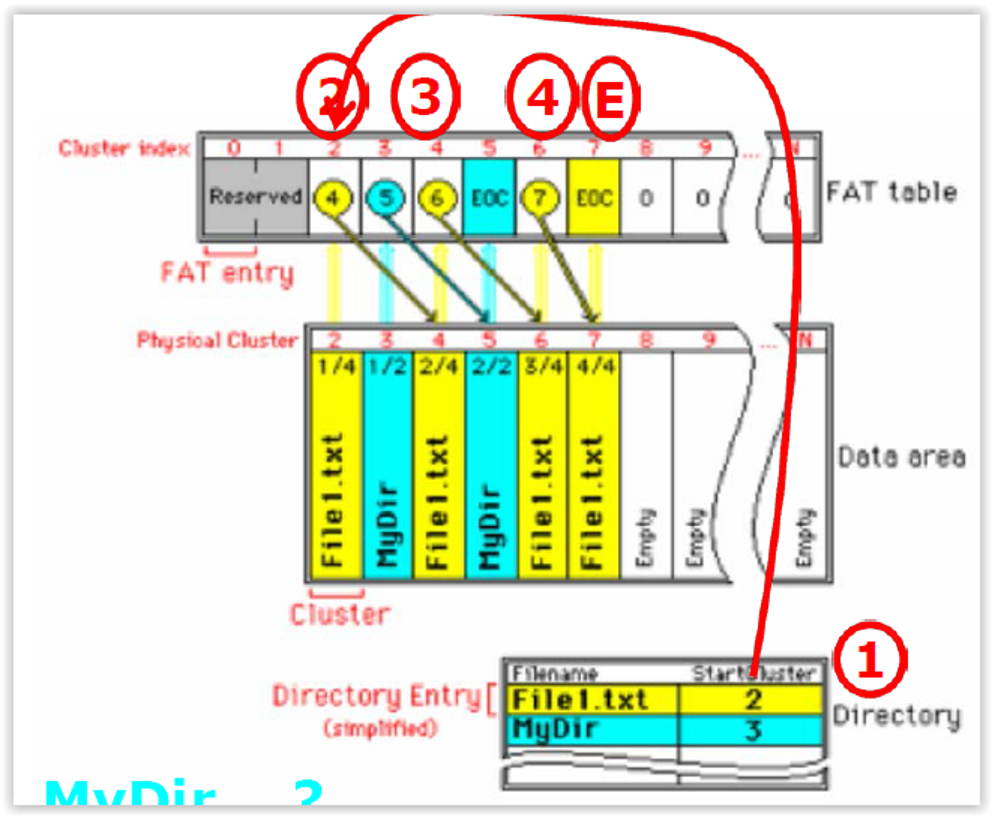


**Definition - What does *Cluster (Disk)* mean?**

A cluster, in the context of a hard disk, is a group of sectors within a disk and is the grouping by which disk files are organized. A cluster is larger than a sector, and most files fill many clusters of disk space. The hard drive is able to find all the clusters on a disk because each cluster possesses its own ID.

* Interne Fragmentierung = Wenn Letzter Cluster Platz nicht ganz gebraucht wurde, also noch Bytes Platz hätten
* Externe Fragmentierung = (Plattenfragmentierung) Zerstückelung von Dateien in nicht zusammenhängende Cluster

Beispiel wie ein File gelesen wird:



## Miniaturisierung:

SSD Technologien

|  |  |  |
| --- | --- | --- |
| SLC (Single layer Cell) | TLC (Three layer Cell) | MLC(Multi layer Cell) |
| * + Nachteile     - Hohe kosten     - Eher Kleinere Harddisks   + Vorteile     - Hohe Haltbarkeit     - Hohe Performance     - Niedriger energieverbrauch | * + Nachteile     - Schwache Performance     - Niedrige Haltbarkeit   + Vorteile     - Tiefer Preis     - Möglichst grosse Harddisk | * + Nachteile     - Durchschnittliche Performance     - Durchschnittliche Haltbarkeit   + Vorteile     - Bestes Preis/Leistungsverhältnis |

