InfoSys Zusammenfassung

Autor: Aathavan Theivendram, Kevin Stadelmann, Janik von Rotz

Datum: 12.01.2014

Version: 1.0

Inhaltsverzeichnis

[1. Zahlensysteme 3](#_Toc408864827)

[1.1. Römische (s. 48-49) 3](#_Toc408864828)

[1.2. Positionssysteme (s. 49-55) 4](#_Toc408864829)

[1.2.1. Umrechnungen (s. 56-62) 5](#_Toc408864830)

[1.3. Summenzahl (s. 50/55) 6](#_Toc408864831)

[1.4. Operationen im Dualsystem (s. 62-67) 6](#_Toc408864832)

[1.4.1. Addition (s. 62) 6](#_Toc408864833)

[1.4.2. Subtraktion (s. 63) 7](#_Toc408864834)

[1.4.3. Ganzzahl-Domäne (Wertebereich) 9](#_Toc408864835)

[1.5. Reelle Zahlen (s. 68-70) 9](#_Toc408864836)

[1.6. Zifferfolgen 13](#_Toc408864837)

[2. Codes 14](#_Toc408864838)

[2.1. Zeichensatz vs. Zeichencode 14](#_Toc408864839)

[2.2. ASCII (American Standard for Coded Information Interchange) (s. 71-73) 14](#_Toc408864840)

[2.3. Unicode (s. 74-76) 16](#_Toc408864841)

[2.3.1. UTF (Unicode Transformation Format) (s. 75-76) 17](#_Toc408864842)

[2.4. Big Endian vs. Little Endian 18](#_Toc408864843)

[2.5. BCD-Code (s. 75-76) 19](#_Toc408864844)

[2.6. Duale Grössenangabe (s. 77) 19](#_Toc408864845)

[2.7. Codegewinnung 20](#_Toc408864846)

[2.7.1. Diskretisierung 20](#_Toc408864847)

[2.7.2. Modullierung 21](#_Toc408864848)

[2.8. Fehlertolerante Codierung 25](#_Toc408864849)

[2.8.1. k aus n 25](#_Toc408864850)

[2.8.2. 1-Bit-ECC (nicht hamming optimiert) 25](#_Toc408864851)

[2.8.3. Eindimensionale Parity-Prüfung 26](#_Toc408864852)

[2.8.4. Zweidimensionale Parity-Prüfung 26](#_Toc408864853)

[2.8.5. CRC-Codierung 26](#_Toc408864854)

[2.8.6. Hamming-Code 27](#_Toc408864855)

[3. Boolesche Algebra (s. 82ff) 28](#_Toc408864856)

[3.1. Operatoren (s. 547) 28](#_Toc408864857)

[3.2. Gesetze (s. 86) 29](#_Toc408864858)

[3.3. Normalformen (s. 556) 32](#_Toc408864859)

[3.3.1. Disjunktiv 32](#_Toc408864860)

[3.4. Transistoren (s. 542) 32](#_Toc408864861)

[4. Hardware und Firmware 34](#_Toc408864862)

[4.1. Grundlagen eines Neumann-Rechners (s. 91+) 34](#_Toc408864863)

[4.2. Allgemeinwissen 35](#_Toc408864864)

[4.3. Holzi 36](#_Toc408864865)

[4.4. Speicher 39](#_Toc408864866)

[4.4.1. Entwicklung 39](#_Toc408864867)

[4.4.2. Darstellung (S.123) 39](#_Toc408864868)

[4.4.3. Bios (s. 111) 40](#_Toc408864869)

[4.4.4. Allgemeines zum Speicher 41](#_Toc408864870)

[4.4.5. Speicher im Geschäft 46](#_Toc408864871)

[5. Programm Entwicklung 49](#_Toc408864872)

[5.1. Prozess / Entwicklungsmodelle (s. 529) 49](#_Toc408864873)

[5.1.1. Schwergewichtige (klassische) 49](#_Toc408864874)

[5.1.2. Leichtgewichtige 49](#_Toc408864875)

[5.2. Kompilieren vs. Interpretieren (S. 138) 49](#_Toc408864876)

[5.2.1. Unix vs. WIndows 50](#_Toc408864877)

[5.2.2. Funktionsweise eines Compilers (Java…) 50](#_Toc408864878)

[6. Programmiersprachen und Algorithmen 51](#_Toc408864879)

[6.1. Generationen 51](#_Toc408864880)

[6.2. Prozedurale vs. Funktionale Programmiersprachen 51](#_Toc408864881)

[6.3. Paradigmen 51](#_Toc408864882)

[6.4. Algorithmus 53](#_Toc408864883)

[6.5. Begriffe (s. 152) 53](#_Toc408864884)

[6.6. Darstellung (s. 153-154) 54](#_Toc408864885)

[6.7. O-Notation/Komplexität (s. 380) 55](#_Toc408864886)

[6.8. Rekursive Programmierung (s. 199) 57](#_Toc408864887)

[6.9. Sorts 58](#_Toc408864888)

[8. Bäume (s.352) 61](#_Toc408864889)

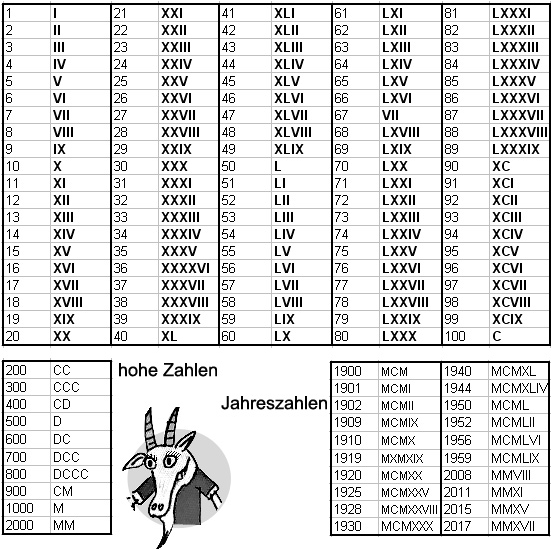
[8.1. Traversieren mit Binärbäumen 63](#_Toc408864890)

[8.2. Stacks (s. 343) 64](#_Toc408864891)

# Zahlensysteme

## Römische (s. 48-49)

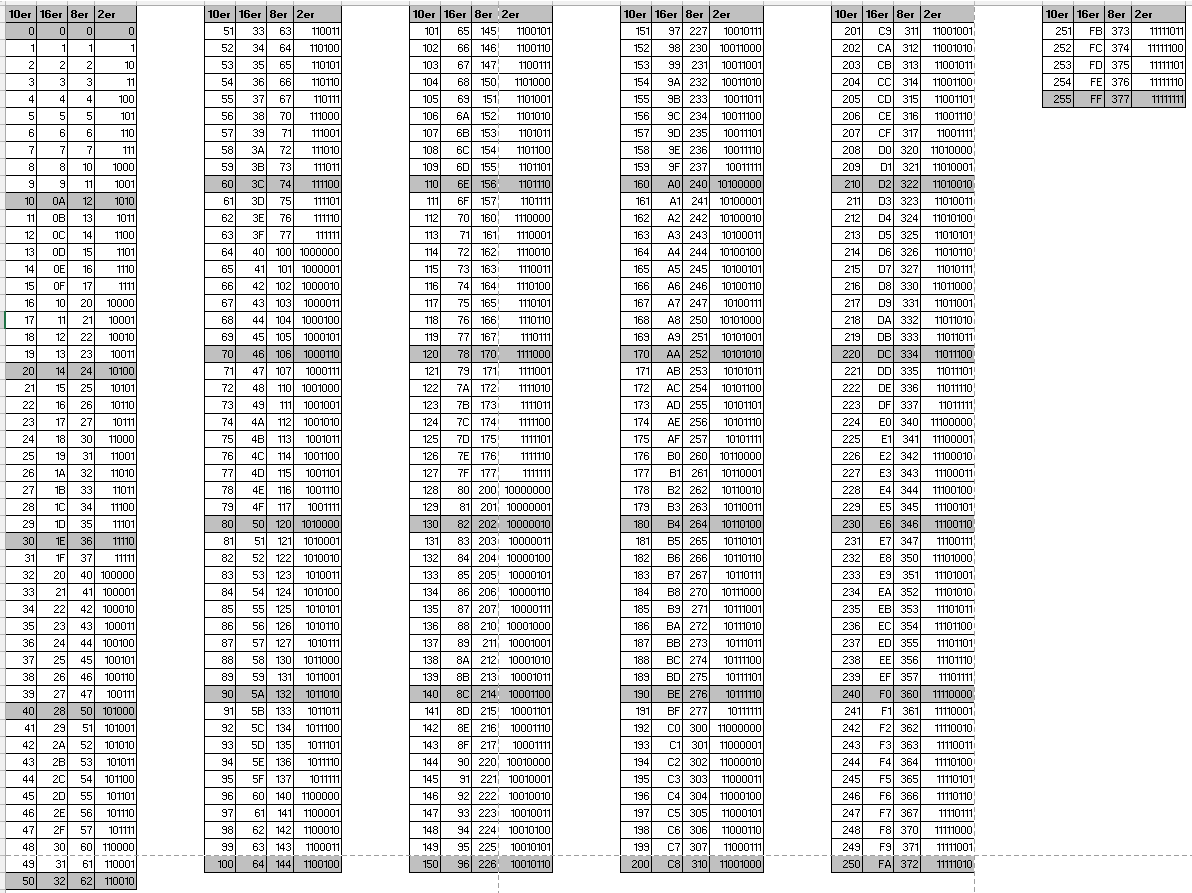
Kam ursprünglich aus Ägypten. Wert einer Zahl wird einfach durch die Form und Anzahl der Zeichen bestimmt. Die Zahl 0 ist nicht abgebildet.



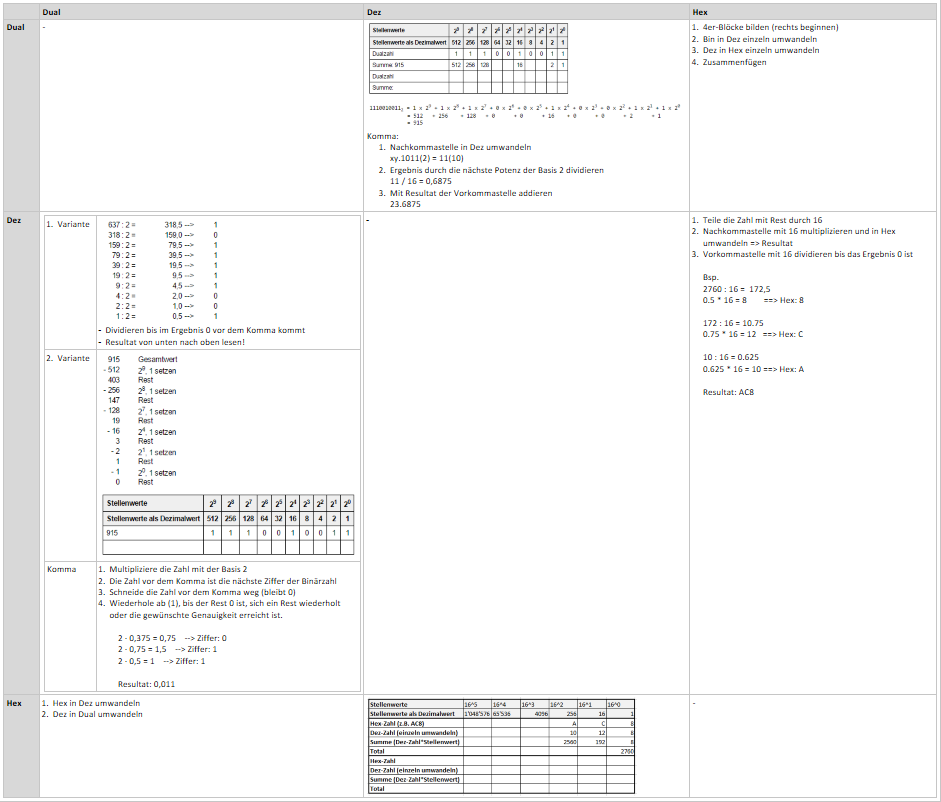
|  |  |
| --- | --- |
| I, X und C | max. 3x |
| M | Beliebig |
| V, L und D | nur 1x |
| Gross/Klein | Addition der Zahlen |
| Klein/Gross | Subtraktion der Zahlen |

## Positionssysteme (s. 49-55)

Wurde von Babylonier, Chinesen, Mayas und Inder genutzt. Wert einer Zahl hängt von der Form und Position der Zeichen ab (Stellenwertsysteme). Im Gegensatz zu römischen kann man einfacher und schneller gerechnet werden.



### Umrechnungen (s. 56-62)



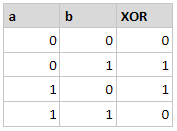
## Summenzahl (s. 50/55)

|  |  |
| --- | --- |
| Natürliche Zahlen |  |
| Gebrochene Zahlen |  |

## Operationen im Dualsystem (s. 62-67)

### Addition (s. 62)

Entspricht der logischen XOR. Wenn bei der letzten Stelle noch ein Rest gibt, wird es einfach beim Resultat vorne angehängt.



|  |  |
| --- | --- |
| Ganze Zahlen |  |
| Gebrochene Zahlen |  |

### Subtraktion (s. 63)

Um mit reinen Addierwerken auszukommen, werden Subtraktionen als Addition dargestellt (5-8 = 5+(-8)). Um dies zu erreichen brauch es eine Komplementbildung mit der entsprechenden Basis (B).

• (B)-Komplement

• (B-1)-Komplement

#### Einer-Komplement (s. 65-66)

Wird in der Praxis wenig gebraucht, weil es eine positive und negative Null gibt. Die negative Zahl kann aber schneller berechnet werden (Symmetrie).

|  |  |
| --- | --- |
| Darstellung |  |
| Berechnung | |  |  | | --- | --- | | Ohne Überlauf   * Wert invertieren |  | | Mit Überlauf   * Wert invertieren * 1 addieren |  | |

#### Zweier-Komplement (s. 63-64)

Wird meistens verwendet, weil es technisch leichter verwendbar ist. Jede Zahl hat genau eine Repräsentation. Das erste Bit ist das MSB (Most significant Bit) und gibt das Vorzeichen an.

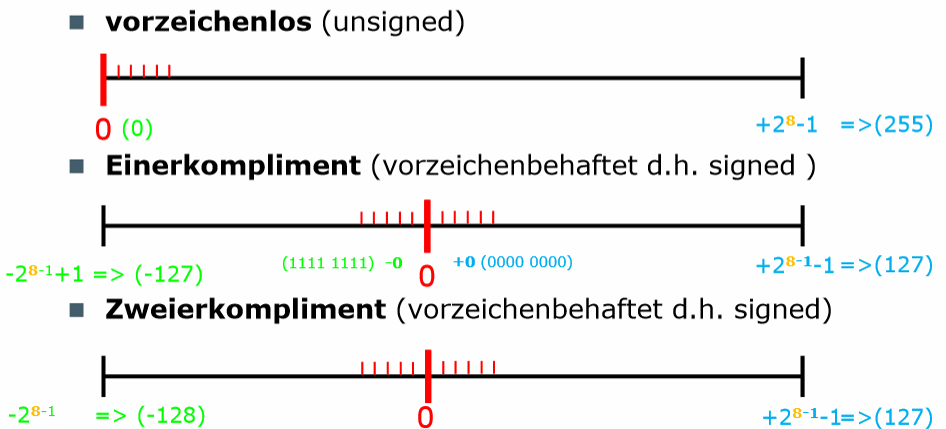
B = Zahlensystem (Bsp. mit 2er)

s = Registergrösse (= Anzahl Bit)

|  |  |
| --- | --- |
| Kleinste darstellbare negative Zahl |  |
| Grösste darstellbare positive Zahl |  |
| Darstellung |  |
| Berechnung |  |
| Ausnahme | Wenn das Ergebnis nicht im darstellbaren Zahlenbereich von der entsprechenden Basis und Registergrösse liegt, entsteht eine falsche Lösung.    Kleinste mögliche Zahl: -2^(5-1) = -16  Grösste mögliche Zahl: 2^(5-1)-1 = 15  Erkenntnis: -9 - 13 = -22 => -22 liegt nicht im Zahlenbereich! |

### Ganzzahl-Domäne (Wertebereich)

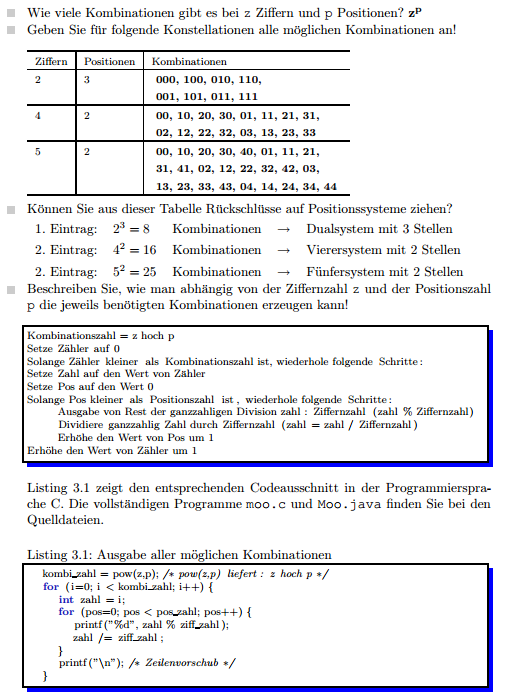
Basis = 2, Registergrösse (Anzahl Bit) = 8



## Reelle Zahlen (s. 68-70)

|  |  |
| --- | --- |
| Festpunktzahl  (Fixed point) | Der Punkt (bzw. Komma) ist an einer festen Stelle und wird bei der Umrechnung einfach mitgenommen. Man kann nur einen beschränkten Wertebereich abdecken. Die Darstellung von ganz grossen Zahlen ist mühsam und deshalb wird es nur in Rechnern mit Spezialanwendung verwendet. |
| Gleitpunktzahl  (Floating point) | Der Punkt kann innerhalt der Zahl verschoben werden. Wird in den meisten Rechnern verwendet, jedoch ist dort die Basis 2.    Anzahl von Bytes legt fest, ob man mit einfacher (float = 32 Bit) oder mit doppelter (double = 64 Bit) Genauigkeit arbeitet. Die Elemente sind nach der Priorität geordnet. |
| IEEE | Darstellung der normalisierten Gleitpunktzahlen   |  |  | | --- | --- | | Dez | Normalisieren heisst das Komma nach der ersten Stelle (x > 0) setzen und die Potenz entsprechend anpassen.     * Punkt verschiebt sich x-Stellen nach **links**: **Exponent + X** * Punkt verschiebt sich x-Stellen nach **rechts**: **Exponent - X** | | Dual | Punkt liegt immer rechts von der ersten 1. Diese 1 wird nie gespeichert, weil es immer gleich ist.    Grund für die Normalisierung | | | BIAS | Falls der Exponent negativ ist, braucht es zusätzlich Platz für das Vorzeichen. Um das zu verhindern, wird der Wertebereich so verschoben, dass es nicht mehr negativ sein kann.    Darstellung:   |  |  |  | | --- | --- | --- | | Vorzeichen (1 Bit) | Exponent (8 oder 11 Bit) | Mantisse (23 oder 52 Bit) |   Berechnung:  Bias + Exponent = neuer Exponent (ohne Vorzeichen)  (Bias float = 127, Bias double = 1023)  **DEZ to Maschinenzahl**  **17.625 -> ?**   |  |  | | --- | --- | | 1. **Binär darstellen** | 10001.1101 \* 20 | | 1. **Normalisierung** | 1.0001101 \* 24 | | 1. **Maschinenzahldarstellung…**   Einfache Genauigkeit (float -> 127)  + Plus Exponent von Normalisierung  = **Biased Exponent** | 0111 1111  0000 0100  **1000 0011** |  * Darstellung: 0 | 1000 0011 | 0001101   Erklärung  0 -> Vorzeichen (positive Zahl)  1000 0011 -> Biased Exponent  0001101 -> Nachkommabereich  **Maschinenzahl to DEZ**  1100 0000 1110 0000 0000 0000 0000 00002  Analyse:   * Vorzeichen ist Minus * Biased Exponent = 10000001 -> 129!   Bias + **Exponent** = 129 🡪 127 + **2** = 129   * Nachkommabereich = 11 🡪 0.75 + 1 = 1.75 (Plus eins weil die eins VOR dem Komma nie dargestellt wird)   Rechnung: 1.75 + 2**2 = -7** (Minus weil Vorzeichenbit gesetzt)    Denormalisierte Zahl:  Bei denormalisiertem Format, wird eine 0 als kleinste Ganzzahl-Stelle angenommen. Damit lässt sich die Lücke zwischen der kleinsten normalisierten Zahl und Null füllen. Denormalisierte Zahlen haben jedoch eine geringere (relative) Genauigkeit als normalisierte Zahlen; die Anzahl der signifikanten Stellen in der Mantisse nimmt zur Null hin ab. | | | Datentyp berechnen | Annahme wir haben eine Dezimalzahl und wollen diese in dualen Datentypen verschiedener Grösse und wechselndem Vorzeichen speichern.  Gegebene Dezimalzahl: 39036  x = Anzahl Bits Datentyp   |  |  | | --- | --- | | **Unsigned** | **Signed** | | 39036 / 2^x = z,y  **z = 0**  dann hat der Wert "platz"  **z > 0**  dann übersteigt der Wert den Speicher  y\*2^x = d ergibt den Dezimalwert der Bits, die die Speicherstelle überschreiben. | 39036 / 2^x-1 = z,y  **z = 0**  dann hat der Werte "platz"  **z > 0**  dann übersteigt der Wert den Speicher   * Ist z ungerade: handelt es sich um einen negativen Wert   y \* 2^(x-1) – 2^(x-1) = d ergibt den negativen Dezimalwert, der sich aus der Überschreibung ergibt.   * Ist z gerade: handelt es sich um einen positiven Wert   y \* 2^(x-1) = d ergibt den positiven Dezimalwert, der sich aus der Überschreibung ergibt. | | | |

## Zifferfolgen



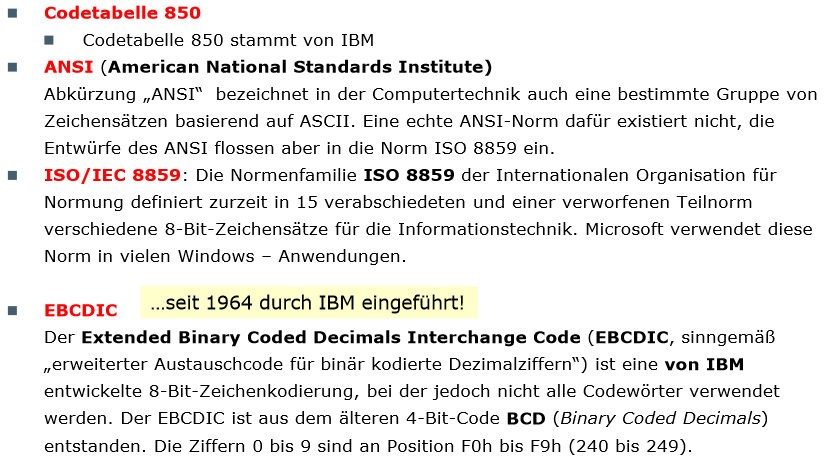
# Codes

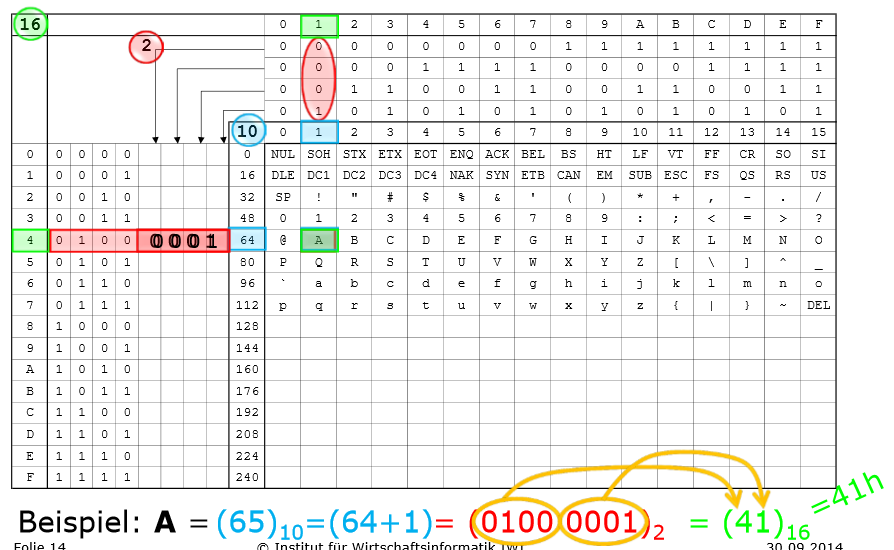
## Zeichensatz vs. Zeichencode

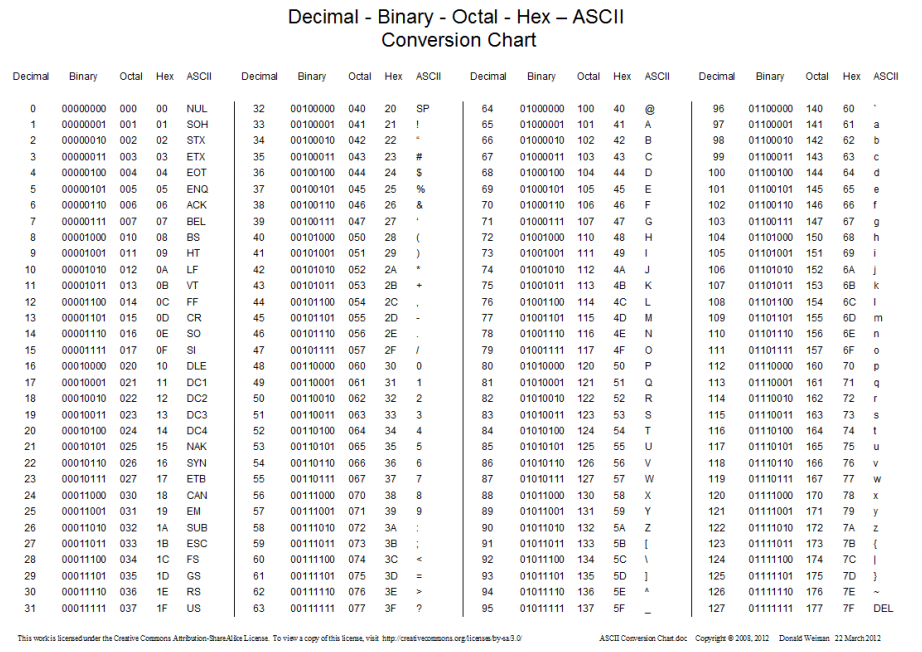
|  |  |
| --- | --- |
| Zeichensatz | Vorrat an Elementen (Buchstaben, Ziffer oder andere Symbole wie Sonderzeichen) zur Darstellung von Sachverhalten. |
| Zeichencode | Enthält zusätzlich noch eine definierte Nummerierung der Schriftzeichen des Zeichensatzes. Dies erlaubt die eindeutige Zuordnung von Schriftzeichen und Symbolen (vom Zeichensatz) zu einem Zahlenwert. Die Zuordnung ist auch gleichzeitig die Kodierung des Zeichensatzes. |

## ASCII (American Standard for Coded Information Interchange) (s. 71-73)

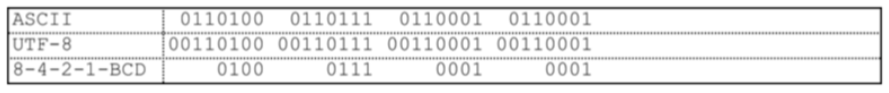
Wurde 1963 durch die ASA (American Standards Association) festgelegt. Besteht aus Klein- und Grossbuchstaben (ohne ä, ö, ü), Ziffern und Sonderzeichen, welche mit 1 Byte (8 Bit) kodiert sind => 256 Zeichen. Hex-Werte bestehen aus Nibbles (Nibble = 4 Bit). Es gibt weitere Codes als ASCII-Erweiterungen (siehe Skript s. 21ff).







Zeichenkette „4711“ umwandeln (ASCII => 7 Bit)



Bei ASCII mit 7 Bit wird das 8. Bit für die Prüfung verwendet. Es gibt heutzutage auch 8 Bit ASCII.

## Unicode (s. 74-76)

Internationaler Standard, in dem für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Wird vom Unicode-Konsortium (Xerox und Apple aus Silicon Valley) seit dem 3. Januar 1990 entwickelt. Die verschiedenen Kodierungen sollen damit vereinheitlicht werden, um die ständigen Umwandlungen zu umgehen. Ist heutzutage Standard für die meisten Betriebssysteme.

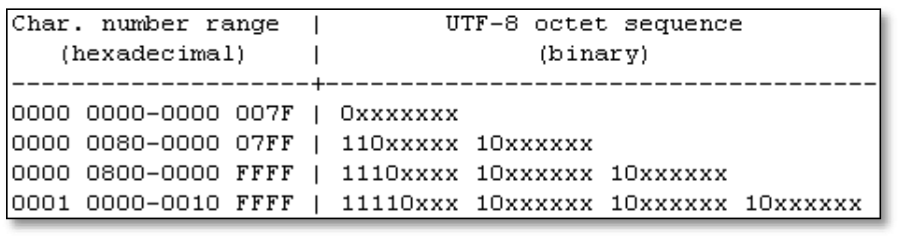
|  |  |
| --- | --- |
| BMP  (Basic Multilingual Plane)   * Zwei-Byte-Schema | Die Zeichen des Unicode sind in Planes organisiert. Es ist eine quadratische Tabelle (256x256 => 2^16 = 65‘536 Felder). Die wichtigsten sind im BMP dargestellt. |
| SMP (Supplementary Multilingual Plane)   * Vier-Byte-Schema | Wurde aufgrund des grossen Wachstums eingeführt. Es soll einfach Platz schaffen für mehr Zeichen => Aktuell gibt es 17 Planes (v6.2)    Unicode war ursprünglich 16-Bit (= 1 Plane) und falls es 32-Bit gewesen wäre, bräuchte man heute nicht so viele Planes. |
| Darstellung | |  |  | | --- | --- | | **Stellen in Hex** | **Bit** | | 4 | 16 | | 6 | 24 | | 8 (wie oben) | 32 |   Führende Nullen können in dieser Schreibung paarweise weggelassen werden. Diakritische sind aus zwei oder mehr Zeichen zusammengesetzt und erhalten entsprechend viele Codepunkte.  Zeichen + Diärese = neues Zeichen (Komposition)  („u“ = U+0075) + („Doppelpunkt“ = U+0308) = ü  Jedoch gibt es für oft verwendete Diakritische Zeichen einen eigenen Codepunkt. |
| Dingbats |  |

### UTF (Unicode Transformation Format) (s. 75-76)

Da die Unicode-Form nicht direkt die Repräsentation der Codepunkte und ihre Zeichen in Bit und Bytes aufzeigt, wurden zur Abbildung von Zeichen in Binärmuster Transformationen entwickelt. Mit UTF können Unicode-Zeichen in Folgen von Bits abgebildet werden. Für die Repräsentation der Unicode-Zeichen zum Zweck der elektronischen Datenverarbeitung gibt es verschiedene Varianten.

* **UTF-8** (für Texte mit vorwiegend lateinischen Buchstaben (häufigste Verwendung)

Bis zu 8 Bits pro Zeichen => bis zu 3 Folgebytes



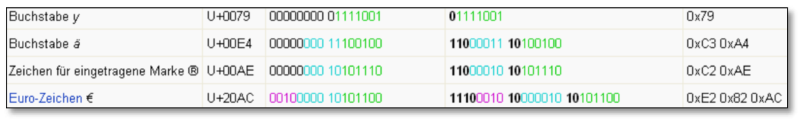
1. UTF-Wert umwandeln (Hex => Dual)
2. Anzahl Bits der binären Zahl < 8 = mit 0 auffüllen und dann die binäre Zahl

Anzahl Bits der binären Zahl < 12 = binäre Zahl mit einem Folgebyte auffüllen

Anzahl Bits der binären Zahl < 17 = binäre Zahl mit zwei Folgebyte auffüllen

Anzahl Bits der binären Zahl < 22 = binäre Zahl mit drei Folgebyte auffüllen

1. Gesamte binäre Zahl zurück in Hex umwandeln und jeweils mit 0x.. darstellen



* **UTF-16** (für Texte mit z.B. asiatischen Zeichen)

Bis zu 16 Bits pro Zeichen => alle Zeichen des BMP lassen sich darstellen

Einige Codepoints der BMP muss die Wortlänge auf 32 Bits erweitert werden => Surrogate Characters

Bsp. mit

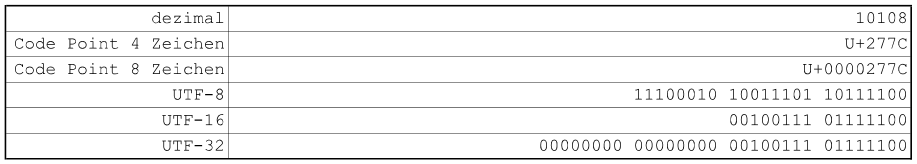
1. Dezimale Zahl in Binäre umwandeln
2. UTF-16 verfügt über 16Bit. Daher muss nun lediglich von vorne aufgefüllt werden bis 16Bit vorhanden sind:  
   **00**100111 01111100

* **UTF-32** (Für höchste Performance bei grösstem Speicherbedarf)

Einfachste Kodierung, welches Codepoints direkt übersetzt werden kann.