## RAID (Redundant Array of Inexpensive/Independet Disks):

Wichtigste:

|  |  |  |
| --- | --- | --- |
| RAID 0 (Striping) | RAID 1 (Mirroring) | RAID 5 (Striping with parity) |
| Computergenerierter Alternativtext: RAID O Al A2 A3 A4 A5 A6 A A8 Disk O Disk 1   * Dateien sind Verteilt zwischen Disks * Hohe Performance * Keine Redundanz | Computergenerierter Alternativtext: RAID i Al Al A2 LA2 A3 LA3 A4 A4 Disk O Disk i   * Alles wird dupliziert zwischen zwei Disks * Doppelte Diskgrösse wird verbraucht (100GB Daten = 200GB Disks) * Hohe Redundanz (Alles doppelt vorhanden) | Computergenerierter Alternativtext: __ RAID 5 Disk O Disk i Disk 2 Disk 3   * Daten werden gesplittet, Dabei wird auf eine Disk ein Partity gespeichert (Bei Verlust eines Blocks kann es mit dem Parity rekonstruiert werden) * Ein wenig erhöhter Speicherplatzverbrauch (Für Parity) * Hohe Redundanz (Parity Berechnung kann Performance beeinflussen). |

|  |  |
| --- | --- |
| Software RAID | Hardware RAID |
| * + Feature des Betriebssystems   + Braucht keine spezielle Hardware   + Schlechtere Performance   + Billiger | * + Ist im Harddisk Controller   + Betriebssystem weiss nicht, dass ein RAID vorhanden ist.   + Hohe Performance   + Teurer |

|  |  |
| --- | --- |
| Computergenerierter Alternativtext: RAID 0+1 RAID i RAIDO RAIDO Disk O Disk i Disk 2 Disk 3 | * + Kombination aus RAID 0 und RAID 1   + Zuerst zwei Platten mit RAID 0   + Die so entstandenen Logical Volumes werden als RAID 1 Zusammengefasst. |
| Computergenerierter Alternativtext: RAID 1+0 RAID O RAIDi RAID i Disk O Disk i Disk 2 Disk 3 | * + Kombination aus RAID 1 und RAID 0   + Zuerst zwei Platten mit RAID 1   + Die so entstandenen Logical Volumes weren via RAID |

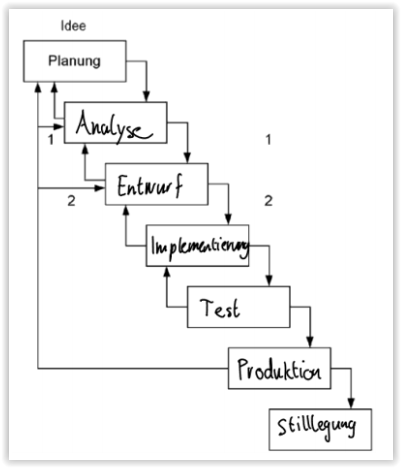
Hot Replacing, Swapping, Plugging -> Wechsel einer Disk im laufenden Betrieb

Host Spare (Sparing) -> Laufendes nicht verwendetes Laufwerk, einsatz bei Ausfall

# Programmentwicklung (Kap. 6, p. 136ff)

## Prozessmodelle:

### Schwergewichtige (klassische)

* + Wasserfall-Modell
    - 
  + V-Modell
    - (ähnlich Wassefall jedoch kongruent v-seitig Testphasen zu jedem Step)
  + Inkrementelle und iterative Modelle
    - Spiralmodell
    - Rational Unified Process (IBM)
    - Open Unified Process