Bsp. mit

1. Dezimale Zahl in Binäre umwandeln

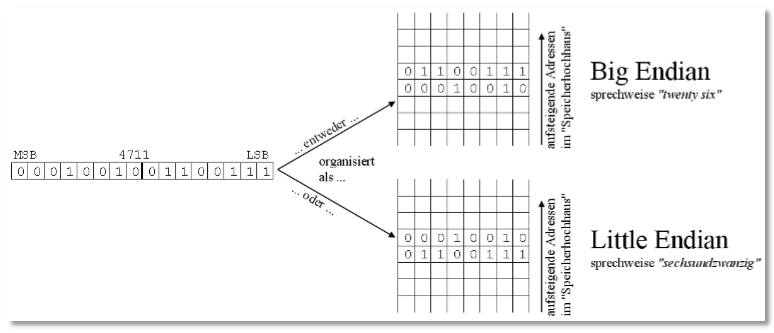


In jeder Variante lassen sich alle Unicode-Zeichen darstellen. Die UTF’s können auch verlustfrei untereinander konvertiert werden. Sie unterscheiden sich u.a. in Speichereffizienz, Laufzeitverhalten und Kompatibilität zu anderen Kodierungen.

## Big Endian vs. Little Endian

Je nachdem ob der Prozessor von unten nach oben oder umgekehrt liest (kein klarer Sieger), aber zum Lesen ist Big Endian besser.

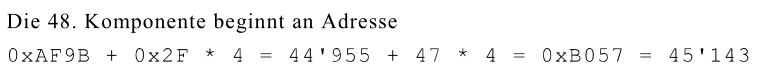
* Windows (Intel CPU) => Little Endian
* Mac (PowerPC) und allgemein Unix (Motorola CPU) => Big Endian



BOM (Byte Order Mark) zeigt an welche Rheinfolge gilt

* FF FE = Little Endian
* FE FF = Big Endian

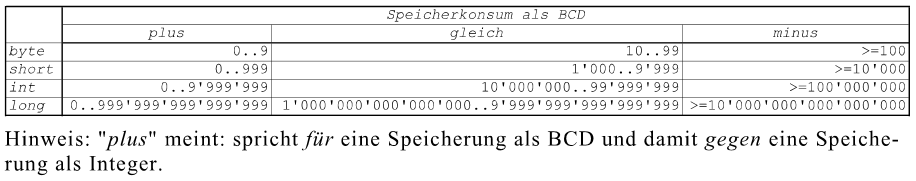
Berechnung Speicherstelle



Startadresse +/- (bei Little Endian aufzählend und Big Endian abzählend) Anzahl Stellen (bei 48 Stellen, 47 weil es bei 0 anfängt) \* Datentyp (short = 4, int = 8, long = 16)

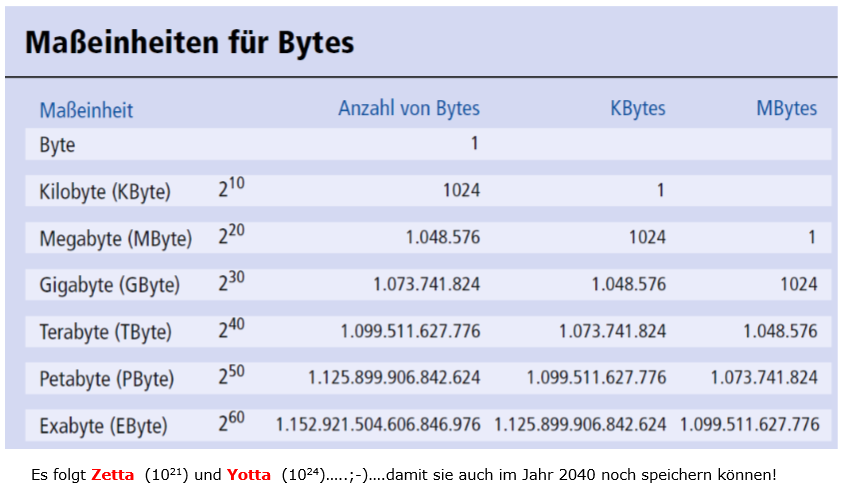
## BCD-Code (s. 75-76)

Wird für Ansteuerung von LCD-Anzeigen (z.B. SBB-Schilder) verwendet.



|  |  |
| --- | --- |
| Byte | 8 Bit / 4 = 2 => nur 2-stellige Zahlen sind gleich gross |
| Short | 16 Bit / 4 = 4 => nur 4-stellige Zahlen sind gleich gross |
| Int | 32 Bit / 4 = 8 => nur 8-stellige Zahlen sind gleich gross |
| Long | 64 Bit / 4 = 16 => nur 16-stellige Zahlen sind gleich gross |

## Duale Grössenangabe (s. 77)



|  |  |  |
| --- | --- | --- |
| **Datentyp-Bezeichnung** | **Bitzahl** | **Wertebereich** |
| signed **byte** | 8 |  |
| unsigned **byte (!)** | 8 |  |
| signed **char** | 8 |  |
| unsigned **char (!)** | 8 |  |
| signed **short** | 16 |  |
| unsigned **short (!)** | 16 |  |
| signed **int** | 32 |  |
| unsigned **int (!)** | 32 |  |
| signed **long** | 32 |  |
| unsigned **long (!)** | 32 |  |
| **float** | 32 |  |
| **double** | 64 |  |
| **long double** | 96 |  |

## Codegewinnung

Beim EVA-Prinzip entsteht immer ein Verarbeitungsverlust bei der Digitalisierung.

### Diskretisierung

|  |  |
| --- | --- |
| Diskretisierung und Digitalisierung | * Klassische Live-Konzerte sind digital nie in ihrer Tonqualität zu erreichen, aber die Digitalisierung kann Fehler korrigieren * Diskretisierung ist immer und unwiederbringlich mit Informationsverlust verbunden * Digitalisierung setzt immer eine Diskretisierung voraus * Klangvolumen einer Schallplatte ist besser, aber die CD überspielt Kratzer besser, weil es redudant speichert * Es gibt Musiker, die den Unterschied zwischen digitaler und analoger Aufnahme merken |
| Begriffe | kontinuierlich = analog, stufenlos (Fiebermesser mit Quecksilber)  diskret = digital, feste Stufen (digitaler Fiebermesser) |
| Diskretisierung eines kontinuierlichen Signals | Analoges Singal wird regelmässig gemessen z.B. alle 0.2ms. Der gemessene Wert wird konsequent auf die darunter liegende diskrete Stufe abgerundet.    Umso höher die Scanstufe (Sample-Rate), desto genauer ist die Digitalisierung. |
| Diskretisierung in Binär-Form |  |

### Modullierung

|  |  |
| --- | --- |
| Transport und Speicherung | Sobald die Bit-Übertragungsfolge definiert ist, können die Bits in elektrische Pegel umgeformt werden. |
| Modullierungsverfahren | |  |  | | --- | --- | | Manchester Codierung |  | |
| Harmonische Schwingung | Zeitlicher Verlauf kann durch eine Sinusfunktion beschrieben werden.     |  |  | | --- | --- | | Amplitude (y0) | Die maximale Auslenkung einer sinusförmigen Wechselgröße | | Periodendauer (T) | * von x1 nach x2 * von max zu max (Beispiel oben) * von min zu min | | Frequenz (f) | Kehrwert der Periodendauer (f = 1/T) | | Hertz (Hz) | Einheit der Frequenz (1 Hz = 1^s-1 = 1 \* 1/s)  1Hz = 1 Schwingung/Sample pro Sek | |
| Modulation |  |
| Abtastrate/Abtastfrequenz | Die Zeit zwischen zwei Abtastungen bei einer Abtast- und Halteschaltung (S/H). S/H ist eine elektronische Schaltung, welches die Signalprobe aus einer analogen Spannung entnimmt und das abgetastete Spannungswert für einen bestimmten Zeitraum haltet.   * Samples per second (S/s)       AD-Wandler sind elektronische Schaltungen zur Digitalisierung von analogen Signalen. |

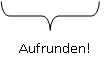
**Aufgabe 1**

*Ein Messsystem, z.B. ein Druckmessung, liefert unaufhörlich und regelmässig Messergebnisse. Diese Werden auf* ***25*** *Stufen abgetastet, sofort diskretisiert, digitalisiert und übertragen.*

* *Das digitale Wort hat die kleinste, theoretische notwendige Breite*
* *Jedem digitalen Wort wird noch* ***1*** *Paritätsbit beigefügt*
* *Zur Übermittlung des digitalen Signals steht Ihnen eine Leitung mit* ***2400*** *bps zur Verfügung*

*Mit welcher Abtastfrequenz in HZ kann höchstens gearbeitet werden?*

Aufrunden!



*Wie viele Signaldrähte hätte ein Parallelkabel, welches die digitalen Ergebnisse „abführt“?*

6 (besser wäre: 6+1, für Erde)

*Es sollte nun die Anzahl diskreter Stufen erhöht werden. Die Qualität der Signalleitung bleibt unverändert. Bis auf welche Anzahl diskreter Stufen kann ohne Folgen erhöht werden?*

*Unter den Bedingungen von oben müssen nun noch mehr diskrete Stufen realisiert werden. Mit welchen Massnahmen ist dies möglich? (zwei Möglichkeiten, eine davon riskanter)*

* Reduktion Abtastfrequenz
* Elimination Paritätsbit

**Aufgabe 2**

*Ein kontinuierliches Signal wird alle* ***125******Mikrosekunden*** *(Millionstel) über* ***200 Stufen*** *abgetastet, digitalisiert und ohne Overhead übertragen. Reicht ein Nutzkanal des ISDN-Basisanschlusses mit 64kbps? Welche Übertragungskapazität wäre nötig, wenn jedem Datenwort noch ein Startbit(Protokoll) und ein Paritätsbit (Fehlererkennung) beigefügt würden?*



Aufrunden

***Wie viele Bits brauchen wir?***

**Aufgabe 3**

*Eine CD-ROM* ***650Mb*** *wird mit* ***44.1 kHz*** *auf* ***65‘356 Stufen*** *und in* ***zwei*** *getrennten Kanälen abgetastet. Wie viele Minuten haben Platz?*

Mbits

Anzahl Kanäle:  
Mono: 1  
Stereo: 2

Bit Konsum

**Aufgabe 1**

*Ihr Memory Stick fasst* ***4******GBytes*** *und kann mit* ***FAT16*** *oder FAT32 formatiert werden. Berechnen Sie die minimale Clustergrösse in exakten Vielfachen von 1 Sektor, wenn die Formatierung die ganze Kapazität des Sticks verfügbar macht. Die Sektorengrösse ist* ***1 KByte****.*

***FAT16***

***FAT32***

1 Cluster muss maximal 1 Sektor gross sein also *1 Byte*

FAT 16 = 2hoch 16

**Aufgabe 2**

*Ein Memory Stick wurde mit dem FAT Dateisystem (Auch etwa* ***FAT16*** *genannt) formatiert. Obige Datei würde „Grösse auf Datenträger* ***8.00******KB****“ angeben. Welche (adressierbare) Kapazität hat der Memory Stick im Maximum? Was brächte ein Wechsel zum System FAT 32 oder NTFS: Clustergrössse, max. Anzahl Dateien interne Fragmentierung, externe Fragmentierung?*

* Ein Wechsel bräche eine Schrumpfen der Clustergrösse auf die Sektorgrösse mit sich, da viel mehr Adressen zur Verfügung stehen:
* Anzahl Cluster = Anzahl Sektoren = max. Anzahl (kleine!) Dateien (ich müsste die Sektorengrösse kennen, um eine genaue Angabe machen zu können)
* interne Fragmentierung entschärft,
* externe Fragmentierung tendenziell grösser, da grössere Dateien sich auf mehr Cluster verteilen.

**Aufgabe 3**

*Modellindex das FAT16 formatiert ist mit einer calc.xls-Datei deren erste Speicheradresse 3AC1 ist:*

*Wie viele Cluster kann das Dateisystem maximal verwalten?*

An welcher dezimalen und binären Plattenadresse beginnt die Datei calc.xls

hex = 3AC1 dez = 15‘039 binär = 00111010‘10111111

*Wie viele Sektoren kann die Hardware maximal verwalten? ---> wissen wir nicht*

*An welcher dezimalen und binären RAM-Adresse beginnt die Datei calc.xls* ----> wissen wir nicht

*Eine Datei sei inhaltlich 854 Bytes gross. Wie viel Platz belegt sie auf dem Datenträger*? ---> Clustergrösse nicht bekannt, wissen wir nicht

**Aufgabe 4**

*Der Datenträger sei ein Memory Stick mit* ***16 GBytes****, die Sektorengrösse* ***4 Kbytes,******FAT16***

*Wie viele Sektoren umfasst ein Cluster, Wieviele Bytes umfasst ein Cluster?*

Kapazität des Datenträgers:

Clustergrösse in Bytes:

Clustergrösse in Sektoren:

*Eine Datei sei inhaltlich 854 Bytes gross. Wie viel Platz belegt sie auf dem Datenträger?*

854 Byte-Datei belegt: 262'144 (ein Cluster)

*Deine Datei sei inhaltlich 270‘339 Bytes gross. Wie viel Platz belegt sie auf dem Datenträger?*

270'339 Byte-Datei belegt: (270'339 DIV 262'144 = 1 voller Cluster) + 1 intern

fragmentierter Cluster = total 2 Cluster

## Fehlertolerante Codierung

Sie ermöglichen es dem Empfänger zu erkennen (fehlererkennenden Code, auch EDC), ob bei der Übertragung ein Fehler aufgetreten ist und falls ja, diesen eventuell sogar zu korrigieren (fehlerkorrigierenden Code, auch ECC).