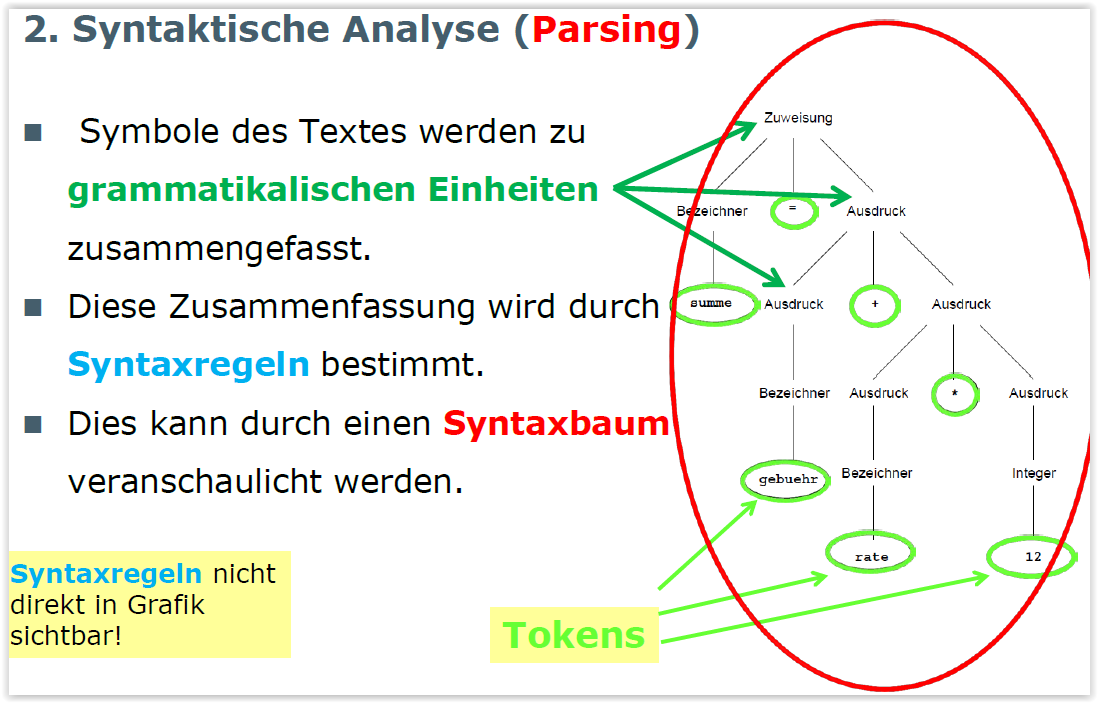
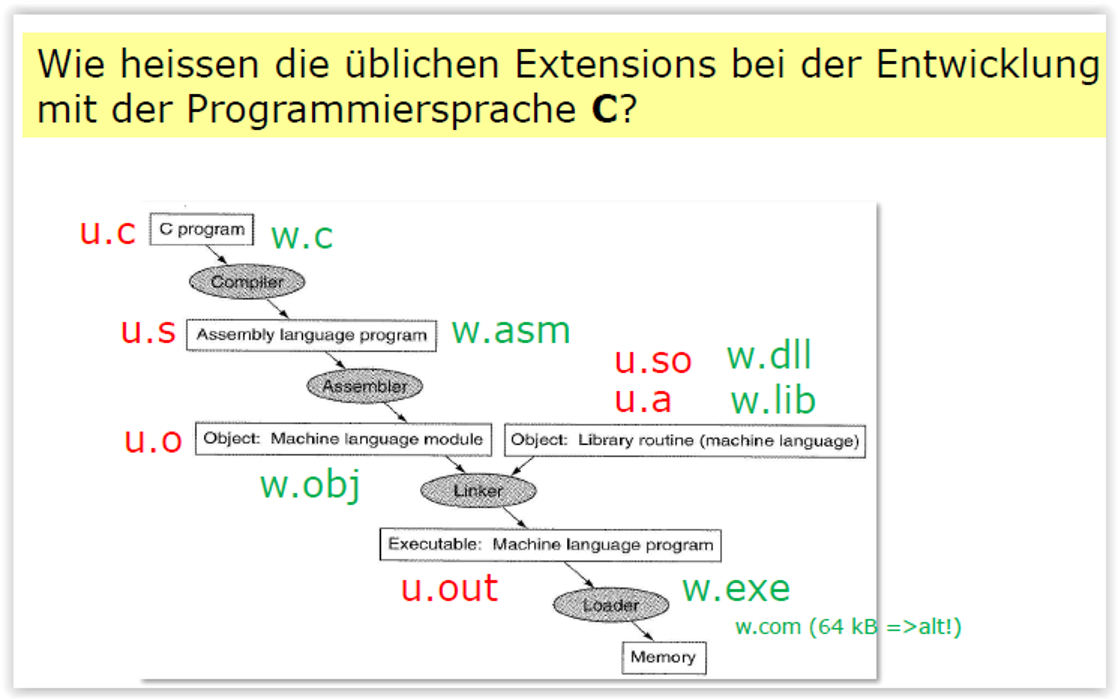
### Leichtgewichtige (Agile)