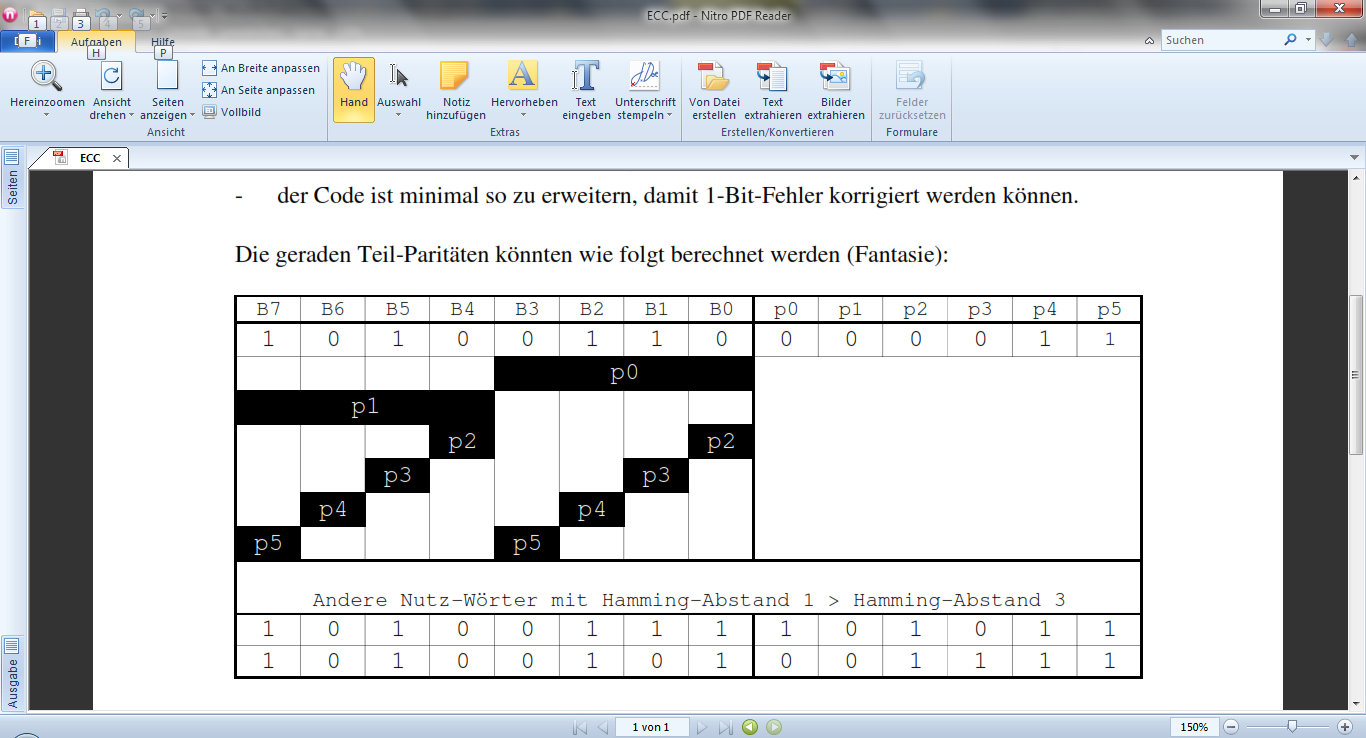
### k aus n

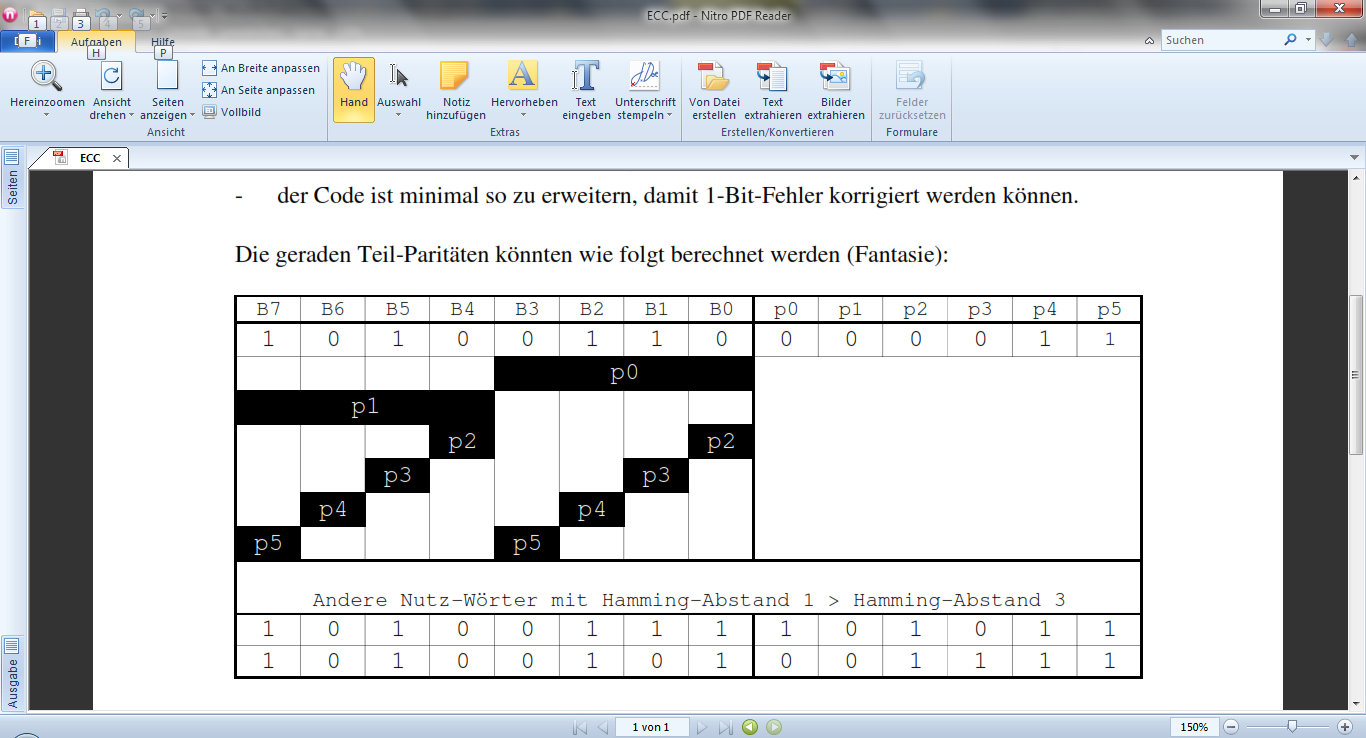
Nicht gesamter möglicher Binärcode wird verwendet, sondern nur eine gegebene Bitanzahl. Fehler können erkannt und allenfalls korrigiert werden.

n = wie viele Bits in Codewort, k = wie viele Bits jeweils gesetzt sind (z.B. 2 aus 5)

### 1-Bit-ECC (nicht hamming optimiert)

Nachteil: Gesendete Zeichenfolgen müssen eine bestimmte Länge haben





Für B4 wird Kontrolle gemacht mit:

### Eindimensionale Parity-Prüfung

Um aus einem nicht fehlertoleranten Code einen 1-fehlererkennenden Code zu erhalten wird ein Parity-Bit hinzugefügt, dass eine 1 ist, wenn das Codewort eine ungerade Anzahl von Einsen besitzt. Bei einer geraden Anzahl von Einser-Bits im Codewort ist das Parity-Bit 0.

**Bsp**.

|  |  |  |  |
| --- | --- | --- | --- |
| A 1000001**0** | C 1000011**1** | W 1010111**1** | Z 1011010**0** |

**Even und Odd Parität**

Odd: 0110 --> Parität 1

Even: 0110 --> Parität 0

Even (gerade) und Odd (ungerade) beschreiben wie es am Schluss sein soll

### Zweidimensionale Parity-Prüfung

Die zweidimensionale Parity-Prüfung wird bei Übertragung von Blöcken eingesetzt.

* Für jedes Zeichen wir ein eigenes Parity-Bit verwendet
* Nachdem alle Zeichen übertragen wurden, wird ein weiteres Codewort übertragen, das die Parity-Bits zu allen Spalten des Blocks enthält

Bsp.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | **1** | Codewort 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | **1** | Codewort 2 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | **1** | Codewort 3 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **1** | Codewort 4 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **1** | Codewort 5 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **1** | Codewort 6 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **1** | Codewort 7 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **1** | Codewort 8 |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **Spalten –Parity-Bits** |

Ein Code mit zweidimensionaler Parity-Prüfung ist 1-fehlerkorrigierend und 3-fehlererkennend

Bsp. Doppelfehler, der an vier falschen Parity-Bits erkannt wird:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | **1** |  |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | **1** |  |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | **0** | **🡨** |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | **1** |  |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | **0** | **🡨** |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | **0** |  |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | **1** |  |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | **1** |  |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **Spalten –Parity-Bits** |
|  |  |  | **🡩** |  | **🡩** |  |  |  |

### CRC-Codierung

CRC = Cyclic Redundancy Check 🡪 Am häufigsten verwendeter fehlertolerante Code. Wobei ein 16- oder 32Bit-Prüfwert durch Divison des Werts durch einen Divisor gebildet werden soll.

**Dividend (Rahmen) modulo Divisor (Generatorpolynom) = Quotient + Rest**

**Übertragende Daten: Rahmen + Rest mit Grad\_des\_Generatorpolynom\_Anzahl\_minus\_Rest\_Nullen am Anfang.**

Um Dualzahlen zu dividieren ist zur Ermittlung einer Stelle jeweils nur eine Subtraktion notwendig, da man sofort feststellen kann ob der Divisor grösser oder kleiner als der Teildividend ist.

**Beispiel:**

Generatorpolynom (CRC-Polynom): 110101, Grad: 5 (Anzahl Stellen – 1)

Nutzdaten/Rahmen/frame (Übertragende Bitfolge): 11011

Rahmen mit Anhang (Rahmen + 5(Grad) Nullen): 1101100000

Berechnung: „Rahmen mit Anhang“ / Generatorpolynom = Ergebnis

↓  immer mit der ersten gemeinsamen 1 anfangen

1101100000

110101

------

0000110000

110101

------

101 (Rest)

Zu übertragende Daten: **11011**(Rahmen)**00**(5*(Grad)* – 3 *(Stellen vom Rest)* Nullen)**101**(Rest)

= **11011 00 101**

### Hamming-Code

Hamming-Code = 1-fehler korrigierender Code mit einem Hammingabstand von *d* = 3.  
Der einfachste Hamming-Code ist ein (7,4)-Code der eine Läge von 7 Bits hat, wovon allerdings nur 4 Bits Nutzinformationen sind und die restlichen 3 Bits der Fehlerkorrektur dienen.

**Formel gesamte Länge:   
Formel Nutzinformationen:**r = Anz. Korrekturbits

Jedes Parity Bit überprüft eine bestimmte Stelle:

* Parity-Bit an Stelle 1, überprüft die Stellen 1,3,5,7 (Daher 🡪 Alle Dualzahlen die an letzter Stelle eine 1 besitzen: 001, 011, 101, 111)
* Parity-Bit an Stelle 2, überprüft die Stellen 2,3,6,7 (Daher 🡪 Alle Dualzahlen die an vorletzter Stelle eine 1 besitzen: 010, 011, 110, 111)
* Parity-Bit an Stelle 4, überprüft die Stellen 4,5,6,7 (Daher 🡪 Alle Dualzahlen die an drittletzter Stelle eine 1 besitzen: 100, 101, 110, 111)

Tritt nun ein Fehler auf, ergibt die Binärfolge der Auswertung der Parity-Bits die Position an welcher der Fehler aufgetreten ist

**Bsp.** Es soll die folgende Bitfolge übertragen werden 1100110. Bei der Übertragung gibt es einen Fehler dabei kippt das fünfte Bit 🡪 11**1**0110

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | Paritätsbit | wird gewertet als |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | - | 1 | - | 1 | - | 0 | falsch | 1 |
| 1 | 1 | - | - | 1 | 1 | - | richtig | 0 |
| 1 | 1 | 1 | 0 | - | - | - | falsch | 1 |

Schreibt man nun die Bitkombination aus der letzten Spalte von unten nach oben, erhält man die Dualzahl 101 = Dezimal 5. Und genau da im 5. Bit ist der Fehler aufgetreten.

# Boolesche Algebra (s. 82ff)

## Operatoren (s. 547)

|  |  |  |
| --- | --- | --- |
| OR | Computergenerierter Alternativtext: a+b | Computergenerierter Alternativtext: a  0  baORb |
| AND | Computergenerierter Alternativtext: ab | Computergenerierter Alternativtext: a b a ANDI)  11 |
| NOT | Computergenerierter Alternativtext: | Computergenerierter Alternativtext: |
| NAND | Computergenerierter Alternativtext: AND | Computergenerierter Alternativtext: NAND |
| NOR | Computergenerierter Alternativtext: NOR  a+b | Computergenerierter Alternativtext: b  NOR |

Ausdrücke zeichnen

|  |  |
| --- | --- |
| Computergenerierter Alternativtext: | Computergenerierter Alternativtext: |
| Computergenerierter Alternativtext: | Computergenerierter Alternativtext: |
| Computergenerierter Alternativtext: | Computergenerierter Alternativtext: OR  AND  Nur mit NAND-Gatters! |

## Gesetze (s. 86)

|  |  |
| --- | --- |
| De Morgan | Computergenerierter Alternativtext: ab ä + b |
| Verschmelzungsgesetze | Computergenerierter Alternativtext:  Computergenerierter Alternativtext: a(ä + b)  ab |

Beweis für die Gesetze (2 Beispiele):

a\*b(invers) = a(invers) + b(invers) => beweisen

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | **a(invers)** | **b(invers)** | **a(invers) + b(invers)** | **a\*b** | **a\*b(invers)** |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |

Gleichung stimmt => w.z.b.w! (was zu beweisen war)

a + a(invers)\*b = a + b => beweisen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a** | **b** | **a(invers)** | **a(invers)\*b** | **a + b** | **a + a(invers)\*b** |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |

Gleichung stimmt => w.z.b.w!

**Sätze und Rechenregeln**

Computergenerierter Alternativtext:
XI,X2 0,1 1,0 1,1 
Zweistellige Boolesche Funktionen 
Funktion 
Name 
IXI, X2J 
XI X2 
XI ex-u 
X21 
XI 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
O 
XI 
XI 
XI 
(XI 
Kontradiktion, Nulltunktion 
Konjunktion, AND(XI, x2) 
Inhibition von XI 
Identität von XI 
Inhibition von x2 
Identität von x2 
Antivalenz, Alternative, XOR(XI, x2) 
Disjunktion, OR(XI, x2) 
Nihilition, Peirce-Funktion, NOR(XI, x2) 
Äquivalenz XNOR (XI,X2) 
Negation von x2, NOT(X2) 
Replikation 
Negation von XI, NOT(XI) 
Implikation 
Exklusion, Shetter-Funktion, NAND(XI, x2) 
Tautologie, Einstunktion 
XI 
XI 
XI IX2 
X21 
XI X2 
IXI, X2J 

Computergenerierter Alternativtext:
Grundgesetze: 
Komplemente: 
De Morgan: 
Korn mutativ-Gesetz: 
Distributiv-Gesetz: 
Assoziativ-Gesetz: 
Absorption: 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 
17. 
18. 
19. 
AAA- 
A 
AVA- 
A 
A 
AVI-l 
AVO- 
A 
o 
AAA- 
AVA-l 
AVB 
AA B 
BAA 
BVA 
AVB- 
- (AA B) v (AA C) 
AA (BV C) - 
(AAB)AC 
(AvB)vC 
A 
AV(AvB): 
A Und:

C:\Users\Aathavan\AppData\Local\Temp\msohtmlclip1\02\clip_image001.png

Oder:

C:\Users\Aathavan\AppData\Local\Temp\msohtmlclip1\02\clip_image002.png

|  |  |
| --- | --- |
| Mal => Plus | Computergenerierter Alternativtext: a*b —a + b  a*b —a + b |
| Plus => Mal | Computergenerierter Alternativtext: |
| Substituieren | Computergenerierter Alternativtext: bedeutet ist definiert durch  a*b  a*b |

1. **Satz**

Darstellung mit Negation, Konjunktion und Disjunktion

1. **Satz**

Darstellung mit (Negation & Konjunktion) oder (Negation & Disjunktion)

Bsp.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Normale Darstellung | Computergenerierter Alternativtext: ab + ab | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | a | b | /a | /b | /a \* b | a \* /b | (/a \* b) + (a \* /b) | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | 1 | 0 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| Negation & Disjunktion (ODER, Plus) | Computergenerierter Alternativtext: | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | a | b | /a | /b | /a + b | a + /b | /(/a + b) | /(a + /b) | /(/a + b) + /(a + /b) | | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | |
| Negation & Konjunktion (UND, Mal) | Computergenerierter Alternativtext: | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | a | b | /a | /b | /a \* b | a \* /b | /(/a \* b) | /(a \* /b) | /(/a \* b) \* /(a \* /b) | /(/(/a \* b) \* /(a \* /b)) | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | |

1. **Satz**

Darstellung mit NAND oder NOR

## Normalformen (s. 556)

### Disjunktiv

Bei drei Variablen => 2^3 = 8 Möglichkeiten bzw. Signale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stock** | **a** | **b** | **c** | **Beispiel mit D** |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |

1. Elemente mit Ergebnis 1 aufschreiben

s = /(abc) + a/b/c + a/bc + ab/c + abc

1. Ähnliche Terme vereinfachen

s = /(abc) + a/b(/c + c) + ab(/c + c)

s = /(abc) + a/b + ab

s = /(abc) + a(/b + b)

s = /(abc) + a

## Transistoren (s. 542)

**Definition**

Schalter ohne Mechanik => früher mit Relais (funktioniert mit Magneten und von hier kommt auch das Wort debuggen, weil dort manchmal Wanzen zwischen den Relais sich eingenistet hatten)

**Funktionsweise**

Silizium wird mit Fremdatomen dotiert.



|  |  |
| --- | --- |
| B | Basisstrom |
| C | Collector |
| E | Emitter |

Mit dem Basisstrom kann die Grösse des Collectorstroms bestimmt werden. Umso mehr der Basiskanal geöffnet wird, desto mehr Wasser fliesst vom Collector zum Emitter.

|  |  |
| --- | --- |
| NPN | Pfeil geht vom Basisstrom (B) zum Emitter (E) => nMOS (für uns relevant) |
| PNP | Pfeil geht vom Emitter (E) zum Basisstrom (B) |
| Steuerstrom | Basisstrom |
| Laststrom | Collector + Emitter |

**Schaltungen**

Kombination aus Transistoren. Mit NAND oder NOT kann ein kompletter PC gebaut werden!