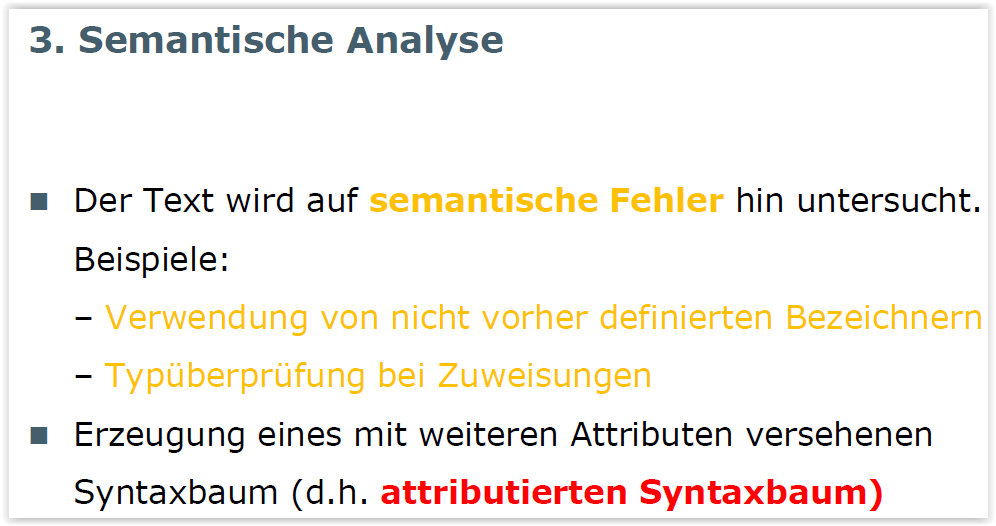
* + Extrem Programming
    - Programmierung zu zweit während eines Zeitraums -> Fazit: zu streng
  + Agile Vorgehensweisen

|  |  |
| --- | --- |
| Scrum | Im Mittelpunkt von Scrum steht das selbstorganisierte Entwicklerteam, das ohne Projektleiter  auskommt. Um dem Team eine störungsfreie Arbeit zu ermöglichen, gibt es den ScrumMaster, der  als Methodenfachmann dafür sorgt, dass der Entwicklungsprozess nicht zerbricht.  Der ScrumMaster stellt auch die Schnittstelle zum Produktverantwortlichen (Product-Owner) dar,  dem die Aufgabe zukommt, Anforderungen zu definieren, zu priorisieren und auch zu tauschen.  Allerdings ist in Scrum klar geregelt, wann der Produktverantwortliche neue oder geänderte  Anforderungen beauftragen darf - so gibt es ungestörte Entwicklungszyklen von 2-4 Wochen  (Sprints), in denen ihm untersagt ist, das Entwicklerteam zu "stören". Während eines Sprints wird  deshalb der Product Owner seine Vorstellungen von der weiteren Entwicklung ins Product Backlog  eintragen und so für kommende Sprints einplanen. |
| Kanban | Kanban ist eine agile Methode für evolutionäres Change Management. Das bedeutet, dass der bestehende Prozess in kleinen Schritte (evolutionär) verbessert wird.  Indem viele kleine Änderungen durchgeführt werden (anstatt einer großen), wird das  Risiko für jede einzelne Maßnahme reduziert.  Darüber hinaus führt der eher sanfte Stil von Kanban in der Regel zu weniger  Widerständen bei den Beteiligten.  **So funktioniert Kanban:**  Der erste Schritt bei der Einführung von Kanban besteht darin, den bestehenden Workflow, die  vorhandene Arbeit sowie Probleme zu visualisieren.  Dies wird in Form eines Kanban-Boards getan, das z.B. aus einem einfachen Whiteboard und  Haftnotizen oder Karteikarten besteht. Jede Karte auf dem Board repräsentiert dabei eine Aufgabe. |