=> NOT-Schalter können mit folgenden Transistoren erstellt werden

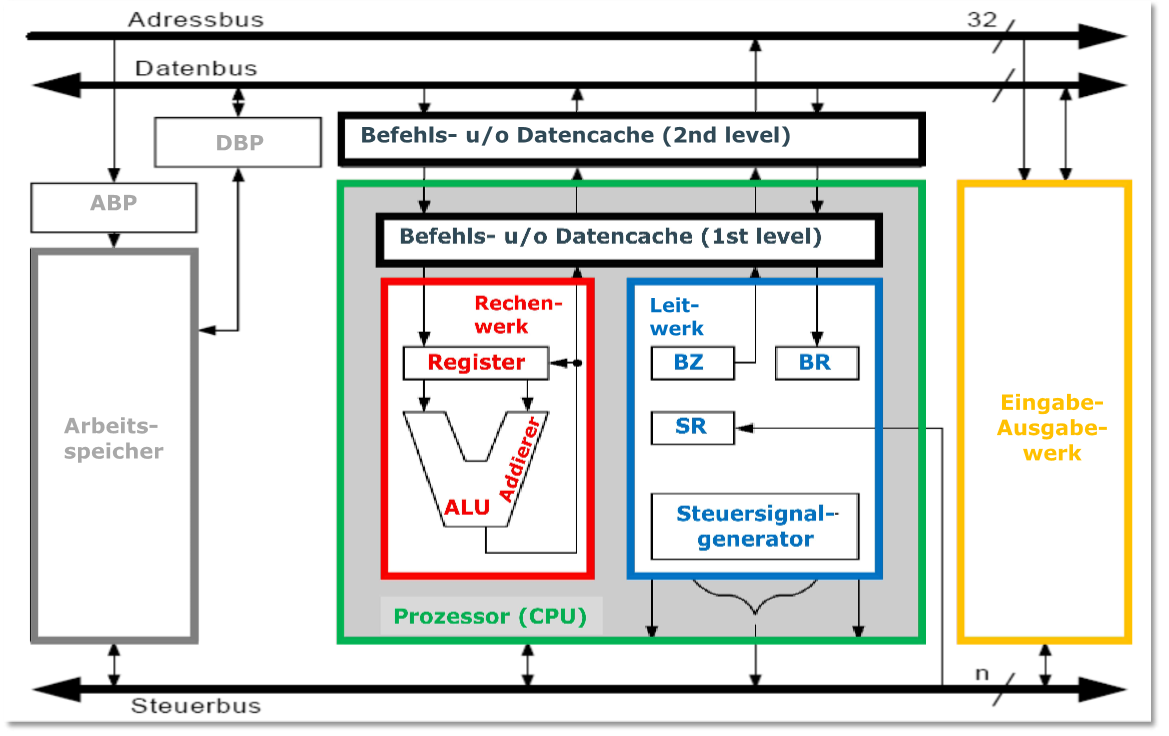
|  |  |
| --- | --- |
| nMOS | keine Ladung = Schalter offen  Computergenerierter Alternativtext: |
| pMOS | keine Ladung = Schalter zu |

Verschiedene Schaltungen

|  |  |
| --- | --- |
| NAND-Gatter | Besteht aus je 2 nMOS und pMOS Transistoren  Computergenerierter Alternativtext: NAM). Gatter  1+5 V  : Masse  pull- up-  Net: werk  Netzwerk |
| AND | Computergenerierter Alternativtext: AND  AND |
| OR | Computergenerierter Alternativtext: |

# Hardware und Firmware

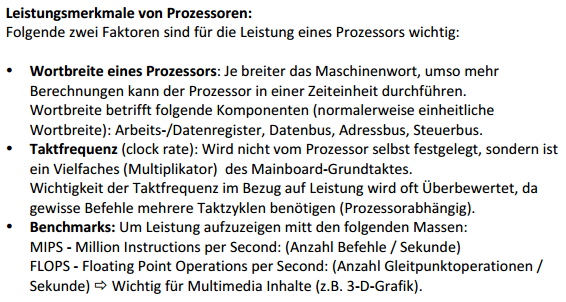
## Grundlagen eines Neumann-Rechners (s. 91+)



DBP = Datenbuspuffer (Transport von z.B. "Hallo“), ABP = Adressbuspuffer (Transport von Hex z.B. A001)

(Unterschied zu Harvard: Harvard verwendet zwei Arbeitsspeicher (Für Daten und Programme getrennt!))

|  |  |
| --- | --- |
| **Prozessor (Central Processing Unit)**  **Besteht aus Steuerwerk (=Leitwerk) + Rechenwerk** | |
| **Rechenwerk** | Besteht aus:   * **ALU** (Arithmetic Logical Unit) welche nur addieren (Ganzzahlen, für Float FPU) und logische Entscheide fällen kann (if a>b) * **Div. Register** welche dem Prozessor als eigene Speicherbereiche dienen (schneller als Cache und spezieller Speicherbereich für flüchtige Inhalte) * **Cache** Ort wo Informationen rasch gespeichert werden (s. 105/636) |
| **Leitwerk / Steuerwerk** | **BZ** = Befehlszähler auch „counter“ genannt, zeigt auf den nächsten Befehl  **BR** = Befehlsregister: „Gango luege“ was der Befehl heisst im Katalog  **SR =** Statusregister: „wie geht’s der Hardware“, wenn z.B. Stackoverflow produziert wird |
| **Speicher (=Arbeitsspeicher, Festwertspeicher)** | Ist wie ein Hochhaus aufgebaut: in jedem Stock hat es Daten    Stackoverflow -> Wenn Daten-Speicher und Programm-Speicher sich berühren…  von Neumann enthält Daten wie auch Programme |
| **Bus** | **Adressbus:** geht nach aussen (nicht geschwindigkeitsmassgebend) => 32 Bit  **Datenbus:** Betriebssystem bezieht sich immer auf den Datenbus (geht in beide Richtungen => Datenautobahn)  -> 32 Bit Windows hat somit einen 32 Bit Datenbus  **Steuerbus:** steuert auf beiden Richtungen (systemspezifisch, Intel, HP,…) |



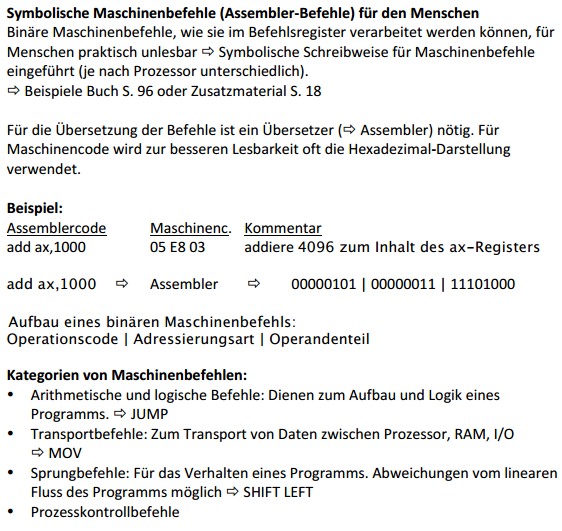
## Allgemeinwissen

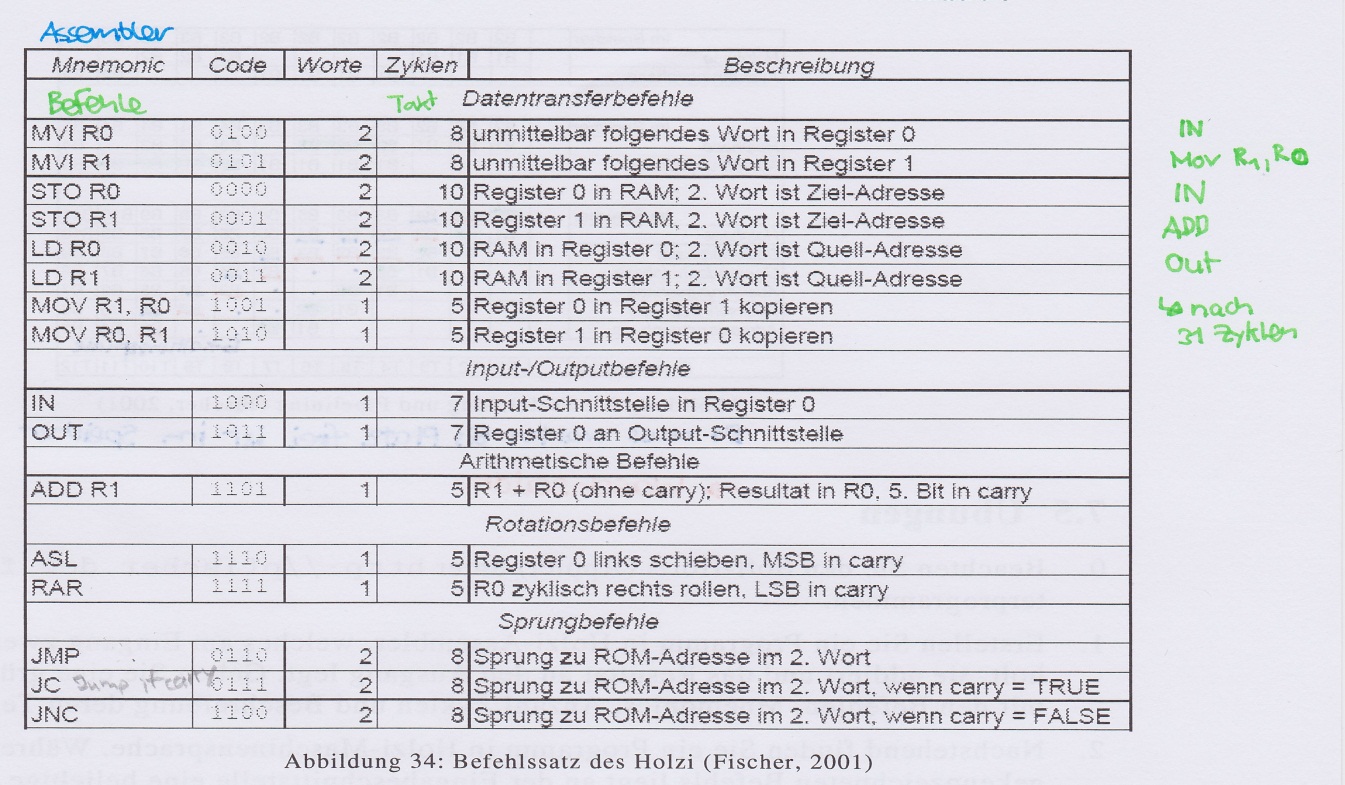
|  |  |
| --- | --- |
| **Grössenangabe** | Computergenerierter Alternativtext: Prefixes for binary multiples  Factor Name  Symbol Origin  Derivation  kilo: (103)1  mega: (103)2  glga. (103)3  tera: (103) 4  peta: (103)5  exa: (103)6  60  kibi  mebl  glbl  tebl  pebi  exbi  Pi  kilobmary: (2 0)  megabmary (210)2  glgabmary (210)3  terabmary: (210)4  petabmary: (210)5  exabmary: (210) 6  Examples and comparisons With SI prefixes  one kibibit  one kilobit  one mebibyte  one megabyte  one gibibyte  one gigabyte  1 Kibit  1 kbit  1 MiB  1 GiB-  210 1024 bit  103 1000 bit  220 B  106 B  109 B  1 048 576 B  1 000000 B  1 073 741 824 B  1 000 000 000 B  MB != Mbyte -> 103 != 210  Festplatte mit 1 GB -> 73 Megabyte weniger als 1Gbyte |
| **Internet-Speed** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Effektive Downloadrate** | Wenn man laut Provider ein 100'000er Netz hat bedeutet das in Wirklichkeit:   |  |  |  | | --- | --- | --- | | 100'000 kbit/s | d.h. 100 Mbit/s | 1 Byte = 8 Bit => 100 Mbit / 8 = 12.5 MB/s | | | **WLAN Downloadrate** | |  |  | | --- | --- | | z.B. 1900 Mbps | d.h. ca. 235 MB/s |     Dazu kommt noch, dass der WLAN Router je nach Frequenz mehr oder weniger Bandbreite hat   |  | | --- | | 1900 Mbit/s nur bei 5 GHz und 600 Mbit/s bei 2.4 GHz | | | **Durchsatz** | 1 GB-LAN ist 6 bis 13 mal schneller als die besten WLAN's | |

|  |  |
| --- | --- |
| **EVA** | Eingabe-Verarbeitung-Ausgabe: Manuelle vs. Maschinelle Informationsverarbeitung: |

## Holzi

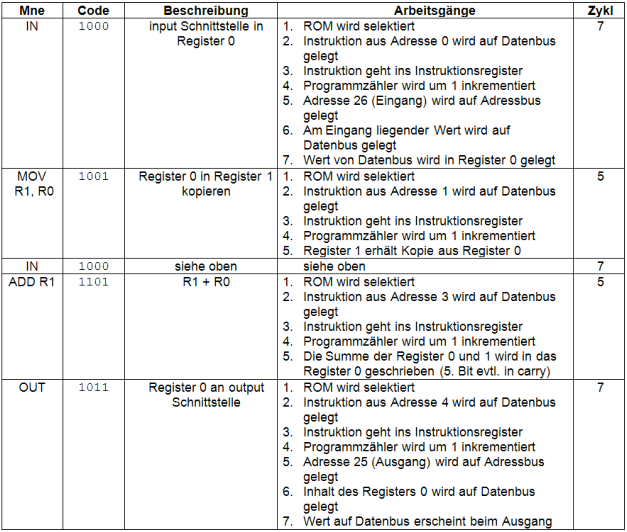






**Aufgabe 1**

*Erstellen Sie ein Programm in Holzi-Assembler, welches am Eingang zwei sSummanden abholt, sie addiert und das Resultat an den Ausgang legt.*



**Aufgabe 2**

*Entwickeln Sie ein Holziprogramm, welches den (ganzzahligen) Mittelwert M zweier einzugebender Zahlen a und b ermittelt: M = (a + b) div 2*

|  |  |  |
| --- | --- | --- |
| IN | 1000 | Inputschnittstelle in Register 0 |
| MOV R1,R0 | 1001 | Register 0 in Register 1 verschieben |
| IN | 1000 | Inputschnittstelle in Register 0 |
| ADD R1 | 1101 | R1 mit R0 addieren |
| RAR | 1111 | zyklisches Rechtsrollen |
| OUT | 1011 | Register 0 an output Schnittstelle versetzen |

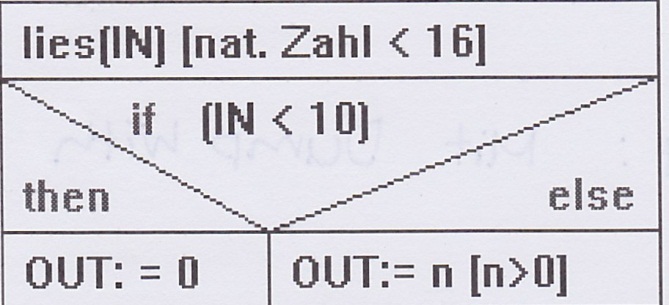
**Aufgabe 3**

*Erstellen Sie ein Programm in Holzi-Assembler, welches bei einer an Input liegenden Zahl untersucht, ob diese gerade sei oder ungerade. Ist die Zahl ungerade, muss an OUT und in RAM 16 der Wert 1111 aufleuchten, sonst je 0000. Das Programm muss im vorhandenen Programmspeicher Platz haben*

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | IN | 1000 | Inputschnittstelle in Register 0 |
| 1 | RAR | 1111 | R0 zyklisch nach rechts rollen, LSB in Carry |
| 2 | JNC | 1100 | 1. Sprung zu ROM-Adresse im , wenn Carry = False = 0  2. Sonst Spring zur ROM-Adresse auf dem nächsten Befehl |
| 3 | ………………..gehört zu JNC………………. | 0110 | Binär: 6 |
| 4 | MVI R1 | 0101 | unmittelbar folgendes Wort in R1 |
| 5 | …………………gehört zu MVI R1…………. | 1111 | Binär: 16 |
| 6 | MOV R0, R1 | 1010 | Register 1 in Register 0 kopieren |
| 7 | OUT | 1011 | Register 0 an Outputschnittstelle |

**Aufgabe 4**

*Folgendes Programm im Holzi darstellen:*

****

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | IN | 1000 | Inputschnittstelle in Register 0 |
| 1 | MVI R1 | 0101 | unmittelbar folgendes Wort in R1 |
| 2 | 0110 | 0110 | Binär 6 |
| 3 | ADD R1 | 1101 | R1 mit R0 addieren |
| 4 | JC | 0111 | 1. Sprung zu ROM-Adresse im , wenn Carry = True = 1  2. Sonst Spring zur ROM-Adresse auf dem nächsten Befehl |
| 5 | ……….gehört zu JC… | 1000 | Binär: 8 |
| 6 | MVI R0 | 0100 | unmittelbar folgendes Wort in R0 |
| 7 | ……….gehört zu MVI R0……….. | 0000 | Binär: 0 |
| 8 | RAR | 1111 | R0 zyklisch nach rechts rollen, LSB in Carry |
|  | OUT | 1011 | Register 0 an Outputschnittstelle |

## Speicher

### Entwicklung

|  |  |  |
| --- | --- | --- |
| **Datenbänder/Tapes (Tertiärspeicher)** | **Plattenmedien/RAMAC (Sekundärspeicher)** | **Arbeitsspeicher/RAM** |
| Werden heute noch als langfristiges  Backup bei Grossunternehmen gebraucht | 1956 von IBM entwickelt  Wurden immer kleiner mit der Zeit |  |
| mehrere Jahre | 25 Tage | 1 Sekunde |

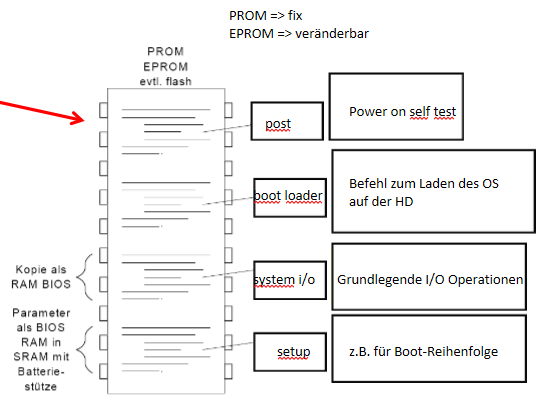
### Darstellung (S.123)

|  |  |
| --- | --- |
| **Darstellung 1** | Computergenerierter Alternativtext: Primärs  elektronisch  volatil (werflü  c 11)  SM Registe-  Ist Level  2nd  High Speed RAM  ReferenzIokaIitÉt  Zwischenspeicher hÉufig benutzter  Befehle und Operanden  RAM  Referenzlokar  Zwischenspeicherung hÉuflS  benutzter Datenbldcke  magnetisch, optisch  persisten t  Tertiär.pe  magnetisch, (optisch  persistent  selten) |
| **Darstellung 2** | Computergenerierter Alternativtext: Frage:  Was ist neu /  zusätzlich?  Frage:  ...und wo sind die  "neuen'  SSD-Disk's?  zwischen Hauptspeicher und Performance Disks  vorheriges Bild beim Plattenspeicher  Arttivspeither :  Analotp Deterr Parier fib |
| **Darstellung 3** | C:\Users\Aathavan\AppData\Local\Temp\msohtmlclip1\02\clip_image027.jpg |
|  |  |

### Bios (s. 111)

Chip auf dem Motherboard, wo die Firmware gespeichert ist

* EFI (Extensible Firmware Interface)
* UEFI (Universal EFI) => ist heute Standard

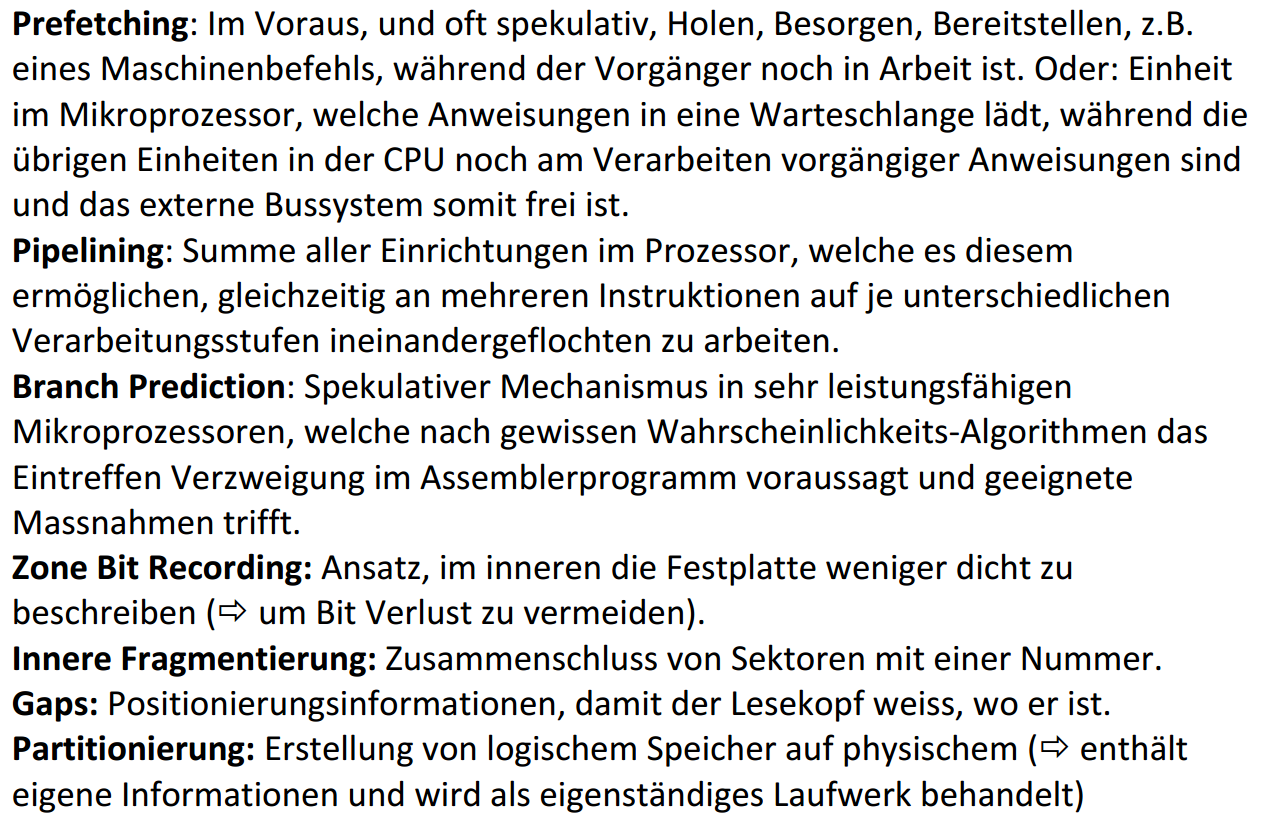


### Allgemeines zum Speicher

|  |
| --- |
| **Polling**  Beim Polling fragt man Signale regelmäßig innerhalb der while-Schleife ab und wertet diese Signale aus. Ein großer Nachteil beim Polling ist allerdings, dass auf Signale nicht sofort reagiert werden kann, sondern erst mit einer Verzögerung.   * Busy waiting * Round robin * selten benutzt |
| **Interrupting**  Bei Interrupts wird sofort reagiert. Wenn ein Interrupts-Signal auftritt, wird der aktuelle Befehl des Hauptprogramms beendet.   * Interrupt requesting * z.B. Tastatureingabe * wird meistens benutzt |
| Dateiverwaltung (s. 119) **Logische Sicht**  Computergenerierter Alternativtext: Beschränkung  des Volumens durch Grösse .  Beschrånkung der  Suchindex  Suchindex verweist auf  Hifsm assnahme  Bücherei  der Bücherei  R ale von "IKEA"  Bucher durch  Anzahl Kärtchen im Karteisc  hrank  Karteischrank  mit Bücherei-spezifischem  Signatursystem  Raum  Regal-Nr.  Regal-Ebene  Bildung von Regalverbünden  wenn zu wenig Karteikarten  Dateisystem  des Datenträgers  Formatierun durch OS  Dateien durch  Breite des Adresswortes  File Allocation Tab e (DOS. Win ex)  Master File able (Win NT/XP)  l-Nodes (Unix, Linux)  andere  Ebene  Spur  Sektor  Bildung von Sektorverbünden (clusters)  wenn binärer Adressvorrat aufgebraucht  **Physische Sicht**  Computergenerierter Alternativtext: fabxeh  »X oca€xoo  vege  -b erexcxn  **HD- Grösse berechnen:**  FAT => 2^28 => 268'435'456 Sektoren (adressierbare Stellen)  Clustergrösse 1: 512 Byte => Nachteil wegen kleinerer Harddisk-Grösse  Clustergrösse 2: 32'768 Byte => Nachteil wegen grossen Verlusten bei kleinen Dateien    Dateisystem-Grösse \* Clustergrösse = Grösse der Festplatte  2^28 \* 512 Byte = 137438953472  2^28 \* 32'768 Byte = 8796093022208    Es ist je nach Speichergrösse sinnvoller eine andere Clustergrösse zu nehmen, damit nicht pro Cluster viel Leerspeicher entsteht.  **Probeprüfung Beispiel**  4 GB Memory Stick -> 4 \* 230  FAT16 = 216 Adressen  FAT32 = 232 Adressen  Sektor Grösse = 1 kByte  Clustergrösse in Bytes:  4 \* 230 / 216 = 65‘536 -> ca. 64kByte  Bzw.  65‘536 / 210 = 64 |

#### Begriffe

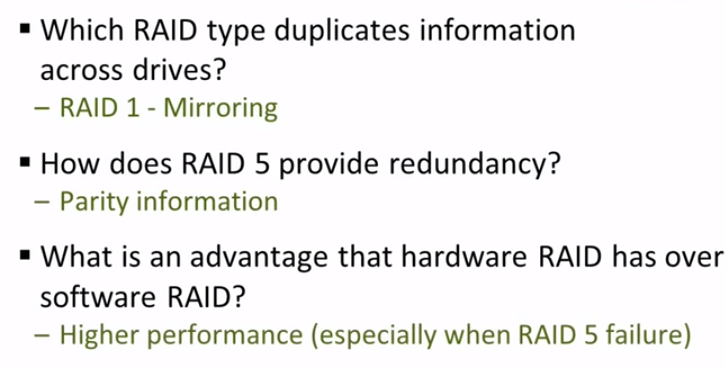
|  |  |
| --- | --- |
| Spur (Track), Sektor, Zylinder, Zone | Die Daten auf einer Festplatte sind in Zylinder, **Tracks (Spuren)** und **Sektoren** organisiert. **Zylinder** nennt man alle konzentrisch übereinanderliegenden Tracks auf den Magnetscheiben. Tracks wiederum beherbergen die einzelnen Sektoren, die normalerweise 512 Bytes fassen.     Jede Zone ist in eine gewisse Anzahl Sektoren unterteilt… |
| Sektorverbund (Cluster) | Müssen nicht immer aneinandergehängt sein, weil die Platte zu schnell dreht und die Nadel damit besser umgeht |
| Low-Level Formatierung | Die physikalische Einteilung eines Speichermediums in Spuren und Sektoren durch den Controller. |
| High-Level Formatierung | Die logische Einteilung der Partitionsstruktur mit einem Dateisystem durch eine Software (meist durch das Betriebssystem). |
| Partitionierung | Die physikalische und logische Einteilung des Speichermediums in zusammenhängende Strukturen. |
| Interleaving-Faktor | Ist eine Optimierungstechnik bei der Datenübertragungoder -speicherung. Dabei werden die Daten in einer bestimmten Reihenfolge angeordnet, um einen höheren Durchsatz zu erreichen. |
| Zone Bit Recording | Ansatz, im inneren die Festplatte weniger dicht zu beschreiben (=> um Bit Verlust zu vermeiden) |
| Volle vs. Schnelle Formatierung | Normalformatierung – Wird auf einem Datenträger eine normale Formatierung durchgeführt, so wird unabhängig vom Dateisystem zunächst eine Suche nach fehlerhaften Sektoren durchgeführt. Diese nimmt den Großteil der Zeit in Anspruch. Anschließend erfolgt das Schreiben der Dateisystem-Metadaten und somit die Löschung der vorhandenen Dateien.  Schnellformatierung – Wird ein Datenträger ebenfalls unabhängig vom Dateisystem mit der Schnell-Methode formatiert, werden zwar die Dateien aus dem Inhaltsverzeichnis des Datenträger beziehungsweise der betreffenden Partition entfernt, die Suche nach fehlerhaften Sektoren wird jedoch ausgelassen. |
| Dateilöschung | |  |  | | --- | --- | | In Papierkorb | Man verschiebt nur die Files in eine andere Speicherstelle. | | Aus Papierkorb | Die Speicherstellen des Files werden gelöscht | |
| Partition | Unter einer Partition versteht man einen zusammenhängenden Teil des Speicherplatzes eines geeigneten physischen oder logischen Datenträgers. Eine Partition ist ihrerseits ein logischer Datenträger, es gibt jedoch noch weitere Arten logischer oder virtueller Datenträger. Man bezeichnet eine Partition daher auch oft als logisches Laufwerk. Der Zugriff auf die gespeicherten Daten obliegt dem Betriebssystem, das die Daten in den meisten Fällen über ein Dateisystem verwaltet. |
| Volume | In the context of computer operating systems, a volume or logical drive is a single accessible storage area with a single file system, typically (though not necessarily) resident on a single partition of a hard disk. |
| Logical Volume Manger | Der Logical Volume Manager (LVM) ist eine hauptsächlich im Unix- und Linux-Umfeld verbreitete Abstraktionsebene zwischen Festplatten, Partitionen und Dateisystemen. Durch den LVM ist es möglich, dynamisch veränderbare Partitionen (Logical Volumes, kurz LV) zu bilden |
| Schnittstellen | |  |  | | --- | --- | | ATA (IDE) |  | | SATA |  | | SCSI | Small Computer System Interface, Erlaubt Anschluss von 7 Geräten. Geräte dürfen nur hintereinander verbunden werden | | iSCSI | Protokoll über TCP | | SAS | Seriel Attached SCSI | | Fibre Channel | FC für SAN | |



#### Raid

|  |  |  |  |
| --- | --- | --- | --- |
| **RAID** | **Beschreibung** | **Vorteile** | **Nachteile** |
| 0 (Striping) | Die Datenblöcke/Cluster werden aufgeteilt  Computergenerierter Alternativtext: RAID O  Al  Disk O  Disk 1 | Paralleles schreiben und lesen | Keine Redundanz => Datenverlust |
| 1 (Mirroring) | Exakte Kopie auf die andere Disk  Computergenerierter Alternativtext: RAID  Al  Disk O  1  Al  Disk 1 | Kein Problem wenn Datenblöcke verloren gehen oder ganze Disk | Benötigt doppelt so viel Speicherplatz, weil als doppelt gespeichert wird |
| 5 (Striping with Parity) | Kombination aus 0 und 1. Schreibt die Daten aufgeteilt auf 2 Disks (wie RAID 0), neu wird auf die 3. Disk mittels XOR die Parity Bit gespeichert. => <https://www.youtube.com/watch?v=LTq4pGZtzho>  Computergenerierter Alternativtext: RAID 5  Disk O  Disk 1  Disk 2 | Redundanz durch die Parity information und höhere Lese-Geschwindigkeit wegen RAID 0 | Benötigt noch mehr Disks bzw. Speicher um die Parity-Bits zu speichern und Parity-Berechnung braucht extra Performance beim Schreiben |
| 01 oder 0 + 1 | 2 RAID 0 HD's (je 2 Disks) werden zu einem RAID 1 zusammengeführt  Computergenerierter Alternativtext: D sk2  D ska  D sko  RAD 0+1  RAID  RAID O  RAID O | Grösserer Datendurchsatz | Weniger Sicherheit, Zeitverlust beim Rekonfigurieren |
| 10 oder 1 + 0 | 2 RAID 1 HD's (je 2 Disks) werden zu einem RAID 0 zusammengeführt  Computergenerierter Alternativtext: RAID 1+0  RAID o | Einfache und schnelle Rekonfiguration (sogar im Hintergrund) => wird eher gebraucht |  |
| 3 | Computergenerierter Alternativtext: Disk O  RAID 3  Disk 3  Disk 1  Disk 2 |  |  |
| JBOD | Just a bound of Disk  Computergenerierter Alternativtext: Disk O  JBOD  A64  A6S  A66  A67  Disk  A92  A93  A94  Ags  Disk 2 |  |  |

|  |  |
| --- | --- |
| Software RAID | OS-Feature, welches keine spezielle Hardware benötigt. Hat aber eine kleinere Performance, weil alles vom Betriebssystem berechnet werden muss. |
| Hardware RAID | HD-Controller-Feature, welches direkt auf dem Controller erstellt wird. OS kennt diesen RAID nicht und schreibt einfach. Viel mehr Performance (vor allem bei RAID 5 Fehler) |