## Compilieren und Interpretieren

|  |  |
| --- | --- |
| Compiler | Interpreter |
| * + Übersetzt alles auf einmal   + Keine live Fehlerbehebung-   + Programm läuft schnell | * + Übersetzt Anweisung für Anweisung   + Ermöglicht live Fehlerbehebung   + Läuft langsam |





# Programmiersprachen, Algorithmen (Kap.7/8, p. 148ff)

## Prozedurale Programmierung

* Wird am nächsten mit Programmierung in Verbindung gesetzt
* Quellcode wird von oben nach unten gelesen -> Liste von Instruktionen
* Sprachen: C, C++ und Java -> nicht nur OOP sondern auch prozedural
* Sind immer auch imperative Programmiersprachen
* -> Befehl pro Linie an Compiler○ Anweisung
* Anweisung kann man als Zustand des Programmes werten, z.B. Variable wird
* inkrementiert
* Imperative Programmierung beschreibt wie etwas berechnet werden soll
* Maschinencode

## Funktionale Programmierung

* Funktion im mathematischen Sine
* Zustandslos -> Eingabeparameter verändern Zustand des Programmes nicht
* Funktion kann ihren eigenen Zustand verändern -> concurrency, mehrere Instanzen laufen unabhängig
* Funktionen können andere Funktionen aufrufen (Rekursion)
* -> Entwicklung von Funktionen
* Funktionale Programmiersprachen sind immer deklarative Programmiersprachen
* -> Deklarative beschreibt was berechnet werden Soll
* SQL z.B. beschreibt das Resultat, der Lösungsweg mach die Programmiersprache
  + Beispiele: Haskell, LISP oder Erlang -> beliebt in akademischen Krisen als Mittel der
* Mathematik

Prozedurale Sprache = Beschreibt wie etwas berechnet werden soll (C, C++, java, php)

Funktionale Sprache = Beschreibt was berechnet werden muss. Man beschreibt das Resultat (haskell, lisp, erlang, java8)

Computergenerierter Alternativtext:
Entwicklungsparadigmen: Klassisch
A iwend’ ng
4
AB- Compiler
f 1P la tif
1
Maschienencode
V
I Iattfor ,
Betriebssystem XY
Quelle in Sprache AB (z.B. Cobol, Pascal, C)
..... 
HardwareComputergenerierter Alternativtext:
Entwicklungsparadigmen: In JEE
- WeIt
Java Quellcode (hello.java)
Ar wend ing
Zomii
Java Bytecode (hello.class)
1 
Patifor r
4
Virtuelle Java Masc
(ip
\r Jf2
‘
hine (JVM)
F
[d LJ 
Maschinencode
,
J Betriebsystem
Hardware

Computergenerierter Alternativtext:
Entwicklungsparadigmen: In .NET’
0d Quellcode J# Pascal Andere
A dwe dung
____ Common Intermediate Language (CLI)
Common Language Runtime
Çoniji3ç)
PI
M
Maschinencode
4
ttfori
cros ft
Windows (Desktop und Mobile)
Hardware.

Computergenerierter Alternativtext:
i konsrar Jede Anweisung eines Programms wird höchstens einmal ausgeführt.
Dies ist der ldealzustarid für einen Algorithmus.
log n iogarilhmisch Speicher- oder Zeitverbrauch wachsen nur mit der Problemgrö& n.
Die Basis des Logarithmus wird häufig 2 sein, d. h. vierfache Daten-
menge verursacht doppelten Ressourcenverbrauch, 8-fache Daten-
menge verursacht 3-fahen Verbrauch und i 024-fache Datenmenge
10-fachen Verbrauch.
n linear Speicher- oder Zeitverbrauch wachsen direkt proportional mit der
Problemgröl3e ri.
n log n n log n Der Ressourcenverbrauch liegt zwischen n (linear) und ri2 (quadra
tisch).
n2 quadratisch Speicher- oder Zeitverbrauch wachsen quadratisch mit der Problem
qröß. Solche Alorithmon lscn sich prkticch nur für klir Pro
bleme anwenden.
n3 kubisch Speicher- oder Zeilverbrauch wachsen kubisch mit der Problemgröl3e.
Solche Algorithmen Issen sich in der Praxis nur für sehr kleine
Problemgröl3en anwenden.
2’ exponentiell Bei doppelte dreifacher und i 0-facher Datenmenge steigt der Res
sourcenverbrauth auf das 4-, 8- bzw. 1 024-fache. Solche Algorithmen
sind praktisch kaum verwendbar.Computergenerierter Alternativtext:
logn)
0(n)
O((ogn)
Eingaberöße (n)