### Speicher im Geschäft

|  |  |
| --- | --- |
| **DAS** - Direct Attached Network  Computergenerierter Alternativtext: System | **NAS** - Network Attached Network  Computergenerierter Alternativtext: ) s über  File  NAS |
| **SAN** Storage Attached Network  Computergenerierter Alternativtext: SAN | **Cloud Storage**  Computergenerierter Alternativtext: Cloud Storage  san |

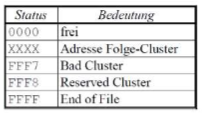
#### SSD Arten

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Art** | **Vorteil** | **Nachteil** | | SLC – Single Level Cell  (1 Bit pro Zelle) | * Sehr schnell (nur Zustand 1 oder 0 zum auslesen) * Lange Lebensdauer | * Sehr teuer * Kleiner Speicherplatz | | MLC – Multi Level Cell  (2 Bit pro Zelle) | * Erlaubt höhere Kapazitäten | * Performance sinkt wegen verschiedenen Vergleichsspannungen | | TLC – Triple Level Cell  (3 Bit pro Zelle) | * Bei gleicher physikalischer grösse wie SLC, 8 fache Speichergrösse * Sehr günstige SSD | * „kurze“ Lebensdauer * Wenn eine Zelle kaputt geht, 8 Bit Datenverlust | |  |

#### Stichworte

|  |  |
| --- | --- |
| Disc | A disc refers to optical media, such as an audio CD, CD-ROM, DVD-ROM, DVD-RAM, or DVD-Video etc. |
| Disk | A disk refers to magnetic media, such as a floppy disk, the disk in your computer's hard drive, an external hard drive. |
| Disk Mirroring | Ein RAID 1 speichert auf allen Festplatten die gleichen Daten (Spiegelung) und bietet somit volle Redundanz. |
| Disk Duplexing | Duplexing ist fast das gleiche wie Mirroring, nur hat jede Festplatte ein eigener Controller |
| Striping | Raid 0 -> Daten können parallel von den Festplatten gelesen werden, da die Daten „in streifen gelegt“ sind: |
| Striping Breite | Streifenbreite bezeichnet die Anzahl von parallelen Streifen , die aus gleichzeitig geschrieben oder gelesen werden können.  Im Printscreen von „Striping“ oben wäre die Breite 2 |
| Striping Granularität | Die Größe der Datenblöcke wird als Striping-Granularität (auch stripe size, chunk size oder interlace size) bezeichnet. Meistens wird bei RAID 0 eine chunk size von 64 kB gewählt. |
| Hot  Replacing /  Swapping /  Plugging | Hot Swapping und Hot bezeichnen den Wechsel und die Wechselbarkeit von Systemkomponenten und Modulen im laufenden Betrieb des Systems.  Hot Swapping bezieht sich auf den Austausch von Komponenten, die nicht mit Software interagieren.  Hot Plugging bezieht sich auf Komponenten, die mit Software (in der Regel dem Betriebssystem) interagieren und meist eine vorinstallierte Unterstützung durch Software benötigen. |
| Hot Spare (Sparing) | Eine Hot-Spare-Festplatte ist eine in einem System in Reserve gehaltene (normalerweise nicht verwendete) [Festplatte](http://de.wikipedia.org/wiki/Festplatte). Fällt eine andere Platte aus, wird die Hot-Spare-Platte im laufenden Betrieb automatisch anstelle der defekten eingebunden. |
| Pairtät(splatte) | Der wesentliche Gedanke bei RAID 3 ist die Datensicherung. Im RAID 3 werden die eigentlichen Nutzdaten normal auf einer oder mehreren Datenplatten gespeichert.  Außerdem wird eine Summeninformation auf einer zusätzlichen Paritätsplatte gespeichert. Für die Paritätsplatte werden die Bits der Datenplatten zusammengezählt und die errechnete Summe wird darauf untersucht, ob sie eine gerade oder eine ungerade Summe darstellt; eine gerade Summe wird auf der Paritätsplatte mit dem Bit-Wert 0 gekennzeichnet; eine ungerade Summe wird mit dem Bit-Wert 1 gekennzeichnet. Die Datenplatten enthalten also normale Nutzdaten, während die Paritätsplatte nur die Summeninformationen enthält. |

#### Dateiindex



#### Fragmentierung

#### Miniatisierung

|  |  |
| --- | --- |
| Tangentiale | Computergenerierter Alternativtext: |
| Radiale | Computergenerierter Alternativtext: |
| Perpendikulare | Computergenerierter Alternativtext: |

# Programm Entwicklung

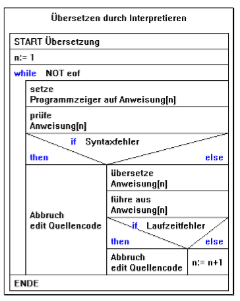
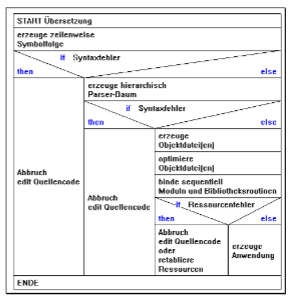
**Unterschied Wasserfallmodell und V-Modell:**

Zu jedem Step im V-Model (linke Seite) gibt es eine Testphase (rechte Seite)

## Prozess / Entwicklungsmodelle (s. 529)

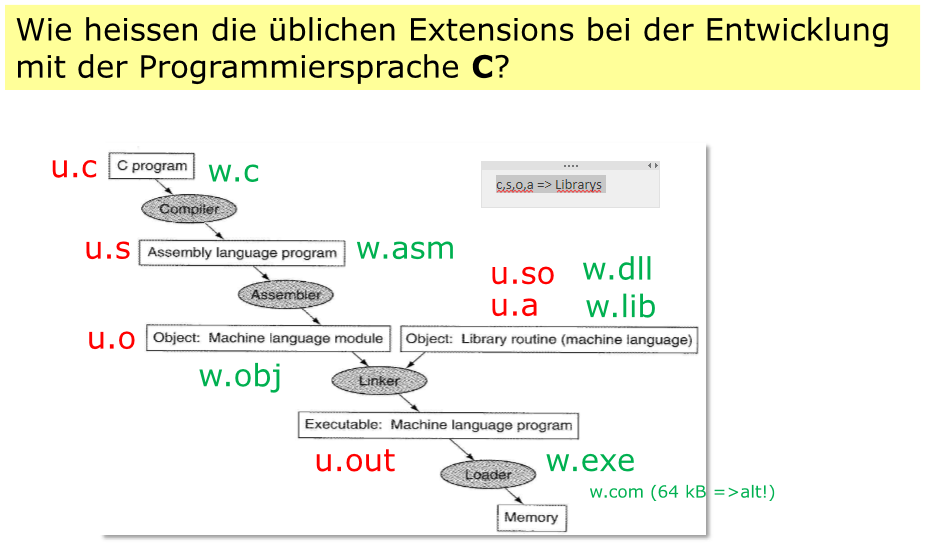
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Schwergewichtige (klassische)  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **Wasserfall-Modell** | |  |  | | --- | --- | | Klassische | Ohne Rückschritte (Grund für diverse Scheiterungen) | | Adaptiertes | Rückkehr ist möglich aber wird als Niederlage angesehen. Validierung erst am Schluss. | | Iterative-Entwicklung | Immer ein wenig Implementieren und Testen wieder ein wenig Implementieren und Testen (immer wieder im Kreis) | | | **V-Modell** | Jede Stufe hat seine eigene Testphase, anstatt wie beim Wasserfall-Modell erst am Schluss zu testen. | | **Inkrementelle und iterative Prozessmodelle** | |  |  | | --- | --- | | Spiralmodelle | Mehrfache Durchläufe | | Rational Unified Process | Bei IBM | | Open Unified Process |  | | |
| Leichtgewichtige  |  |  | | --- | --- | | **Extreme Programming (XP)** | Einer programmiert und der andere schaut zu bzw. unterstützt (keine Ablenkungen wie Handys oder Email)  Es wird immer z.B. nach 3 Stunden gewechselt. Extrem anstrengend aber grosse Fortschritte. | | **Scrum** |  | | **Kanban** |  | |

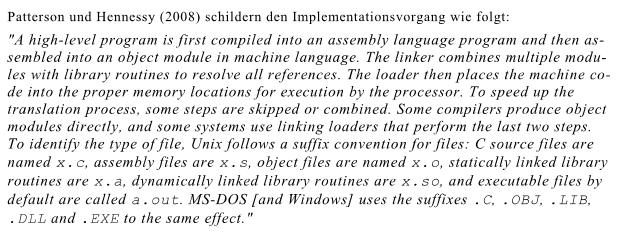
## Kompilieren vs. Interpretieren (S. 138)



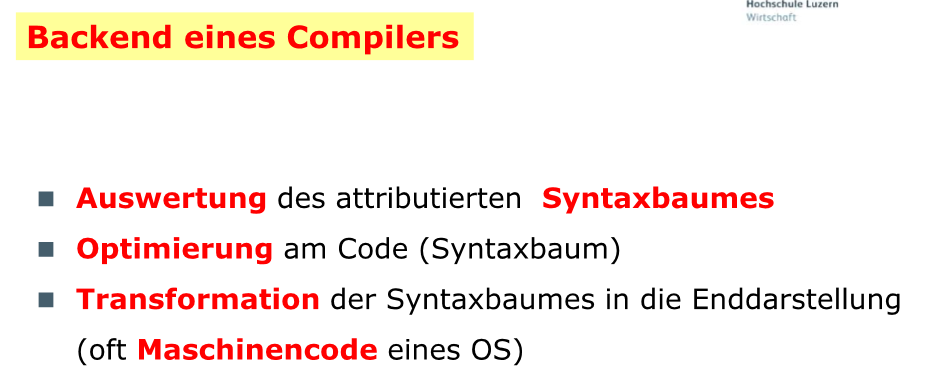
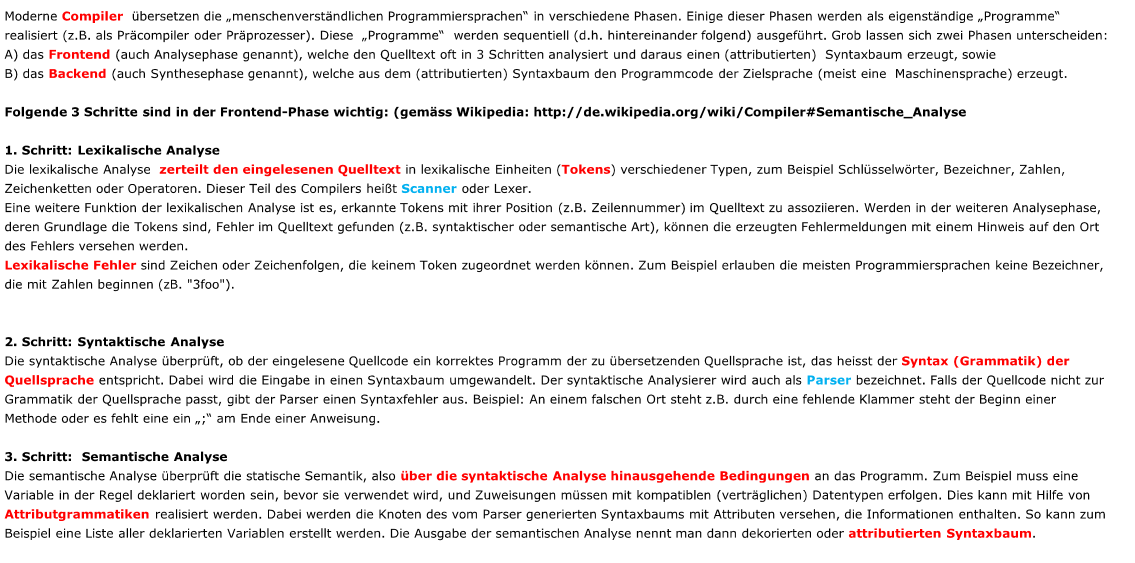
|  |  |
| --- | --- |
| **Interpreter** | * Korrektur möglich * Langsam * Ergebnis immer wieder sehen * Zukunft => selbst korrigierende Maschinen (KI) |
| **Compiler** | * Keine Korrekturen möglich * Sehr schnell (braucht aber Vorbereitungszeit) * Aktuell sehr häufig vertreten wegen der Geschwindigkeit |

### Unix vs. WIndows



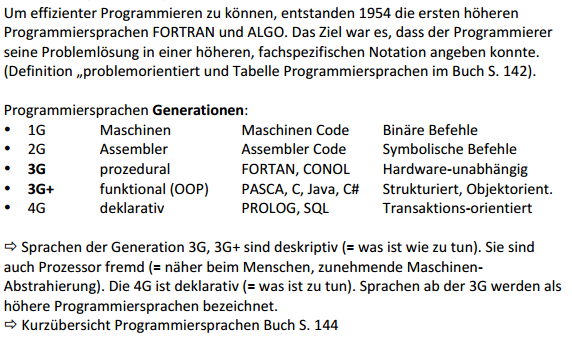


### Funktionsweise eines Compilers (Java…)



# Programmiersprachen und Algorithmen

## Generationen



## Prozedurale vs. Funktionale Programmiersprachen

**Prozedural**

Den Ansatz, Programme **in kleinere Teilaufgaben aufzuspalten**, bezeichnet man als prozedurale Programmierung. Die entstehenden Teilprogramme werden Prozeduren genannt. Praktisch alle aktuellen imperativen Programmiersprachen beinhalten den prozeduralen Ansatz.

Die Entwicklung prozeduraler Programmiersprachen und -techniken waren ein wesentlicher Schritt zwischen Assemblersprache und Hochsprachen, indem sie [Abstraktion](http://de.wikipedia.org/wiki/Abstraktion) und Zerlegung von Algorithmen ermöglichen.

**Funktional**

Die Aufgabenstellung und die bekannten Prämissen werden hier als funktionaler Ausdruck formuliert. Das selbstständige Anwenden von Funktionsersetzung und Auswertung seitens des Interpreters oder Compilers lösen dann die Aufgabenstellung. Das Programm kann als Abbildung der Eingabe auf die Ausgabe aufgefasst werden.

## Paradigmen

Paradigma = Denkweise

|  |  |
| --- | --- |
| Klassisch | Computergenerierter Alternativtext: ndu  PI ttform  Quellcode in Sprache Aa (z.B. Fortran,  Cobo, Pascal, C)  AB-Compiler für Plattform XY  Betriebssystem XY  Hardware |
| JEE | Computergenerierter Alternativtext: nd ng  lattfo m  Wind  Java Quellcode (Hello.java)  Java Compiler  Java Bytecode (Hello.class)  Virtuelle Java Maschine (JVM)  Interpreter  Maschinencode  Betriebssysteme  Hardware  Andr |
| .NET (Microsoft) | Computergenerierter Alternativtext: Compiler  n ung  Common Intermediate Language (CIL)  Common Language Runtime  JIT Compiler  Maschinencode  attfor  Windows (Desktop und Mobile)  Hardware  Vorteil: auch mit anderen Sprachen möglich zu programmieren  Nachteil: Nur für Windows nutzbar |
| iOS | Ziemlich ähnlich |

## Algorithmus

## Begriffe (s. 152)

|  |  |
| --- | --- |
| Definition |  |
| effektiv | Ziel zu 100% erreichen (Weg spielt keine Rolle) |
| effizient | Ziel mit möglichst wenig Aufwand (Zeit, Geld) erreichen => Wirtschaftlichkeit |
| endlich | Nach einer endlichen Zeit, wird der Algorithmus beendet |
| deterministisch | Nur definierte und reproduzierbare Zustände treten auf. **Bei gleicher Eingabe erfolgt immer gleiche Ausgabe.** Abarbeitungsschritte sind eindeutig festgelegt. |
| determiniert | Algorithmus ist **nach n-Schritten terminiert**, andernfalls handelt es sich um eine nicht-determinierende Ausführung. |
| Determinismus | Wenn für alle Eingabedaten der **Ablauf des Algorithmus eindeutig bestimmt** ist, andernfalls nicht-deterministisch. Ein nicht-deterministischer Algorithmus kann aber auch ein determiniertes Ergebnis haben. (z.B. return(2) am Schluss) |
| Determiniertheit | Wenn bei gleichen zulässigen Eingabewerten **stets das gleiche Ergebnis geliefert wird**, andernfalls nicht-determiniert. |
| 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. |

## Darstellung (s. 153-154)

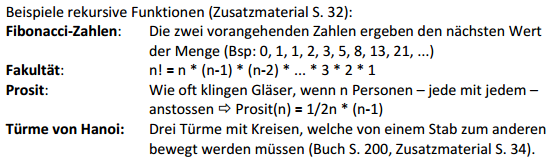
|  |  |
| --- | --- |
| Struktogramm |  |
| Klammercode |  |
| Aktivitätsdiagramm |  |

## O-Notation/Komplexität (s. 380)

0 = Ordnung

|  |  |
| --- | --- |
| **Komplexität** | Computergenerierter Alternativtext: Die O-N0tation ersetzt den  Begriff "ist proportional zu'  log n  und ermöglicht z.B. auch das  Ignorieren von maschinen-  spezifischen Eigenschaften.  Die O-Nation gilt (nur) für  genügend grosse n. Die O-  Notation ist unabhängig  von der Implementierung.  n log n  Sie ist allgemeingültig für alle  "Ausführungsumgebungen"  (OS X, Win8, etc.) und ist  deshalb eine "bleibende"  Eigenschaft eines  Algorithmus.  Beispiel:  f(n) 6n3+251n2+10007  bei sehr grossen Zahlen gilt:  0(n3)  konstant  logarithmisch  linear  n log n  quadratisch  kubisch  exponentiell  Jede Anweisung eines Programms wird höchstens einmal ausgeführt.  Dies ist der Idealzustand für einen Algorithmus.  Speicher- oder Zeitverbrauch wachsen nur mit der Problemgröße n.  Die Basis des Logarithmus wird häufig 2 sein, d. h. vierfache Daten-  menge verursacht doppelten Ressourcenverbrauch, 8-fache Daten-  menge verursacht 3-fachen Verbrauch und 1024-fache Datenmenge  10-fachen Verbrauch.  Speicher- oder Zeitverbrauch wachsen direkt proportional mit der  Problemgröße n.  Der Ressourcenverbrauch liegt zwischen n (linear) und n2 (quadra-  tisch).  Speicher- oder Zeitverbrauch wachsen quadratisch mit der Problem-  größe. Solche Algorithmen lassen sich praktisch nur für kleine Pro-  bleme anwenden.  Speicher- oder Zeitverbrauch wachsen kubisch mit der Problemgröße.  Solche Algorithmen lassen sich in der Praxis nur für sehr kleine  Problem rößen anwenden.  Bei doppelter, dreifacher und 1 0-facher Datenmenge steigt der Res-  sourcenverbrauch auf das 4-, 8- bzw. 1 024-farhe. Solche Algorithmen  Sind raktisch kaum verwendbar.  Computergenerierter Alternativtext: Die Komplexitätsklassen  in doppelt logarithmischer Darstellung  log n)  O(n3)  0(n2)  Anfang nicht entscheidend!  Wichtig ist das Verhalten bei ganz  hohen Zahlen  O(log n)  Eingabegröße (n)  Die Komplexitätsklassen  in "normaler" Darstellung |
| **Kubisch** | Computergenerierter Alternativtext: int maxfolgel(int zo, int n)  int i  , j , k, sum, max  -10000000;  for (i— O; i i++)  for (j —  sum O;  for (k —i, k «z j; k++)  sum zLkJ;  if (sum max)  max sum;  return max;  Zeitaufwand dieses Algorithmus:  3 geschachtelte for-Schlaufen  Aufwand etwa proportional 113 das heisst:  Kubischer Algorithmus |
| **Quadratisch** | Computergenerierter Alternativtext: int maxfolge2(int zLJ, int n)  int i  , j , sum, max  -10000000;  for ( i—O; i«n; i++)  sum O;  for ( j—i; j 41; j++)  sum Ztjl;  if (sum max)  max sum;  return max;  Idee wie bei Quadratischen Algorithmus  ..aber nicht immer neu die vollständige Summe berechnen  sondern die vorhergehende Summe S(i,j-l) verwenden  S(i,j)  n2 Quadratischer Algorithmus  Nur noch 2 Schlaufen d.h. proportional |
| **Lineare** | Computergenerierter Alternativtext: int maxfolge3(int zo,  int n)  int i , s, gesamtmax  —10000000, endesumme —  for ( i—O; i + + )  if ((zLiJ O && Math.abs(zlil)  Math.abs(zfil)»  endesumme ((s—endesumme+ztil) O) ? s  if (endesumme gesamtmax)  gesamtmax endesumme;  return gesamtmax;  O;  O;  • ...noch schneller: Man merkt sich immer die grösste  Abschnittmenge gesamtmax...  damit proportional zu d.h. Linearer Algorithmus  falsch für alle negative Zahlen! |
| **Exponentiell** | Computergenerierter Alternativtext: int prim(int zahl, int teiler )  if (zahl «2 Il zahl%2  O Il zahl%teiler O)  return O; keine Primzahl  else if ( teiler *teiler zahl)  return 1; Primzahl  return prim(zah , teiler+l);  oder: (gleiche Kategorie von Komplexität!)  Kombina-  torscher  Schaltkreis  out  Gegeben ein  kombinatorischer Schaltkreis  . Gibt  mit n  Eingängen  es eine Belegung der Eingänge, so dass der  Ausgang  1 ist?  211 Möglichkeiten (beim Beispiel oben: 25 32 Belegungen)  exponentieller Algorithmus |
| **Vergleich** | Computergenerierter Alternativtext: Die Wahl des Algorithmus ist also sehr gravierend für das  Zeitverhalten des Programms.  Unser Programm soll 10'000 Zahlen haben. Betrachten wir  NUR mal die Anzahl ausgeführter "Summen"  maxfolgel():  maxfolge2( ):  maxfol ge3( ):  1043  100003  1012  t(n)  —1 Billion  1042  100002  — 108  t(n)  100 Millionen  also 10000 mal schneller als maxfol gel( )  100001 104  t(n)  also 10000 mal schneller als maxfol ge2( ) und  100 Millionen mal schneller als maxfol gel( ). |

## Rekursive Programmierung (s. 199)



|  |  |
| --- | --- |
| Factorial | 5! = 120 (alle Zahlen 5 und weniger werden multipliziert)  n! = n \* (n-1)! |
| Zeichenkette Invertierung |  |
| Prosit |  |

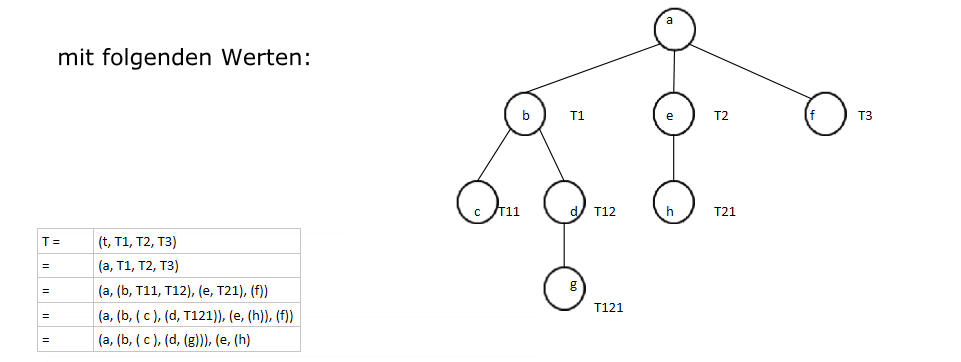
## Sorts

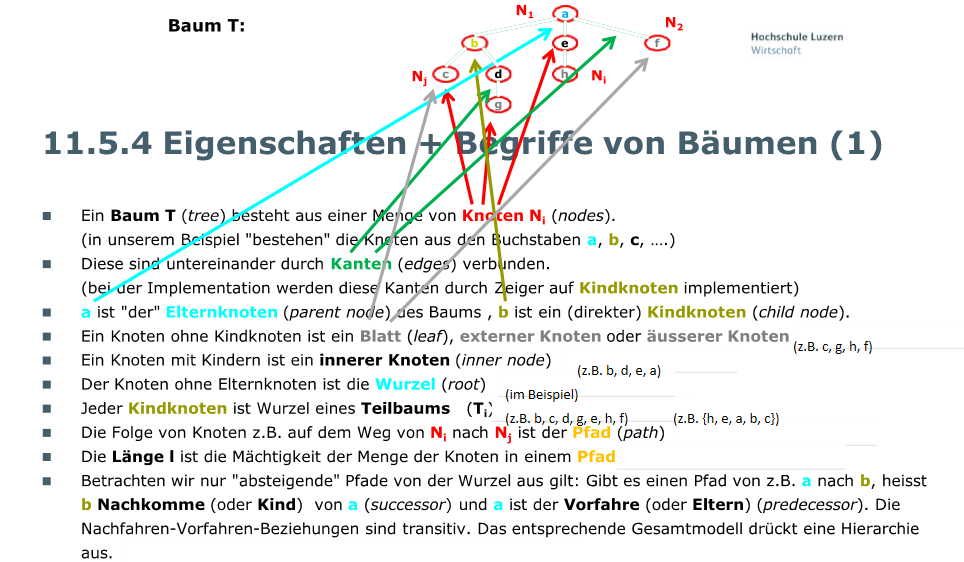


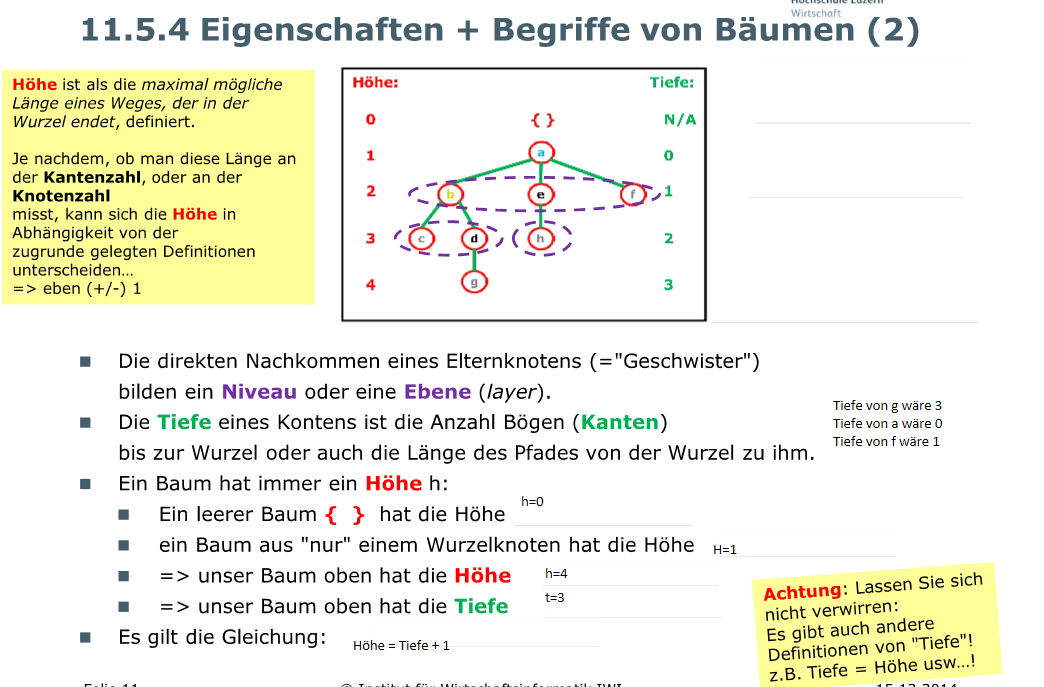
|  |  |  |
| --- | --- | --- |
| Bubble | Man beginnt bei den ersten 2 und vergleicht es (je nach dem vertauschen). Dann wird 2 und 3 verglichen, 3 und 4 usw. bis zum letzten. Das letzte ist somit das grösste/kleinste Element und muss nicht mehr verglichen werden. Es beginnt wieder von vorne. |  |
| Ripple | Das letzte Element wird mit jedem anderen Element verglichen und direkt vertauscht falls grösser oder kleiner. Danach fällt es weg und das zweitletzte Element wird verglichen bis alle sortiert sind. |  |
| Insertion | Man verschiebt immer das Array nach rechts, falls grösser oder kleiner und fügt es ein. So wird jedes Element durch den sortierten Bereich durchgehen. |  |
| Selection | Ähnlich wie Ripple, nur vertausche er nicht direkt die Zahlen aus, sondern merkt sich (schreibt in Variable) die grösste Zahl und vertauscht erst am Ende eines Durchgangs. |  |
| Shell | (s. 405-407) |  |
| Quick | (s. 407-409) |  |
| Merge | s. 409-413 |  |

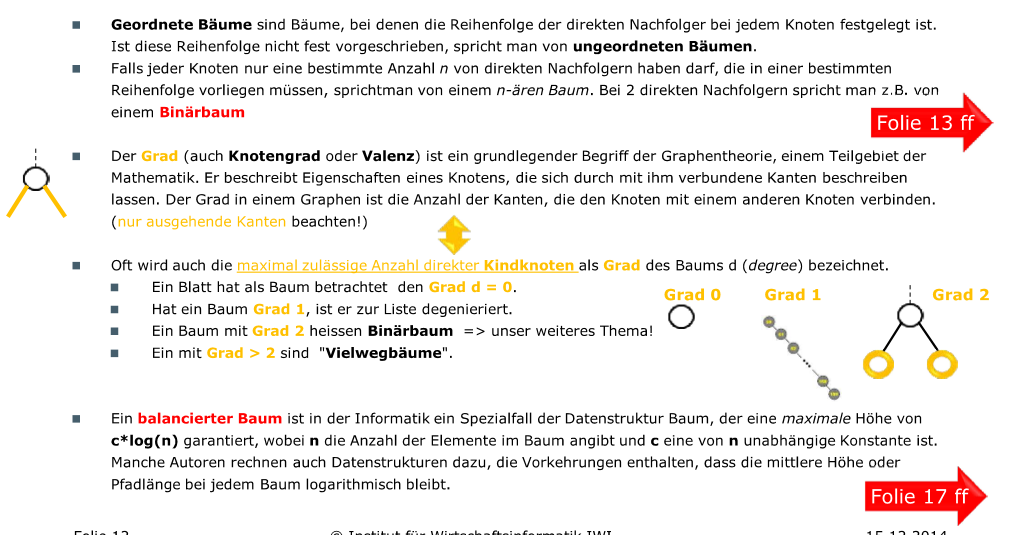
# Bäume (s.352)

Eine Art **dynamischer Datenstrukturen**, die dazu verwendet werden Daten hierarchisch speichern zu können. Dynamisch deshalb weil die Anzahl der Elemente nicht von vornhinein begrenzt ist, sondern jedes Mal wenn ein neues Element gespeichert werden soll Speicherplatz dafür reserviert werden kann und deshalb die Anzahl der Elemente die gespeichert werden können nicht von der Datenstruktur sondern der Größe des zur Verfügung stehenden Speichers abhängig ist.



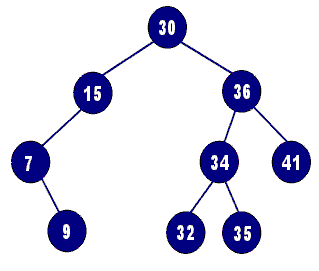






**Binärbäume** =  Besondere Eigenschaft: jeder Knoten **maximal 2 Nachfolger** hat (den linken und den rechten Teilbaum)

## Traversieren mit Binärbäumen



|  |
| --- |
| **Inorder**  Alle Elemente werden in der logisch sortieren Reihenfolge abgearbeitet, in Form eines rekursiven Algorithmus:   1. Behandle linken Teilbaum 2. Behandle die Wurzel 3. Behandle den rechten Teilbaum   Ausgabe: 7 9 15 30 32 34 35 36 41 |
| **Preorder**  Bei dieser Variante wird zuerst die Wurzel behandelt, danach die darunter liegenden Ebenen:   1. Behandle die Wurzel 2. Behandle den linken Teilbaum 3. Behandle den rechten Teilbaum   *Ausgabe:*  30 15 7 9 36 34 32 35 41 |
| **Postorder**  Bei dieser Variante werden zuerst die darunter liegenden Ebenen und abschliessend die übergeordneten Ebenen behandelt:   1. Behandle den linken Teilbaum 2. Behandle den rechten Teilbaum 3. Behandle die Wurzel   *Ausgabe:*  9 7 15 32 35 34 41 36 30 |

|  |
| --- |
| **Levelorder**  Bei dieser Variante werden alle Knoten der entsprechenden Knoten von links nach rechts aufgeschrieben.   1. levNo:= 0 2. solange entsprechende Ebene nicht leer    1. lies die Wurzel von links nach rechts    2. levNo:= levNo + 1   *Ausgabe:*  30 15 36 7 34 41 9 32 35 |
|  |

## Stacks (s. 343)

Für LIFO-Bearbeitung (Last In First Out) der Komponenten. Ist ebenfalls homogen, dynamisch und abstrakt.