# Datenstrukturen (Kap. 7, p. 208ff, Kap. 8, p. 324ff)

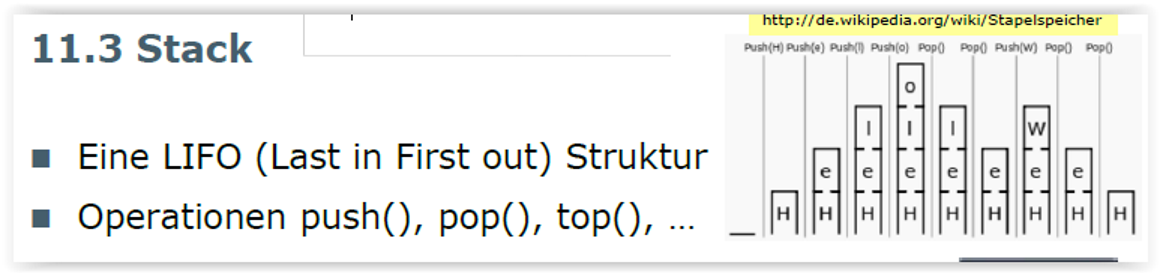


## Sorts:

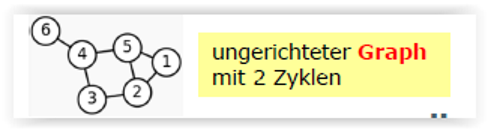
|  |  |
| --- | --- |
| **Bubble:**  Die benachbarten Elemente werden miteinander verglichen ist das linke grösser wird vertauscht. Dies Passiert bis die grösste Zahl am Ende steht. Es beginnt wieder von vorne. Die Letzte Zahl wird nicht mehr beachtet, da sie ja bereits die Grösste/ Kleinste ist. |  |
| **Ripple:**  Ist ähnlich wie Bubble. Jedes Element wird mit dem Element am Ende verglichen und gegebenenfalls getauscht. Ist das Grösste/Kleinste am Schluss angekommen beginnt es von vorne und beachtet das Grösste/Kleinste nicht mehr. |  |
| **Insertion:**  Prinzipiell umgekehrter Bubblesort. Vergleicht Rechtes Element mit linkem Element und somit ist dann die Grösste/Kleinste Zahl am Anfang und wird beim nächsten Durchlauf nicht beachtet. |  |
| **Selection:**  Sucht den Extremwert (Kleinstes/Grösstes Element) und tauscht ihn mit der Zahl an der ersten Stelle (falls Grösser/Kleiner). Beim Zweiten Durchgang wird die Zahl an der ersten Stelle nicht beachtet, da sie die Grösste/Kleinste ist. Es beginnt von vorne. |  |

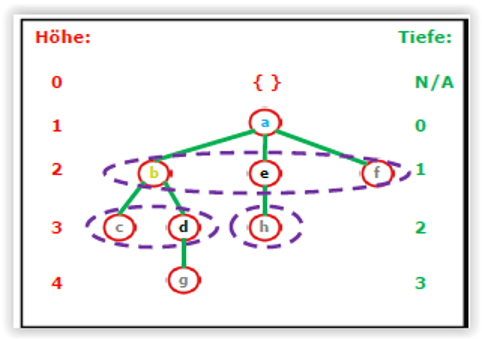
|  |
| --- |
| Shellsort: |
| **S** |
| **Quicksort:**  Die Liste wird in zwei Teile getrennt. Dabei wird beim einfachen Quicksort das rechteste Element als Teilelement gewählt. Innerhalb des Abschnitts wird nun Sortiert und das Teil Element an den Anfang Gerückt. Nun werden die Teile wieder Geteilt und es beginnt von vorne. Dies geschieht so lange bis die Teile 1 oder 0 Elemente beinhalten.    Computergenerierter Alternativtext: o Teile   Teile Teile ... ... |

## Stack



### Bäume





Ein Baum besteht aus Knoten (Nodes), Die Knoten sind durch Kanten/Bögen (edges) verbunden. Die Kanten zeigen vom Elternknoten auf den Kindknoten. Kindknoten ohne eigene Kinder sind sogenannte Leafs, Blatt, Externer oder äusserer Knoten. Knoten die Selber Kinder haben nennt man Innere Knoten. Der Knoten ohne Elternknoten heisst Wurzel. Jeder Knoten ist auch die Wurzel eines Teilbaumes. Der Pfad beschreibt den Weg von einem Knoten zum anderen. Die Länge ist die Anzahl Knoten auf dem Pfad. Direkte Nachfahren eines Elternknotens bilden ein Niveau, befinden sich also auf einer Ebene(Layer). Die Tiefe eines Knotens ist die Anzahl Kanten bis zur Wurzel. Die Höhe eines Baums beschreibt die Anzahl Ebenen. Die Höhe ist = Tiefe +1

Der Knotengrad beschreibt die Anzahl Verzweigungen die vom Knoten abgehen. Grad 1 z.B. ist nichts anders als eine List (Arrays, Linkedlists), Binärbaum = 2 oder weniger Nachfolger also Grad <=2.

Computergenerierter Alternativtext:
Q
o
Q

Desto tiefer ein Baum ist desto schlechter(langsamer) wird die Suche. Bei einem Balancierter Baum sind die Knoten so verteilt, dass die Tiefe/Höhe möglichst klein ist und somit wird die Suche auch viel schneller (Formel für max. Höhe des Baums c\*log(n)[c = unabhängige Konstante, n = die Anzahl der Elemente im Baum]).

Ein Binärbaum ist geordnet wenn jeder innere Knoten en linkes und eventuell ein Rechtes Kind besitzt. Und der Linke Knoten kleiner als der Rechte ist. Der Baum ist voll wenn jeder Knoten entweder zwei oder Keine Knoten hat. Vollständig(Complete) sind volle Bäume bei denen die Blätter die Gleiche Tiefe haben. Bei einem Vollständigen Binärbaum gibt es (2^h) -1 Knoten, (2^(h-1))-1 Innere Knoten, 2^(h-1) Blätter und auf der Höhe h 2^(h-1) Knoten oder auf der Tiefe t 2^t.

**ALV-Baum:**

Uns interessiert vor allem die Höhenbalance

Computergenerierter Alternativtext:
Für ALV-Baum gik: Die Höhe des linken Kindes weicht von der Höhe des rechten
Kindes um höchstens +/- i ab Prüfungsrelevant!
h.

Mit dem logarithmus dualis (Log10 (x) / log10(2)) der Anzahl Knoten kann die Tiefe berechnet werden.

4 Methoden um Binärbäume zu Traversieren (Bei den ersten drei Methoden. Es wird erst geschrieben wenn ich die Wurzel des Teilbaumes erreiche):

|  |  |
| --- | --- |
| Preorder 🡪 WLR |  |
| Inorder 🡪 LWR |  |
| Postorder 🡪 LRW |  |
| Levelorder |  |

### Binäre Suchbäume

Computergenerierter Alternativtext:
Ein binärer Suchbaum hat Grad 2 und jeder Teilbaum
hat folgendes Ordnungsprinzip:
- Er hat eine Wurzel.
Der Inhaltswert des linken Kindes ist
(oder gleich gross).
- Der Inhaltswert des rechten Kindes ist
(oder gleich gross).
Diese “Regel” gilt für alle inneren Knoten!
Ein binärer Suchbaum liefert Inorder ausgelesen eine
korrekte, aufsteigende Sortierung: