Informatik 5bi Schuljahr 2018/2019 als PDF exportieren

Informatik 5. Klasse - Schuljahr 2018/19

Lehrinhalte

- Lehrplaninhalte

Remote-Zugriff auf Schulserver

Kapitel

Leistungsbeurteilung

- **Test (SA)**
  - 2x Tests pro Semester
- **Mitarbeit (MA)**
  - Aktive Mitarbeit im Unterricht (aMA)
  - Mündliche Stundenwiederholungen (mMA)
  - Schriftliche Stundenwiederholungen (sMA)
- **Praktische Arbeiten (PA)**
  - 1x praktischer Arbeitsauftrag pro Woche

- Aktueller Leistungsstand

Stoff für den 4. Test in Informatik – 5BI – 29.05.2019

Stoff für den 3. Test in Informatik – 5BI – 27.03.2019

alles ab Kapitel 6.5 bis 20.03.2019

Stoff für den 2. Test in Informatik – 5BI – 21.01.2018

alles ab Kapitel 4.3 bis 6.5 inklusive (allen Hausübungen und Schulübungsstruktogramme)

Stoff für den 1. Test in Informatik – 5BI – 17.10.2018

Kapitel 1 bis Kapitel 4.2

Was wird in der 5. Klasse gemacht?

Gesellschaftliche Aspekte der Informationstechnologie

Geschichte der Informatik

- Einen Überblick hinsichtlich der wichtigsten Entwicklungsstufen der Informatik gewinnen

Kommunikation und Kooperation

- Digitale Systeme zum Informationsaustausch, zur Unterstützung der Unterrichtsorganisation und zum Lernen auch in kommunikativen und kooperativen Formen verwenden können.
- Informationsmanagement und Lernorganisation für die eigene Lernarbeit und Weiterbildung mit geeigneter Software in der Praxis umsetzen und dabei vorhandene Informationsquellen erschließen und unterschiedliche Informationsdarstellungen ausgehend von den Vorkenntnissen anwenden

Verantwortung, Datenschutz und Datensicherheit

- Wesentliche Maßnahmen und rechtliche Grundlagen im Zusammenhang mit Datensicherheit, Datenschutz und Urheberrecht kennen lernen sowie die Auswirkungen des Technikeinsatzes auf die Einzelnen und die Gesellschaft nachvollziehen

Berufliches Spektrum der Informatik

- Einsatzmöglichkeiten der Informatik in verschiedenen Berufsfeldern kennen lernen und somit in ihrer Berufsorientierung Unterstützung finden

Informatiksysteme – Hardware, Betriebssysteme und Vernetzung

Technische Grundlagen und Funktionsweisen (Hardware)

- Den Aufbau von digitalen Endgeräten beschreiben und erklären können.
- Die Funktionsweise von Informatiksystemen erklären können.
- Die Komponenten eines Rechners und ihr Zusammenspiel kennen
- Die verschiedenen Formen von Datenträgern kennen und bewerten können
- Speichermedien konfigurieren können, Partitionierung von Festplatten erstellen und bearbeiten können

Betriebssysteme und Software (DOS und Windows, Netzwerk)

- Grundlagen von Betriebssystemen erklären, eine graphische Oberfläche und Dienstprogramme benutzen können.
- Grundlegende Konsolenkommandos kennen und anwenden können
- Mit graphische Oberflächen (Desktop, Modern UI-Oberfläche) umgehen können
- Batches und System-Scripts erstellen können
- Wesentliche Konfigurationsmöglichkeiten von Windows kennen
- Ein vernetztes Informationssystem für die individuelle Arbeit aufbauen und nutzen können

Algorithmik und Programmierung

Zahlensysteme und Schaltalgebra

- Die Zahlensysteme der Informatik kennen und damit umgehen können
- Rechenregeln der Schaltalgebra kennen und anwenden können
- Grundschaltungen und sowie Halbaddierer-, Volladdierer-Schaltung kennen und Schaltungssimulationen erstellen können

Algorithmen und Datenstrukturen

- Grundprinzipien von Automaten, Algorithmen, Datenstrukturen und Programmen erklären können
- Grundlegende Datentypen kennen
- Einfache Algorithmen erklären, entwerfen, darstellen können.
- Die Codierung einfacher Algorithmen kennen und an einfachen Beispielen anwenden können
- Wichtige und bekannte Algorithmen aus Mathematik und Informatik (z.B. aus Teilbarkeitslehre, Reihenfolgeprobleme, ...) kennen
- Möglichkeiten der Visualisierungen von Algorithmen kennen

Programmierung (Objektorientierte Programmiersprache)

- Algorithmen in einer Programmiersprache implementieren können
- Kontrollstrukturen (Verzweigungen, Schleifen) kennen und einsetzen können
- Unterprogramme kennen und einsetzen können

Mathematische Arbeitsumgebungen

- Mit mathematischen Arbeitsumgebungen umgehen können

Angewandte Informatik, Datenbanksysteme und Internet

Allgemein

- Einblicke in wesentliche Begriffe und Methoden der Informatik, ihre typischen Denk-und Arbeitsweisen, ihre historische Entwicklung sowie ihre technischen und theoretischen Grundlagen gewinnen und Grundprinzipien von Automaten, Algorithmen und Programmen kennen lernen
- Begriffe und Konzepte der Informatik verstehen und Methoden und Arbeitsweisen anwenden können

Grundlagen der Bild-, Ton- und Videobearbeitung

- Standardsoftware zur Bildbearbeitung kennen und einsetzen können.
- Standardsoftware zur Audiobearbeitung kennen und einsetzen können.
- Standardsoftware zur Videobearbeitung kennen und einsetzen können.

Textverarbeitungs- und Satzsysteme

- Grundlagen der Text- und Seitengestaltung kennen und anwenden können
- Techniken zur Bearbeitung großer Dokumente (wie z.B. VWA) kennen und anwenden können
- den sicheren Umgang mit Standardsoftware zur schriftlichen Korrespondenz, zur Dokumentation, zur Publikation von Arbeiten erreichen

Präsentationssysteme und Visualisierung

- Standardsoftware zur Kommunikation und Dokumentation sowie zur Erstellung, Publikation und multimedialen Präsentation eigener Arbeiten einsetzen können.
- den sicheren Umgang mit Standardsoftware zur multimedialen Präsentation sowie zur Kommunikation erreichen
- Inhalte systematisieren und strukturieren sowie Arbeitsergebnisse zusammenstellen und multimedial präsentieren können

Web-Techniken

- Grundlagen in HTML (Grundformatierung, Tabellen, Container, Einbau von Bild-, Ton- und Videoelementen) kennen und anwenden können
- Das Editieren in einem Content-Management-System beherrschen

---

```
=> 5. Klasse (3 Stunden,  2-3 Tests pro Semester)
```

1) Zahlensysteme in der Informatik

- Kurzüberblick - Geschichtliche Entwicklung des Zählens
- Skriptum zur Einführung
- Informationseinheiten in der Informatik

- 1.01) Dualsystem
- 1.02) Hexadezimalsystem
- 1.03) Oktalsystem
- 1.04) Umrechnungen

1.01) Dualsystem

Das **duale Zahlensystem - auch Dualsystem oder Binärsystem** genannt - besteht aus **2 Ziffern**, gekennzeichnet durch 0 und 1. Man benötigt dieses Zahlensystem in der Informatik, da sich mit technischen Bauteilen sehr leicht die Zustände AN und AUS erzeugen lassen können. Diese Zahlen können entsprechend unserem „normalen“ Dezimalsystem verwendet werden. Man kann sie addieren, subtrahieren, multiplizieren und dividieren. Da sie sich also kaum vom „normalen“ Rechnen unterscheiden, eignen sie sich hervorragend, um in der EDV eingesetzt zu werden.

Zählen im Dualsystem

Auch hier beginnen wir mit 0 und zählen dann 1. Leider haben wir nur 2 Zahlen, also gehen uns hier die Zahlen schnell aus. Wir machen es jetzt aber genau wie im Dezimalsystem und nehmen eine Stelle dazu. Nach 0 und 1 kommen dann also 10 und 11. Wieder reichen die Stellen nicht! Also noch eine dazu: 100, 101, 110, 111, usw.

Zählen von 0 bis 15 im Dezimal- und Dualsystem

Dezimal	Dual
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Eine andere Schreibweise

Man kann Zahlen auch anhand ihrer Basis darstellen. Im Dezimalsystem haben wir 10 Zahlen zur Verfügung, von 0 bis 9. Mit 2 Stellen können wir also 10 * 10 = 100 Zahlen darstellen. 100 Zahlen? Aber 100 hat doch drei Stellen! Dieser Einwand stimmt. Da wir jedoch mit der Zahl 0 beginnen, ist 0 die 1. Zahl, 1 die 2. Zahl, … 98 die 99. Zahl und 99 die 100. Zahl.

Mit 3 Stellen können wir 10 * 10 * 10 = 1000 Zahlen darstellen. Jede Stelle entspricht einer 10-er Potenz.

An einem einfachen Beispiel versuche ich diesen Sachverhalt zu erklären.

Wir nehmen dazu die Zahl 372 und schreiben sie als kleine Rechnung auf:

$372 = 3*100 + 7*10 + 2*1$.

Das kann man jetzt noch anders darstellen als:

$3*10^2 + 7*10^1 + 2*10^0$.

Auf diese Weise kann man jetzt alle anderen Zahlen auch darstellen:

$6574 = 6*10^3 + 5*10^2 + 7*10^1 + 4*10^0$
$12032 = 1*10^4 + 2*10^3 + 0*10^2 + 3*10^1 + 2*10^0$

Verwendung der Potenzschreibweise bei binären Zahlen

Wendet man die Potenzschreibweise bei binären Zahlen an, so muss man eine andere Basis wählen. Es gibt ja nur 2 verschiedene Ziffern, 0 und 1. Also nehmen wir als Basis 2.

Die Zahl 1011 schreibt sich dann als

$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

Die Umrechnung von dezimalen in binäre Zahlen

Bei der Umrechnung der Dezimalzahlen verwenden wir die „Division mit Rest“ aus der Grundschule. Wir teilen die Zahl solange durch 2, bis als Ergebnis 0 herauskommt und merken uns dabei den Rest. Als Beispiel sollen die Zahlen 13 und 14 dienen.

13 / 2 = 6 Rest 1
6 / 2 = 3 Rest 0
3 / 2 = 1 Rest 1
1 / 2 = 0 Rest 1

Die Reste von unten nach oben aneinander gereiht ergeben dann die Dualzahl 1101.

14 / 2 = 7 Rest 0
7 / 2 = 3 Rest 1
3 / 2 = 1 Rest 1
1 / 2 = 0 Rest 1

Hieraus ergibt sich dann die Dualzahl 1110.

Addition von Dualzahlen

Einige erinnern sich vielleicht noch an die Addition von Zahlen wie wir sie in der Grundschule gelernt haben. Wir schreiben die Zahlen untereinander und addieren sie Stelle für Stelle. Dabei beginnen wir mit der letzten Stelle und arbeiten uns langsam nach vorne durch. Die gleiche Vorgehensweise

benutzen wir nun auch wenn wir Dualzahlen addieren wollen.

Die Addition funktioniert wie bei der Addition von Dezimalzahlen:

```
 0111
+0100
 ====
 1011
```

Addition mit Überlauf:

```
 1111
+0100
 ====
10011
```

Es gelten die Regeln 0+0=0, 1+0=1, 0+1=1, 1+1=0 Übertrag 1. Im Prinzip also nichts neues. Addiert man im Dezimalsystem 2 Zahlen so kommt bei 5+5 auch 0 Übertrag 1 heraus. Der Übertrag wird bei beiden System jeweils voran gestellt. Also gilt im Dualsystem 1+1=10.

Vorsicht Überlauf!

Die Addition ist im Prinzip problemlos. Es gibt allerdings einen Haken an der Sache: Die Addition funktioniert nur innerhalb eines bestimmten Wertebereiches. Woran liegt das? In der Realität können wir beliebig grosse Zahlen darstellen. Das geht leider nicht in der Informatik. Wir haben nur einen begrenzten Raum bzw. Speicherplatz zur Verfügung. Wir müssen aus diesem Grund den Speicherbereich einschränken (siehe was ist ein Byte), in unserem Beispiel nehmen wir als Speicherplatz ein Byte. Normalerweise verwendet man zur Addition von ganzen Zahlen, einen deutlich größeren Wertebereich, aber um einen überschaubaren Rahmen zu haben, begrenzen wir uns absichtlich auf ein Byte. Unsere größte darstellbare Zahl ist die 11111111, also dezimal 255. Was passiert nun, wenn wir eine 00000001, also 1, addieren? Das verrückte ist: es kommt 00000000, also 0 heraus. Da bekanntlich 1+1=0 Übertrag 1 gibt, bekommt man als Ergebnis in Wirklichkeit nicht 00000000 sondern 00000000 Übertrag 1. Dieser letzte Übertrag kann jedoch nicht mehr gespeichert werden und wird deshalb einfach ersatzlos gestrichen. Es ergibt sich daraus jedoch auch eine gewisse Logik. Durch meinen beschränkten Wertebereich komme ich irgendwann an meine obere Grenze. Bei der Addition von 1 fängt dann jedoch der Wertebereich wieder von vorne an, ich bin jetzt an der unteren Grenze, man durchläuft dann wieder den Bereich bis zur oberen Grenze, usw. Wenn wir also immer und immer wieder 1 addieren zählen wir unendlich oft von 0 bis 255, dann wieder 0 bis 255 usw.

Subktraktion von Dualzahlen

Drei Schritte zu Subtraktion

Hier kommen wir mit unserer normalen Schulmathematik nicht mehr weiter. Bevor wir uns mit dem komplizierten „Warum ist das denn so?“ beschäftigen, merken wir uns erst einmal den Mechanismus. Die Subtraktion von binären Zahlen wird durch die **Addition des Zweierkomplementes**

durchgeführt. Zur Erklärung beginnen wir im ersten Schritt mit dem **Einerkomplement**, dann schauen wir im zweiten Schritt was das **Zweierkomplement** ist und dann kommen wir im letzten Schritt zur **Subtraktion**.

Das Einerkomplement

Was ist das Komplement von Dualzahlen? Man bildet das sogenannte Einerkomplement, indem man jede Zahl durch ihr Gegenteil ersetzt, also die 0 durch die 1 und die 1 durch die 0.

01011010 wird zu 10100101 11101101 wird zu 00010010

Das Zweierkomplement

Das Zweierkomplement entspricht dem Einerkomplement, nur wird zusätzlich noch 00000001 addiert.

01011010 wird im Einerkomplement zu 10100101 im Zweierkomplement zu 10100110 11101101 wird im Einerkomplement zu 00010010 im Zweierkomplement zu 00010011

Die Subtraktion von Dualzahlen

Der Satz lautet: Die Subtraktion von 2 Zahlen erfolgt durch die Addition des Zweierkomplementes. Als konkretes Beispiel nehmen wir dazu die Rechnung 14-9=5.

!!WICHTIG:!!
Die restlichen Ziffern müssen immer mit 0 aufgefüllt werden.

9 ist im Dualsystem 00001001.

Das Einerkomplement zu 00001001 ist 11110110.

Das Zweierkomplement 11110111.

Dies addieren wir nun zu 14 also 00001110.

```
 00001110
+11110111
 ========
 00000101
```

Auch hier wäre die richtige Zahl eigentlich 00000101 Übertrag 1, da wir den Übertrag jedoch nicht speichern können, bleiben wir bei 00000101 was ja der Dezimalzahl 5 entspricht.

1.02) Hexadezimalsystem

Besonders **wichtig** ist in der **Informatik und Digitaltechnik** neben dem Binärsystem auch das **Hexadezimalsystem (Sedezimalsystem)**. Das Hexadezimalsystem verwendet die **Basis 16**, d.h. es gibt **16 verschiedene Ziffern**, **0 bis 9** und zusätzlich die Buchstaben **A bis F** (sog. Zahlzeichen; können auch als klein geschrieben werden: a-f).

Mit dem Hexadezimalsystem können auf einfachere und kürzere Weise Binärzahlen notiert werden. Mit einer 4-stelligen Binärzahl (auch als Halbbyte oder Nibble bezeichnet) lassen sich 16 (24 = 16) verschiedene Zahlen darstellen, und zwar 0 bis 15 (die Null zählt mit!). Da das Hexadezimalsystem die Basis 16 (= 24) verwendet, reicht eine (!) Hexadezimalzahl aus, um vier Bits (Binärziffern) darzustellen. Mit zwei Hexadezimalzahlen kann ein Byte (8 Bits) angeschrieben werden.

Gegenüberstellung Hexadezimal-, Binär- und Dezimalsystem

Hex	Binär	Dezimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Um eindeutig darauf hinzuweisen, dass es sich um eine Hexadezimalzahl handelt, kann ebenso wie in anderen Zahlensystemen die Basis tiefgestellt dazu geschrieben werden, z.B. $\mathbf{3F_{16}}$ (= 63_{10} dezimal) oder $\mathbf{93_{16}}$ (= 147_{10} dezimal). Es sind aber auch andere Schreibweisen üblich:

a) Vorangestelltes **0x (Prefix), z.B. 0x93**. Diese Notation wird in Programmiersprachen mit C-ähnlicher-Syntax verwendet.

b) Nachgestelltes **h** (Postfix), z.B. **93h**. Letztere Schreibweise ist besonders in der Technik gebräuchlich.

Umrechnung vom Dezimal- ins Hexadezimalsystem

Die Umrechnung funktioniert ähnlich der Umrechnung von Dezimal- zu Binärzahlen (s.o.). Nun muss aber, statt durch 2, durch 16 dividiert werden. Die Reste werden genauso von rechts nach links angeschrieben und geben, wenn das Ergebnis der Ganzzahlendivision 0 ist, das Endergebnis.

Beispiel: Die Dezimalzahl **304_{10}** soll in eine Hexadezimalzahl umgewandelt werden.

```
304 / 16 = 19 => 0 Rest
 19 / 16 =  1 => 3 Rest
  1 / 16 =  0 => 1 Rest
```

Für das Endergebnis werden jetzt die Reste **von unten nach oben gelesen**.
Somit ergibt sich ein Endergebnis von 130_{16}, das entspricht der Dezimalzahl **304_{10}**.

Umrechnung vom Hexadezimal- ins Dezimalsystem

Die Umrechnung vom Hexadezimal- ins Dezimalsystem kann genauso wie oben von Binär→Dezimal demonstriert, erfolgen. Die einzelnen Ziffern werden mit dem jeweiligen Stellenwert (16^n, wobei n = 0, 1, 2, ...) multipliziert und die jeweiligen Ergebnisse aufsummiert. Das folgende Beispiel demonstriert dies anhand der Hexadezimalzahl 130_{16}:

```
0 * 16^0  =    0
3 * 16^1  =   48
1 * 16^2  =  256
---------------
          =  304
```

Als Ergebnis erhalten wir 304 dezimal, womit die Probe - zur vorigen Rechnung in die umgekehrte Richtung - erfolgreich war. 304_{10} entspricht 130_{16}. Diese Antwort hätte in der Praxis natürlich auch ein wissenschaftlicher Taschenrechner geliefert. Es reicht dazu sogar der **Windows-Rechner** (den Sie nur auf die wissenschaftliche Ansicht umstellen müssen) oder unter Linux Programme wie z.B. KCalc.

1.03) Oktalsystem

Das Oktalsystem, auch Achtersystem genannt, verwendet die **Basis 8 (acht)**. Um Zahlen darzustellen, stehen die **Ziffern 0 bis 7** zur Verfügung. Die Bedeutung in der Informatik/Digitaltechnik ergibt sich dadurch, dass sich mit einer **Oktalzahl drei Bits** darstellen lassen. 23 ist 8, somit lassen sich mit 3 Bits 8 verschiedene Möglichkeiten darstellen. Eine Oktalzahl reicht, um diese Information wiederzugeben.

Das Oktalsystem wird hier insbesondere deshalb erwähnt, weil in vielen Programmiersprachen Zahlen auch in Oktalform angegeben werden können. Meist, z.B. in PHP, wird dazu eine 0 (Null) vorangestellt, z.B. 077 für 77_8 (= 63_{10}).

Umrechnungen erfolgen genauso wie beim Hexadezimalsystem gezeigt. Um die Oktalzahl auszurechnen, die einer best. Dezimalzahl entspricht, dividieren Sie die Dezimalzahl fortlaufend durch 8 und schreiben die Reste von rechts nach links an. In umgekehrter Richtung - von Oktal nach Dezimal - multiplizieren Sie die einzelnen Ziffern mit dem Stellenwert (8^n für n = 0, 1, 2, ...) und addieren die Teilergebnisse.

Umrechnung vom Dezimal- ins Oktalsystem

Die Umrechnung funktioniert ähnlich der Umrechnung von Dezimal- zu Binärzahlen (s.o.). Nun muss aber, statt durch 2, durch 8 dividiert werden. Die Reste werden genauso von rechts nach links angeschrieben und geben, wenn das Ergebnis der Ganzzahlendivision 0 ist, das Endergebnis.

Beispiel: Die Dezimalzahl **304_{10}** soll in eine Hexadezimalzahl umgewandelt werden.

```
304 / 8 = 38 => 0 Rest
 38 / 8 =  4 => 6 Rest
  4 / 8 =  0 => 4 Rest
```

Für das Endergebnis werden jetzt die Reste **von unten nach oben gelesen**.
Somit ergibt sich ein Endergebnis von 460_8, das entspricht der Dezimalzahl **304_{10}**.

Umrechnung vom Oktal- ins Dezimalsystem

Die Umrechnung vom Oktal- ins Dezimalsystem kann genauso wie oben von Binär→Dezimal demonstriert, erfolgen. Die einzelnen Ziffern werden mit dem jeweiligen Stellenwert (8^n, wobei n = 0, 1, 2, ...) multipliziert und die jeweiligen Ergebnisse aufsummiert. Das folgende Beispiel demonstriert dies anhand der Oktalzahl 460_8:

```
0 * 8^0  =    0
6 * 8^1  =   48
4 * 8^2  =  256
---------------
         =  304
```

Als Ergebnis erhalten wir 304 dezimal, womit die Probe - zur vorigen Rechnung in die umgekehrte Richtung - erfolgreich war. 304_{10} entspricht 460_{8}. Diese Antwort hätte in der Praxis natürlich auch ein wissenschaftlicher Taschenrechner geliefert. Es reicht dazu sogar der **Windows-Rechner** (den Sie nur auf die wissenschaftliche Ansicht umstellen müssen) oder unter Linux Programme wie z.B. KCalc.

Umrechnungen

Prinzipiell kann man jede Zahl von einem Zahlensystem in ein anderes Zahlensystem umrechnen. Dazu muss man normalerweise die Zahl immer zuerst in das Dezimalsystem und dann in das Ziel-Zahlensystem umrechnen. Ausnahmen gibt es bei Umrechnungen vom Binärsystem in ein anderes Zahlensystem, hier braucht man nicht unbedingt das Dezimalsystem.

Binärsystem ⇔ Oktalsystem

Binärsystem ⇒ Oktalsystem

Wir wissen ja bereits, dass eine Ziffer im Oktalsystem die Ziffern 0-7 annehmen kann und dass man 3 Bits im Dualsystem braucht um 8 verschiedene Zahlenwerte darzustellen.

⇒ Immer 3 Stellen im Binärsystem ergeben eine Ziffer/Stelle im Oktalsystem

Beispiel

Umwandlung der Zahl 010110011111_2 ins Hexadezimalsystem:

```
010 110 011 111
 ⇓   ⇓   ⇓   ⇓
 2   6   3   7
```

Die Zahl 010110011111_2 entspricht somit der Zahl 2637_8.

Oktalsystem ⇒ Binärsystem

Selbiges gilt auch für die Umrechnung vom Oktalsystem in das Binärsystem.

⇒ Eine Stelle im Oktalsystem entspricht 3 Stellen im Binärsytem

Beispiel

Umwandlung der Zahl 672_8 ins Binärsystem:

```
 6   7   2
 ⇓   ⇓   ⇓
110 111 010
```

Die Zahl 672_8 entspricht somit der Zahl 110111010_2.

Binärsystem ⇔ Hexadezimalsystem

Binärsystem ⇒ Hexadezimalsystem

Wir wissen ja bereits, dass eine Ziffer im Hexadezimalsystem die Werte 0-15 annehmen kann und dass man 4 Bits im Dualsystem braucht um 16 verschiedene Zahlenwerte darzustellen.

⇒ Immer 4 Stellen im Binärsystem ergeben eine Ziffer/Stelle im Oktalsystem

Beispiel

Umwandlung der Zahl 010110011111_2 ins Oktalsystem:

```
0101 1001 1111
 ⇓    ⇓    ⇓
 5    9    F
```

Die Zahl 010110011111_2 entspricht somit der Zahl $59F_{16}$.

Hexadezimalsystem ⇒ Binärsystem

Selbiges gilt auch für die Umrechnung vom Hexadezimalsystem in das Binärsystem.

⇒ Eine Stelle im Hexadezimalsystem entspricht 4 Stellen im Binärsytem

Beispiel

Umwandlung der Zahl 672_{16} ins Binärsystem:

```
 6    7    2
 ⇓    ⇓    ⇓
0110 0111 0010
```

Die Zahl 672_{16} entspricht somit der Zahl 11001110010_2.

Oktalsystem ⇒ Hexadezimalsystem

Will man vom Oktalsystem ins Hexadezimalsystem umrechnen, rechnet man am besten zuerst ins Binärsystem und dann ins Hexadezimalsystem um.

Hexadezimalsystem ⇒ Oktalsystem

Will man vom Hexadezimalsystem ins Oktalsystem umrechnen, rechnet man am besten zuerst ins Binärsystem und dann ins Oktalsystem um.

2) Schaltalgebra

https://elektroniktutor.de/swfdatei/gatter.swf

Das Adobe Flash Plugin wird benötigt, um diesen Inhalt anzuzeigen.

2.01) Grundlagen

In der Digitaltechnik sind die Eingangsvariablen so miteinander zu verknüpfen, dass die Ausgangsvariable einen definierten Zustand annimmt, um damit einen Prozess zu steuern. So soll ein Fahrstuhl nur dann fahren, wenn eine Zieletage gewählt wurde, in der er zurzeit nicht steht, seine Tür vollständig geschlossen ist und von ihr nichts eingeklemmt wurde und die Kabine nicht überlastet ist. Man kann durch Kombinieren der beschriebenen digitalen Gatter und Ausprobieren bei einfacheren Aufgaben die eine oder andere Lösung finden. Das eigentliche Ziel ist es, eine wirtschaftliche Digitalschaltung zu entwickeln, die mit einem Minimum gleicher Gattertypen zum Ziel führt.

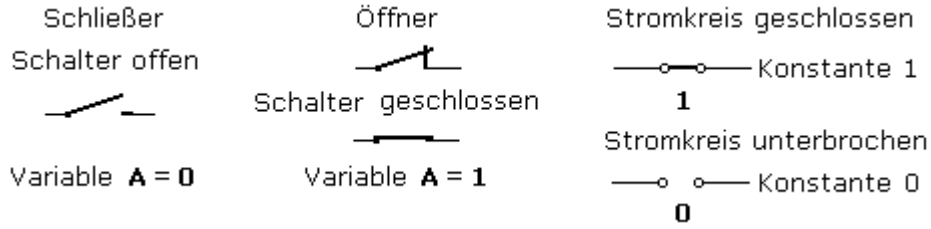
Der englische Mathematiker Georg Boole entwickelte eine Mengenalgebra, die in angepasster Weise als Boolesche Schaltalgebra bei der Problemlösung hilfreich ist. In der Schaltalgebra gibt es Variablen und Konstanten. Die binäre Digitaltechnik kommt mit zwei definierten logischen Zuständen, der 0 und 1 aus.

Konstante

In der Schaltalgebra gibt es nur die zwei konstanten Größen 0 und 1. In der elektronischen Schaltung entspricht eine dauerhaft geschaltete Leitung der Konstanten mit dem Zustand 1. Die dauerhafte Unterbrechung eines Stromkreises steht für die Konstante mit dem Wert 0.

Variable

Veränderbare, schaltbare Eingangsgrößen und davon abhängige veränderliche Ausgangswerte werden als Variable bezeichnet. In der binären Digitaltechnik nehmen sie entweder den Zustand 0 oder 1 an. In elektronischen Schaltungen können sie mit einem Schalter verglichen werden. Der geöffnete Schalter entspricht einer Variablen mit dem Wert 0. Bei geschlossenem Schalter nimmt die Variable den Wert 1 an.



Wir wissen, dass alle Grundrechnungsarten auf eine Addition zurückgeführt werden können. Deshalb ist das Ziel dieses Kapitels eine Maschine simulieren zu können, die addieren kann.

Informationseinheiten

BIT

Die Ziffern 0 und 1 werden physikalischen Zuständen zugeordnet:

```
0   Strom fließt nicht
1   Strom fließt
```

0 und 1 werden in der Informatik mit Bit bezeichnet. Ein Bit stellt die kleinste Informationseinheit dar (nur 2 Zustände – 0 und 1).

1 Bit ⇒ 2^1 ⇒ 2 Möglichkeiten
2 Bit ⇒ 2^2 ⇒ 4 Möglichkeiten
3 Bit ⇒ 2^3 ⇒ 8 Möglichkeiten
8 Bit = 1 Byte ⇒ 2^8 ⇒ 256 Möglichkeiten

BYTES

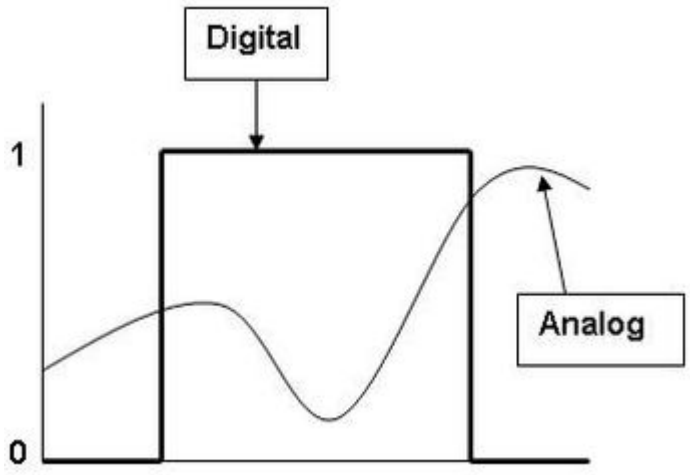
8 Bits sind ein Byte!

1 KB (Kilobyte) = 2^{10} = 1024 Byte
1 MB (Megabyte) = 2^{10} KB = 2^{10} x 2^{10} Byte = 2^{20} Byte = 1048576 Byte
1 GB (Gigabyte) = 2^{10} Byte
1 TB (Terrabyte) = 2^{10} Byte

Darstellung von 0 und 1

Ein Computer verarbeitet systembedingt Digitalsignale, da nur ausgewertet werden kann, ob eine Spannung anliegt oder nicht bzw. innerhalb eines Definitionsbereichs einen Wert über- oder unterschreitet. Diese zwei Zustände werden auch häufig bezeichnet als:

HIGH und **LOW**
TRUE und **FALSE**
AN und **AUS**
1 und **0**



2.02) Grundschaltungen

UND-Verknüpfung (=Serienschaltung)

Bei der UND-Verknüpfung führt der Ausgang das Signal 1, wenn an beiden Eingängen (a und b) das Signal 1 anliegt.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)	Zeichen (Mathematik)
UND	AND	Konjunktion	X=A∧B	X=A*B

Schaltwerttabelle

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

Schaltfunktion

f(a,b)=a∧b

Schaltsymbol

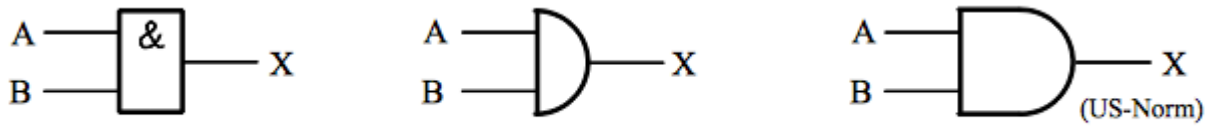


Bild 5-5: Schaltzeichen für UND-Gatter

elektronische Schaltung

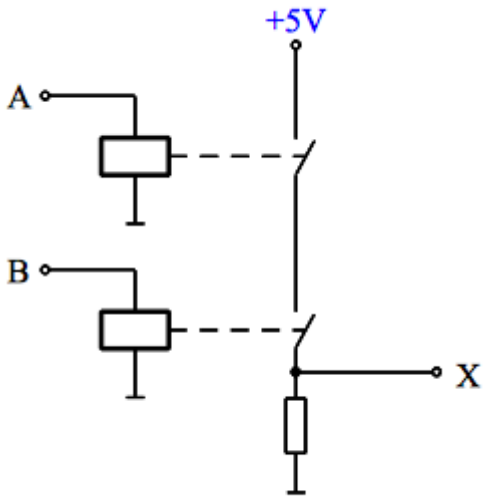


Bild 5-4: UND-Verknüpfung mit Relais

ODER-Verknüpfung (=Parallelschaltung)

Bei der ODER-Verknüpfung führt der Ausgang das Signal 1, wenn an einem der beiden Eingänge das Signal 1 anliegt.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)	Zeichen (Mathematik)
ODER	OR	Disjunktion	X=A∨B	X=A+B

Schaltwerttabelle

a	b	a∧b
0	0	0
0	1	1
1	0	1
1	1	1

Schaltfunktion

f(a,b)=a∨b

Schaltsymbol

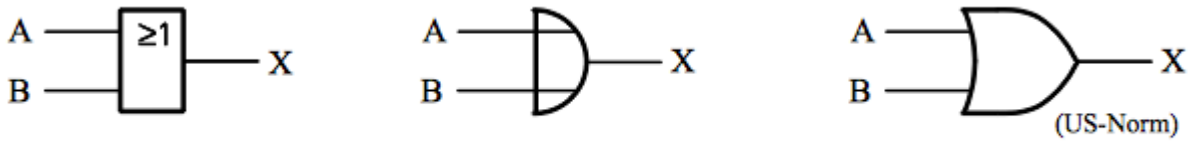


Bild 5-8: Schaltzeichen für ODER-Gatter

elektronische Schaltung

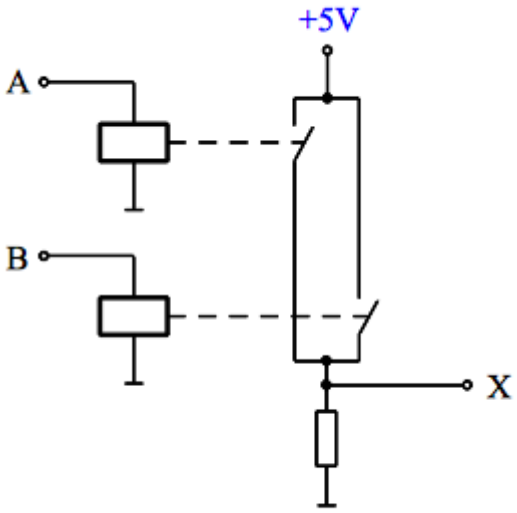


Bild 5-7: ODER-Verknüpfung mit Relais

NICHT-Verknüpfung (=NOT-Schaltung)

Bei einer NICHT-Verknüpfung wird der Ausgang logisch 1, wenn der Eingang logisch 0 ist, und umgekehrt.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)	Zeichen (Mathematik)
NICHT	NOT	Negation	X=¬A	X=Ā

Schaltwerttabelle

a	¬a
0	1
1	0

Schaltfunktion

f(a)=¬a

Schaltsymbol

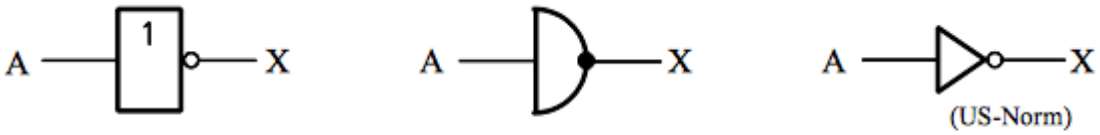


Bild 5-11: Schaltzeichen für NICHT-Gatter

elektronische Schaltung

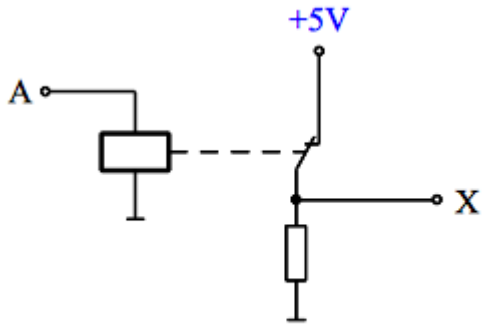


Bild 5-10: NICHT-Verknüpfung mit Relais

2.03) Zusammengesetzte Schaltungen

Die **Grundverknüpfungen** werden in der Regel miteinander **kombiniert**, sodass die Logik der Schaltung verändert wird. Beispielsweise kann man eine UND- mit einer NICHT-Verknüpfung kombinieren und erhält dadurch eine sogenannte NAND-Verknüpfung (NICHT-UND). Da solch zusammengesetzte Schaltfunktionen sehr häufig verwendet werden, haben sie **eigene Schaltzeichen** bekommen. Mit den folgenden Schaltfunktionen werden die Grundverknüpfungen erweitert.

- NAND
- NOR
- XOR
- XNOR

NAND-Verknüpfung

Eine Kombination aus einem **UND-Gatter mit nachfolgendem NICHT-Gatter ergibt ein NAND-Gatter**. Die Ausgangsvariable Z des UND-Gatters wird durch das NICHT-Gatter negiert und erzeugt die Ausgangsvariable X des NAND-Glieds. Viele logische Funktionen lassen sich durch den Einsatz von NAND-Gattern lösen. Oft steigt dadurch die Anzahl der notwendigen Gatter, mit dem wirtschaftlichen Vorteil nur einen Gattertyp zu verwenden.

Der Ausgangszustand eines NAND-Glieds ergibt 1, wenn nicht alle Eingangszustände 1 sind.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)
NEGIERTES UND	NOT AND	Negative Konjunktion	$Z = \overline{A \wedge B}$

Schaltwerttabelle

a	b	a⊼b
0	0	1
0	1	1
1	0	1
1	1	0

Schaltfunktion

f(a,b)=a⊼b

Schaltsymbol

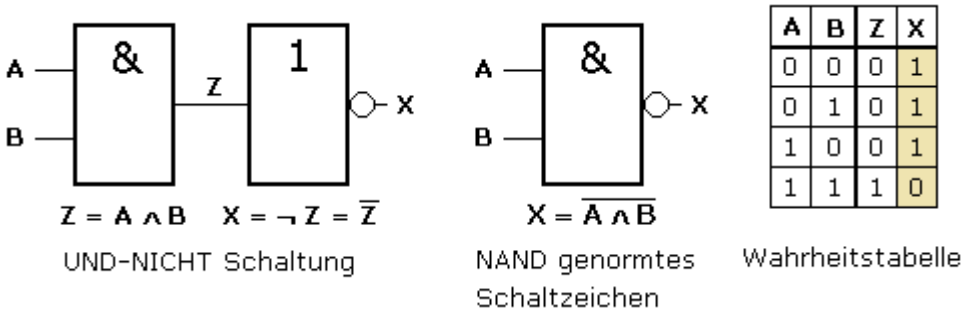


Bild 5-13: Schaltzeichen für NAND-Gatter

NOR-Verknüpfung

Eine Kombination aus einem **ODER-Gatter mit nachfolgendem NICHT-Gatter ergibt ein NOR-Gatter**. Die Ausgangsvariable Z des ODER-Gatters wird durch das NICHT-Gatter negiert und erzeugt die Ausgangsvariable X des NOR-Glieds. NOR-Glieder haben in logischen Schaltungen die gleiche wichtige Bedeutung wie NAND-Glieder.

Der Ausgangszustand eines NOR-Glieds ergibt 1, wenn alle Eingangszustände 0 sind.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)
NEGIERTES ODER	NOT OR	Negative Disjunktion	$Z = \overline{A \vee B}$

Schaltwerttabelle

a	b	a⊽b
0	0	1
0	1	0
1	0	0
1	1	0

Schaltfunktion

f(a,b)=a⊽b

Schaltsymbol

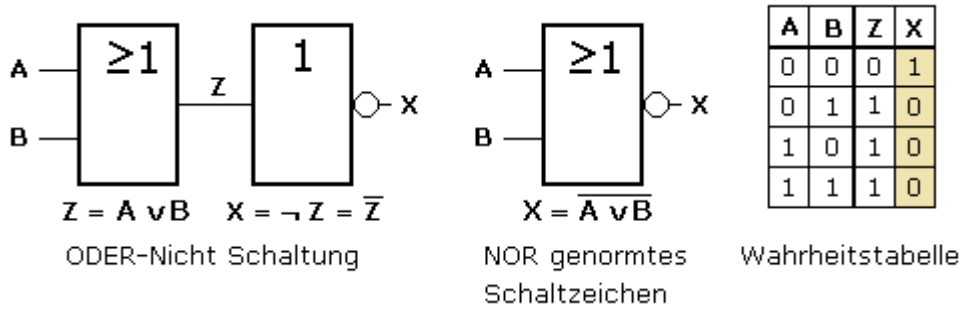


Bild 5-15: Schaltzeichen für NOR-Gatter

XOR-Verknüpfung

Die logische Verknüpfung des XOR-Gatters mit zwei Eingangsvariablen kann mit einem **'entweder - oder'** umschrieben werden. Die **Ausgangsvariable wird immer dann 'true' liefern, wenn die Eingangsvariablen unterschiedliche Zustände haben**. Die Wahrheitstabelle des XOR-Gatters entspricht dem ODER-Gatter mit dem Ausschluss gleicher Eingangszustände, also exklusiv der Äquivalenz. Dieses Verhalten wird als Antivalenz bezeichnet.

Der Ausgangszustand eines XOR-Glieds ergibt 1, wenn eine ungerade Zahl der Eingangszustände 1 und alle anderen Eingangszustände 0 sind.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)
Exklusives ODER (Antivalenz)	Exclusiv OR	Exklusive Disjunktion	Z=a⊻b

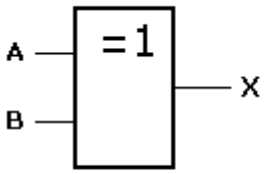
Schaltwerttabelle

a	b	a⊻b
0	0	0
0	1	1
1	0	1
1	1	0

Schaltfunktion

f(a,b)=a⊻b

Schaltsymbol



Exklusiv ODER
XOR – Antivalenz

$X = (A \wedge \overline{B}) \vee (\overline{A} \wedge B)$
$X = A \underline{\vee} B$
$X = A \oplus B$

Funktionsgleichung

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Wahrheitstabelle

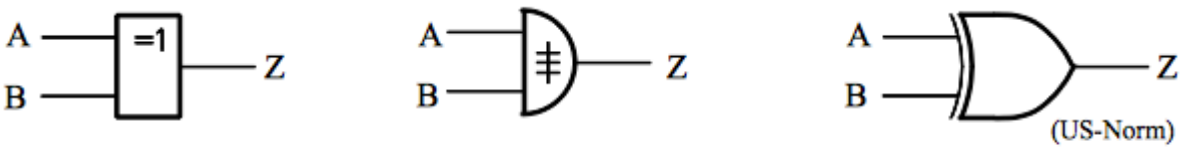


Bild 5-21: Schaltzeichen für ANTIVALENZ-Gatter

XNOR-Verknüpfung

Die Bezeichnung **Äquivalenz bedeutet Gleichwertigkeit**. Beim XNOR-Gatter mit zwei Eingangsvariablen ist der **Zustand der Ausgangsvariable 1, wenn beide Eingangsvariablen den gleichen Zustand 0 oder 1 haben**. Die Funktion kann durch eine Schaltung mit vier NOR-Gattern erreicht werden. Es sind zwei unterschiedliche Schaltzeichen zu finden.

Der Ausgangszustand eines XNOR-Glieds ergibt 1, wenn eine gerade Zahl der Eingangzustände 1 und alle anderen Eingangszustände 0 aufweisen oder alle 0 sind.

Zeichen

Deutsch	Englisch	Fachausdruck	Zeichen (boolsche Algebra)
Exklusives NICHT ODER (Äquivalenz)	Exclusiv NOT OR	Biimplikation	Z=a≡b

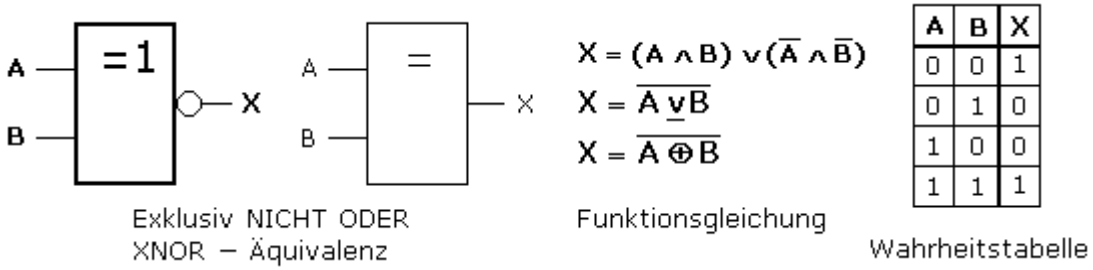
Schaltwerttabelle

a	b	a≡b
0	0	1
0	1	0
1	0	0
1	1	1

Schaltfunktion

f(a,b)=a≡b

Schaltsymbol



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

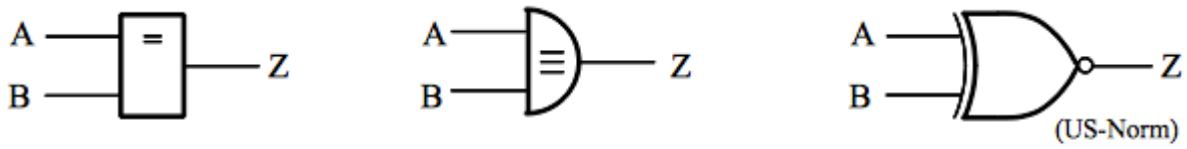


Bild 5-19: Schaltzeichen für ÄQUIVALENZ-Gatter

xor_nand.swf

Gesetze der Schaltalgebra

Vorrang- und Klammerregel

Wie in der Algebra ist auch in der Digitaltechnik auf eine bestimmte Reihenfolge der Operationen zu beachten. Die Negation sollte vor der Konjunktion (UND) und diese vor der Disjunktion (ODER) ausgeführt werden. Das ist vergleichbar mit der Aussage, dass die Multiplikation Vorrang vor der Addition hat.

Bei mehreren Variablen und unterschiedlichen Verknüpfungen kann eine Klammersetzung notwendig sein. Beachtet man die Vorrangregel, kann man auf viele Klammern verzichten. Ein Setzen zeigt sogleich eindeutig, welche der Variablen wie zu verknüpfen ist. Bei ODER sollte immer geklammert werden, ebenfalls beim Anwenden der De Morganschen Gesetze auf NAND- und NOR-Verknüpfungen.

$X = A \vee B \wedge C$

nach der Vorrangregel gilt

$\boxed{X = A \vee (B \wedge C)}$

festgelegte Abfolge der Verknüpfungen

$\boxed{Z = (A \vee B) \wedge C}$

A	B	C	B∧C	A∨(B∧C)	A∨B	(A∨B)∧C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	1	0
0	1	1	1	1	1	1
1	0	0	0	1	1	0
1	0	1	0	1	1	1
1	1	0	0	1	1	0
1	1	1	1	1	1	1

$A \vee (B \wedge C) \neq (A \vee B) \wedge C$

Bei drei Eingangsvariablen und binären Zuständen gibt es 2^3 = 8 Eingangskombinationen. Die Wahrheitstabelle zeigt bei definierter Klammersetzung oder Beachtung der Vorrangregel UND vor ODER unterschiedliche Ergebnisse.

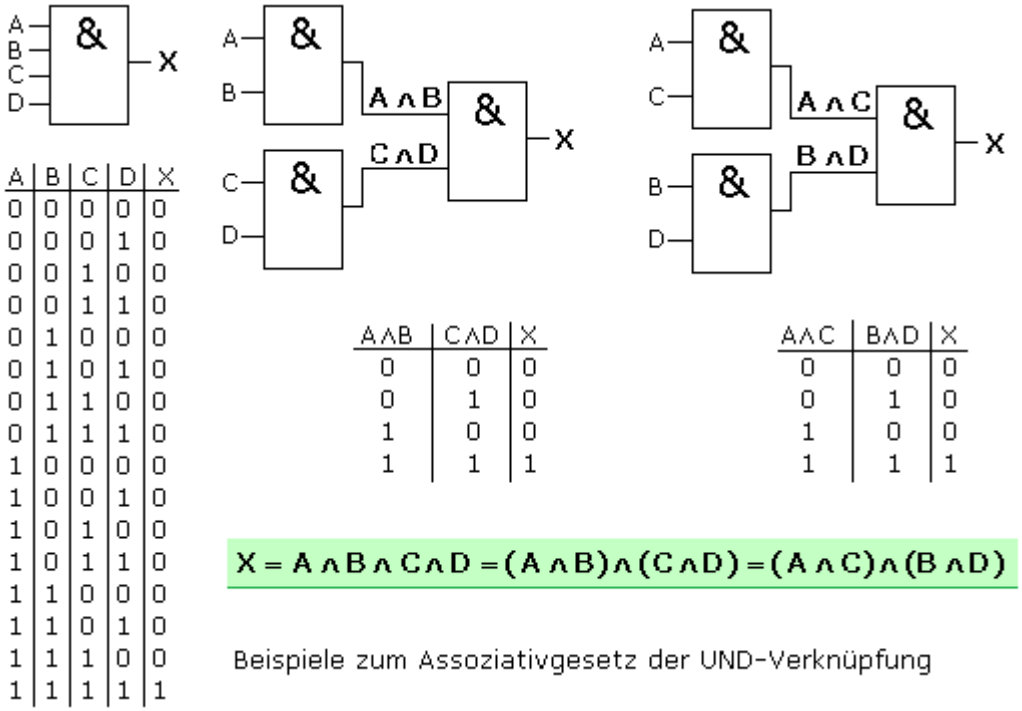
Kommutativgesetz

Das Kommutativ- oder Vertauschungsgesetz besagt, dass bei der UND sowie ODER Verknüpfung auch bei beliebiger Vertauschung der Reihenfolge der Eingangsvariablen das Ergebnis gleich bleibt.

UND	$X = A \wedge B \wedge C = C \wedge A \wedge B = B \wedge C \wedge A$
ODER	$X = A \vee B \vee C = C \vee A \vee B = B \vee C \vee A$

Assoziativgesetz

Das Assoziativ- oder Verbindungsgesetz besagt, dass bei einer UND- beziehungsweise ODER-Verknüpfung mit mehr als zwei Schaltvariablen die Verknüpfung auch stufenweise nacheinander in beliebiger Reihenfolge erfolgen kann.



A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

A∧B	C∧D	X
0	0	0
0	1	0
1	0	0
1	1	1

A∧C	B∧D	X
0	0	0
0	1	0
1	0	0
1	1	1

$$X = A \wedge B \wedge C \wedge D = (A \wedge B) \wedge (C \wedge D) = (A \wedge C) \wedge (B \wedge D)$$

Beispiele zum Assoziativgesetz der UND-Verknüpfung

Anstelle des UND-Gatters mit vier Eingangsvariablen lassen sich zwei UND-Gatter mit zwei Eingängen verwenden. Die Variablen A, B, C, D werden einzeln beliebig mit den Eingängen verbunden. Die Ausgangsvariable der beiden UND-Gatter kann den Wert 0 oder 1 annehmen. Beide Ausgangsvariablen sind nochmals mit einem UND-Gatter zu verknüpfen, wobei sich wieder vier Eingangskombinationen ergeben. Nur wenn alle Eingangsvariablen 1 sind, wird der Ausgangszustand ebenfalls 1.

Das Assoziativgesetz gilt gleichermaßen für die ODER-Verknüpfung. Die Klammersetzung ist nicht notwendig und soll nur verschiedene Verteilungen besser erkennbar machen.

Distributivgesetz

Das Distributiv- oder Verteilungsgesetz wird zur Vereinfachung von Verknüpfungsgleichung angewendet. Es ist vergleichbar mit dem Ausmultiplizieren und Ausklammern von Variablen der normalen Algebra. Da die logische UND-Verknüpfung der algebraischen Multiplikation entspricht, während die ODER-Verknüpfung mit der Addition vergleichbar ist, gibt es zwei unterschiedliche Distributivgesetze.

Konjunktives Distributivgesetz

Eine Variable wird mit UND verknüpft und auf den Folgeausdruck verteilt. Die Vorgehensweise entspricht dem aus der Algebra bekannten Ausmultiplizieren eines Klammerausdrucks mit einem Faktor. Im umgekehrten Fall kann eine Variable, die mit mehreren anderen Variablen verknüpft ist, ausgeklammert werden. Das bedeutet für den Schaltungsaufbau eine Einsparung an Gattern.

$$X = A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

⇨ Ausmultiplizieren
⇦ Ausklammern

$X = A \wedge (A \vee B)$
$X = (A \wedge A) \vee (A \wedge B)$
$X = A \vee (A \wedge B) = A$
$X = A$

		UND	ODER	
A	B	A∧B	A	X
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	1	1

Beim Ausklammern wird die Variable mit ihrem Verknüpfungszeichen, hier UND, vor die Klammer gesetzt. Die in der Klammer stehenden Variablen werden mit dem zuvor zwischen den Klammern stehenden ODER verknüpft.

Disjunktives Distributivgesetz

Eine Variable kann durch ein vergleichbares Ausmultiplizieren auf andere Ausdrücke verteilt werden oder bei mehrfachem Auftreten ausgeklammert werden. Beim Ausklammern wird das ODER Verknüpfungszeichen mit der Variablen vor die Klammer geschrieben. disjunktives Distributivgesetz

$$X = A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

⇨ Ausmultiplizieren
⇦ Ausklammern

$X = A \vee (A \wedge B)$
$X = (A \vee A) \wedge (A \vee B)$
$X = A \wedge (A \vee B)$
$X = A$

		ODER	UND	
A	B	A∨B	A	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

De Morgansche Gesetze

Die boolesche Algebra wurde vom englischen Mathematiker De Morgan weiter entwickelt. Für die Schaltalgebra gibt es zwei De Morgansche Gesetze. Sie machen Aussagen zur Negation einer Verknüpfung und der Umkehr von Verknüpfungszeichen. Mit den De Morganschen Gesetzen lassen sich bei der Entwicklung von Digitalschaltungen Gatter gegeneinander austauschen, Schaltungen verkleinern oder mit nur einem Gattertyp verwirklichen. Das erste Gesetz ist für die NAND-Verknüpfung und das zweite Gesetz entsprechend für die NOR-Verknüpfung definiert.

$$X = \overline{A \wedge B} = \overline{A} \vee \overline{B}$$

1. De Morgansche Gesetz

			ODER		
A	B	$\overline{A \wedge B}$	$\overline{A}$	$\overline{B}$	X
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

$$X = \overline{A \vee B} = \overline{A} \wedge \overline{B}$$

2. De Morgansche Gesetz

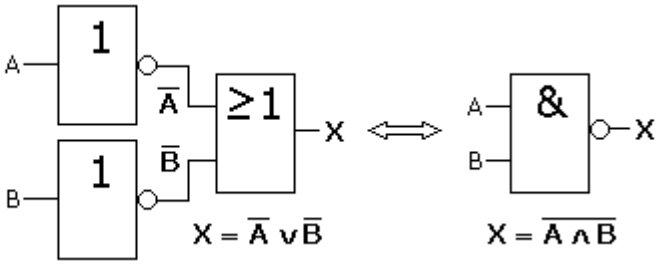
			UND		
A	B	$\overline{A \vee B}$	$\overline{A}$	$\overline{B}$	X
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Ist eine Verknüpfung insgesamt negiert, so ist ihre Ausgangsvariable negiert. Die Negation kann auf die Einzelglieder aufgeteilt werden, wobei aus der Grundverknüpfung UND ein ODER beziehungsweise aus ODER ein UND wird. Eine doppelte Negation hebt sich auf. Getrennte Negationsstriche über

Variablen bedeuten, dass die Eingangsvariablen negiert sind.

Die oben zum 1. De Morganschen Gesetz gezeigte Verknüpfung kann schaltungstechnisch mit zwei NICHT- und einem ODER-Gatter verwirklicht werden. Wie zu erkennen ist, folgt das gleiche Ergebnis mit nur einem NAND-Gatter. Für das 2. De Morgansche Gesetz gilt die entsprechende Aussage. Zwei NICHT- und ein UND-Gatter reduzieren sich auf den Einsatz eines NOR-Gatters. Mit der doppelten Negation und den De Morganschen Gesetzen kann bei einer gegebenen Funktionsgleichung auf das Bestehen einer derartige Vereinfachung geprüft werden. Treten dabei mehrfache Negationen auf, so lassen sie sich von innen nach außen auflösen.

$$X = \overline{A} \vee \overline{B}$$
$$\overline{\overline{X}} = \overline{\overline{A} \vee \overline{B}} \quad \overline{\vee} \rightarrow \wedge$$
$$\overline{\overline{X}} = \overline{\overline{A} \wedge \overline{B}} \quad \overline{\overline{A}} = A$$
$$X = \overline{A \wedge B}$$
$$\boxed{X = \overline{A \wedge B} = \overline{A} \vee \overline{B}}$$



Digitale Rechenschaltungen

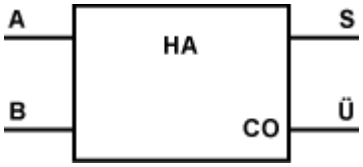
Aus logischen Verknüpfungen lassen sich digitale Schaltungen zusammenbauen, mit denen man Rechenvorgänge durchführen kann. Das heißt, diese Schaltungen haben zwischen ihren Eingängen eine Kombination aus logischen Verknüpfungen, die einem Rechenvorgang entspricht. In der Digitaltechnik kennt man Rechenschaltungen hauptsächlich für das duale Zahlensystem und den BCD-Code. Im Prinzip kann für jedes Zahlensystem eine Rechenschaltung aufgebaut werden.

Einige Rechenschaltungen der Digitaltechnik

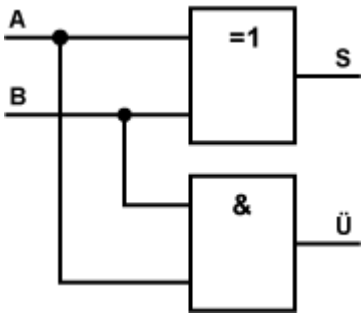
- Addiererschaltungen
- Subtrahiererschaltungen
- Addier-Subtrahier-Werke
- Multiplikationsschaltungen
- Arithmetisch-logische Einheit (ALU)

Stellvertretend für alle Rechenschaltungen dienen die folgenden Ausführungen zum Halbaddierer und dem Volladdierer.

Halbaddierer



Ein Halbaddierer ist die einfachste Rechenschaltung und kann zwei einstellige Dualziffern addieren. Der Eingang A des Halbaddierers ist der Summand A, der Eingang B ist der Summand B. Die Schaltung hat zwei Ausgänge. Der Ausgang S als Summenausgang (2^0) und der Ausgang Ü als Übertrag (2^1) für die nächsthöhere Stelle im dualen Zahlensystem.



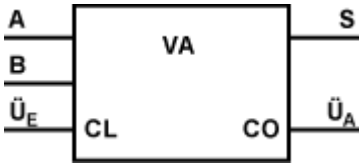
B	A	Ü	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Der Halbaddierer ist eine Schaltung aus den Verknüpfungsgliedern XOR und UND. Das XOR ist die Addier-Verknüpfung. Das UND stellt fest, ob ein Übertrag für die nächsthöhere Stelle vorgenommen

werden muss. Aus der Tabelle ist ersichtlich, dass das Ergebnis aus der Spalte Summe (S) einer Exklusiv-ODER-Verknüpfung (Antivalenz, XOR) entspricht. Das Ergebnis der Spalte Übertrag (Ü) entspricht einer UND-Verknüpfung. Die so entstandene Schaltung wird als Halbaddierer bezeichnet. Sie ist in der Lage zwei 1-Bit Summanden (einstellig) zu addieren.

Volladdierer

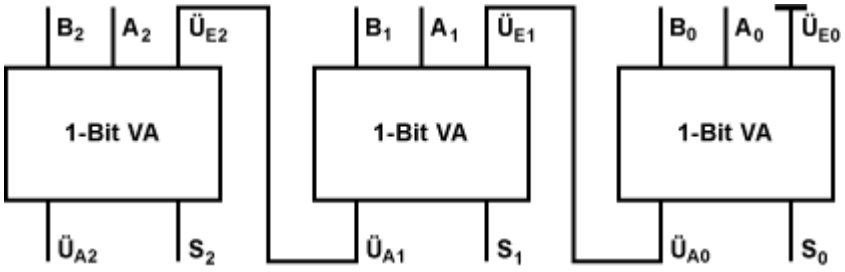
Um **mehrstellige Dualzahlen** addieren zu können benötigt man Schaltungen die auch einen Übertrag einer niederwertigen Stelle berücksichtigen. Man spricht vom **Übertragseingang ÜE**. Die Schaltung bezeichnet man als **Volladdierer (VA)**. Ein Volladdierer kann drei Dualzahlen addieren. Bzw. zwei Dualzahlen addieren und den Übertrag aus einer niederwertigen Stelle berücksichtigen.



Folgende Wahrheitstabelle ergibt sich:

$Ü_E$	B	A	$Ü_A$	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3-Bit-Volladdierer



Folgende Schaltung zeigt einen 3-Bit-Volladdierer (VA), der aus drei 1-Bit-Volladdierer (VA) realisiert wurde. Damit lassen sich zwei dreistellige Dualzahlen addieren. Der Eingang $Ü_{E0}$ liegt an 0 V (Masse), weil in der niederwertigsten Stelle kein Übertrag berücksichtigt werden muss.

Beispiel

```
 010 = A
+111 = B
```

```
---
1001 = S
```

0. Stelle

$Ü_{E0}$	A_0	B_0	$Ü_{A0}$	S_0
0	0	1	0	1

1. Stelle

$Ü_{E1}$	A_1	B_1	$Ü_{A1}$	S_1
0	1	1	1	0

2. Stelle

$Ü_{E2}$	A_2	B_2	$Ü_{A2}$	S_2
1	0	1	1	0

Ergebnis

$Ü_{A2}$ S_2 S_1 S_0

```
1  0  0  1
```

2.07) Übungen

1) Um ihren Swimmingpool schneller füllen zu können, hat Fam. Huber zwei Wasserleitungen (beide verfügen über einen Absperrhahn B und C verlegt, die direkt in den Pool führen. Diese Leitungen zweigen beide von einer Hauptleitung ab, die ebenfalls über einen Absperrhahn A verfügt. Demnach kann Wasser nur in den Pool fließen, wenn der Hauptabsperrhahn A und zumindest einer der anderen Absperrhähne offen sind.
Stelle die Situation mittels einer Wahrheitstabelle und einer Schaltfunktion (LogikSim, …) dar.

2) Frau Müller sagt zu ihrem Mann: „Den ganzen Sonntag hockst du auf dem Sofa, trinkst Bier und stopfst dir Kartoffelchips in den Schlund! Du solltest mal joggen!
Ihr Gatte erwidert: „Ich verspreche dir für nächsten Sonntag Folgendes:

- Wenn ich jogge, werde ich sogar auf Bier oder auf Chips verzichten.
- Wenn ich keine Chips esse, trinke ich kein Bier oder ich jogge nicht.
- Wenn ich aber jogge, brauche ich unbedingt hinterher ein Bier!
- Allerdings: Auch falls ich nicht jogge, werde ich auf jeden Fall Bier oder Chips zu mir nehmen.“

Falls Herr Müller seine Versprechen einhält, wie könnten seine Aktivitäten am nächsten Sonntag aussehen? Erstelle einen schaltalgebraischen Ausdruck und die Wahrheitstabelle zu diesen Aussagen.

3) Zeichne für folgende schaltalgebraischen Ausdrücke die zugehörige Schaltfunktion bzw. erstelle die jeweilige Wahrheitstabelle!

- $f(A,B) = (\neg A \vee \neg B)$
- $f(A,B,C) = (\neg A \wedge \neg B) \wedge C$
- $f(A,B) = (\neg A \vee \neg B) \vee (\neg A \wedge B)$

4) Überprüfe das Gesetz von De Morgan mit Hilfe einer Leitwerttabelle und stelle die Schaltung im Digitalsimulator dar.

5) Überprüfe das Distributivgesetz mittels einer Leitwerttabelle und stelle die Schaltung im Digitalsimulator dar.

6) Vereinfache die Schaltfunktion $f(a,b)= a \wedge (\neg a \vee b)$ und baue die Schaltung im Digitalsimulator auf.

7) Vereinfache die Schaltfunktion $f(a,b)= (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b)$

8) Vereinfache die Schaltfunktion $f(a,b)= (a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge b)$

9) Zur automatischen Brandbekämpfung wurden drei Sensoren in einem Raum angebracht. Melden mindestens zwei der drei Sensoren eine Rauchentwicklung, so schaltet sich die Sprinkleranlage ein. Entwirf eine Leitwerttabelle, eine Schaltfunktion und stelle diese mittels Digitalsimulator dar!

Zeichencodierung

Eine **Zeichenkodierung (englisch Character encoding, kurz Encoding)** erlaubt die eindeutige Zuordnung von Schriftzeichen (i. A. Buchstaben oder Ziffern) und Symbolen innerhalb eines Zeichensatzes. In der elektronischen Datenverarbeitung werden Zeichen über einen Zahlenwert kodiert, um sie zu übertragen oder zu speichern. Der deutsche Umlaut Ü wird zum Beispiel im ISO-8859-1-Zeichensatz mit dem Dezimalwert 220 kodiert. Im EBCDIC-Zeichensatz kodiert derselbe Wert 220 die geschweifte Klammer }. Zur richtigen Darstellung eines Zeichens muss also die Zeichenkodierung bekannt sein; der Zahlenwert allein reicht nicht aus.

Zahlenwerte aus Zeichenkodierungen lassen sich auf verschiedene Art speichern oder übertragen, z. B. als Morsezeichen, verschieden hohe Töne (Faxgerät), verschieden hohe Spannungen.

Geschichte des Character Encoding

Mit der Entwicklung des Computers begann die Umsetzung der im Grunde schon seit dem Baudot-Code verwendeten binären Zeichenkodierung in Bit-Folgen, bzw. intern meist in verschiedene elektrische Spannungswerte als Unterscheidungskriterium, ganz analog zu der bisher zur Unterscheidung der Signalwerte genutzten Tonhöhe oder Signaldauer.

Um diesen Bit-Folgen darstellbare Zeichen zuzuordnen, mussten Übersetzungstabellen, sogenannte Zeichensätze, engl. Charsets, festgelegt werden. 1963 wurde eine erste **7-Bit-Version des ASCII-Codes durch die ASA (American Standards Association)** definiert, um eine **Vereinheitlichung der Zeichenkodierung** zu erreichen. Obwohl IBM an der Definition mitgearbeitet hatte, führte man 1964 einen eigenen **8-Bit-Zeichencode EBCDIC** ein. Beide finden bis heute in der Computertechnik Verwendung.

Da für viele Sprachen jeweils unterschiedliche diakritische Zeichen benötigt werden, mit denen Buchstaben des lateinischen Schriftsystems modifiziert werden, gibt es für viele Sprachgruppen jeweils eigene Zeichensätze. Die **ISO** hat mit der **Normenreihe ISO 8859 Zeichenkodierungen für alle europäischen Sprachen** (einschließlich Türkisch) und Arabisch, Hebräisch sowie Thai standardisiert.

Das **Unicode Consortium** schließlich veröffentlichte 1991 eine erste Fassung des gleichnamigen Standards, der es sich zum Ziel gesetzt hat, alle Zeichen aller Sprachen in Codeform zu definieren. **Unicode** ist gleichzeitig die **internationale Norm ISO 10646**.

Bevor ein Text elektronisch verarbeitet wird, muss der verwendete Zeichensatz und die Zeichenkodierung festgelegt werden. Dazu dienen beispielsweise folgende Angaben:

Definition des Zeichensatzes in einer HTML-Seite

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Definition des Zeichensatzes in den Kopfzeilen (Headern) einer E-Mail oder eines HTTP-Pakets

```
Content-Type: text/plain; charset="ISO-8859-1"
```

ASCII - American Standard Code for Information Interchange

Der **American Standard Code for Information Interchange (ASCII, deutsch „Amerikanischer Standard-Code für den Informationsaustausch“)** ist eine **7-Bit-Zeichenkodierung**; sie entspricht der US-Variante von ISO 646 und dient als Grundlage für spätere, auf mehr Bits basierende Kodierungen für Zeichensätze.

Der ASCII-Code wurde zuerst am 17. Juni 1963 von der **American Standards Association (ASA) als Standard** ASA X3.4-1963 gebilligt und 1967/1968 wesentlich sowie zuletzt im Jahr 1986 von ihren Nachfolgeinstitutionen aktualisiert. Die Zeichenkodierung definiert **128 Zeichen**, bestehend aus **33 nicht druckbaren** sowie **95 druckbaren Zeichen**.

ASCII-Code Tabelle

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

www.VirtualUniversity.ch

Fakten

- sehr verbreiteter Standard, u.a. in PCs (Hardware-Ebene)
- von ISO genormt

- ursprünglich 7-Bit-Code, also 128 Zeichen
- davon einige nach nationalem Bedarf abgewandelt z.B. deutsche Umlaute statt [ ] { } \ |
- unterschiedliche 8-Bit-Erweiterungen mit zusätzlichen Zeichen im Bereich 128 – 255
- Erweiterungen oft problematisch bei älteren Rechnern, im Internet1) etc. → deshalb E-Mail, HTTP etc. auf 7 Bit beschränkt (→ 2. Sem.)

UTF8 - UCS Transformation Format

UTF-8 (Abk. für 8-Bit UCS Transformation Format, wobei UCS wiederum Universal Character Set abkürzt) ist die am weitesten verbreitete Kodierung für Unicode-Zeichen (Unicode und UCS sind praktisch identisch). Die Kodierung wurde im September 1992 von Ken Thompson und Rob Pike bei Arbeiten am Plan-9-Betriebssystem festgelegt.

UTF-8 ist in den **ersten 128 Zeichen (Indizes 0–127) deckungsgleich mit ASCII** und eignet sich mit in der Regel nur **einem Byte Speicherbedarf** für Zeichen vieler westlicher Sprachen besonders für die Kodierung englischsprachiger Texte, die sich im Regelfall ohne Modifikation daher sogar mit nicht-UTF-8-fähigen Texteditoren ohne Beeinträchtigung bearbeiten lassen, was einen der Gründe für den Status als **De-facto-Standard-Zeichenkodierung des Internets** und damit verbundener Dokumenttypen darstellt. Im Oktober 2017 verwendeten **89,9 % aller Websites UTF-8**

In anderen Sprachen ist der Speicherbedarf in Byte pro Zeichen größer, wenn diese vom ASCII-Zeichensatz abweichen: Bereits die **deutschen Umlaute erfordern zwei Byte**, ebenso griechische oder kyrillische Zeichen. Zeichen fernöstlicher Sprachen und von Sprachen aus dem afrikanischen Raum belegen dagegen bis zu 4 Byte je Zeichen. Da die Verarbeitung von UTF-8 als Multibyte-Zeichenfolge wegen der notwendigen Analyse jedes Bytes im Vergleich zu Zeichenkodierungen mit fester Byteanzahl je Zeichen mehr Rechenaufwand und für bestimmte Sprachen auch mehr Speicherplatz erfordert, werden abhängig vom Einsatzszenario auch andere UTF-Kodierungen zur Abbildung von Unicode-Zeichensätzen verwendet: Microsoft Windows als meistgenutztes Desktop-Betriebssystem verwendet intern als Kompromiss zwischen UTF-8 und UTF-32 etwa UTF-16 Little Endian

Fakten Unicode (UTF-8, UTF-16, UTF-32)

- neuere Kodierung, ebenfalls ISO-standardisiert 1993 und heutzutage Standard
- Ziel: Berücksichtigung möglichst vieler Sprachen und ihrer Eigenheiten
- Buchstaben-, Silben-, und Ideogrammsprachen
- Schreibrichtungen (links-rechts, rechts-links, oben-unten)
- außerdem diverse Sonderzeichen, mathematisch-technische Symbole, Diakritika, geometrische Formen, Pfeile, Piktogramme u.v.m.
- dafür 16-Bit-Darstellung (erlaubt 65.536 Zeichen)
- Erweiterung auf 32 Bit für künftigen Bedarf
- Standard auf unixoiden Betriebssystemen

4) Hardware

Hardware ist der Oberbegriff für die **physischen Komponenten (die elektronischen und mechanischen Bestandteile)** eines datenverarbeitenden Systems, als **Gegenteil zu Software (den Programmen und Daten)**

Wortherkunft

„Hardware“ kommt ursprünglich aus dem Englischen und bedeutet übersetzt Eisenwaren.

Hardware vs. Software

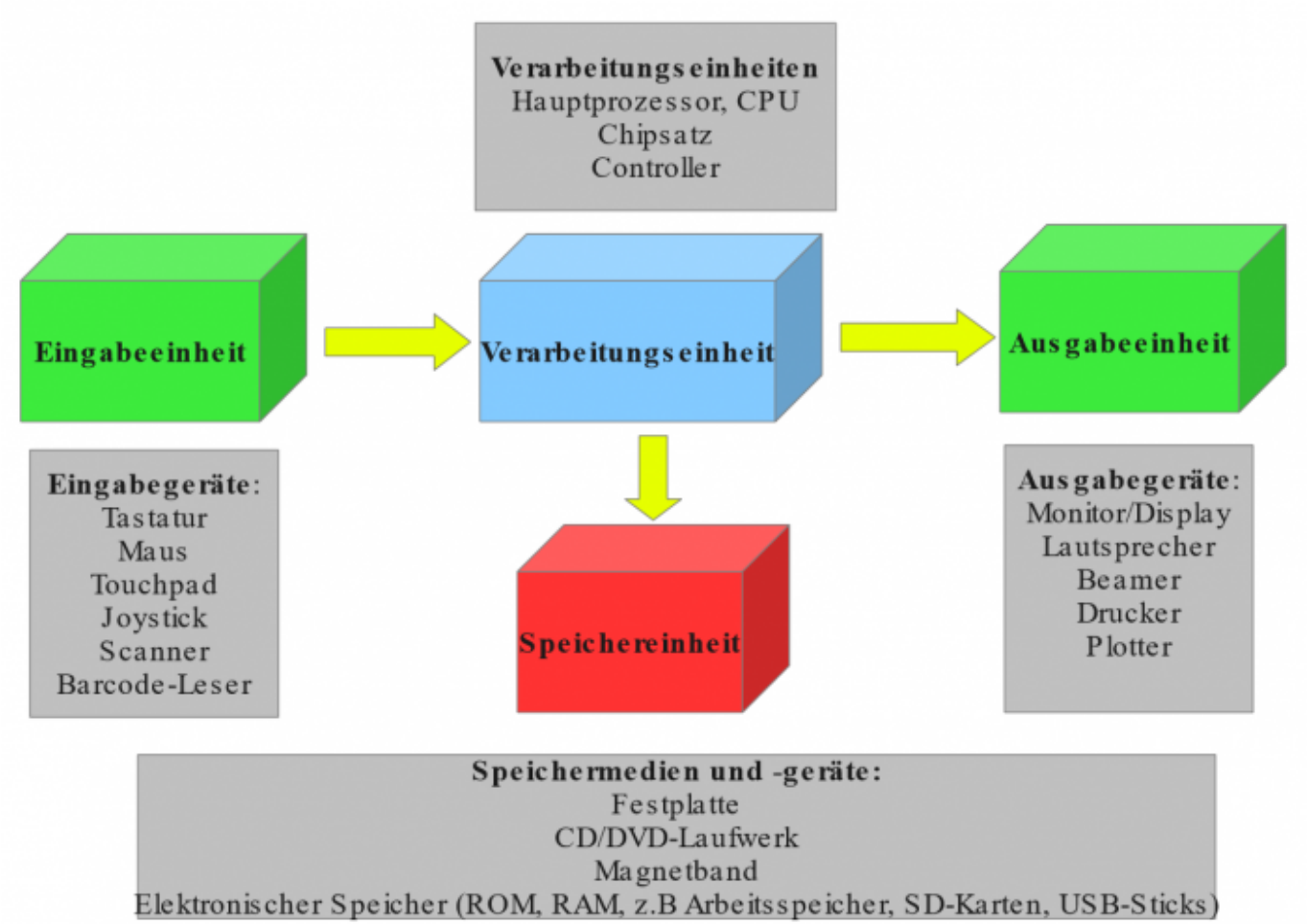
- Hardware = sind alle greifbaren/sichtbaren Elemente eines PCs
- Software = Programme und Daten, also nicht greifbare Elemente eines PCs

Überblick

4.1) EVA/IPO Prinzip

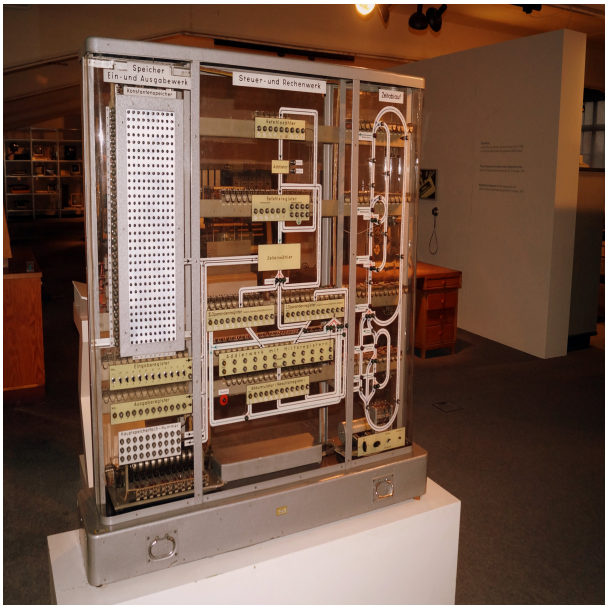
Das **EVA**-Prinzip ist ein Grundprinzip für die Datenverarbeitung und beschreibt die Reihenfolge in der Daten verarbeitet werden.

Eingabe (**I**nput) - **V**erarbeitung (**P**rocess) - **A**usgabe (**O**utput)



4.2) Von-Neumann-Architektur (VNA)

Die **V**on-**N**eumann-**A**rchitektur ist ein Modell für Computer und bietet die Grundlage für alle heutigen Computer. Das Modell wurde von Johann von Neumann im Jahr 1945 entwickelt. Heute ist Johann von Neumann unter seinem amerikanischen Namen John von Neumann bekannt.

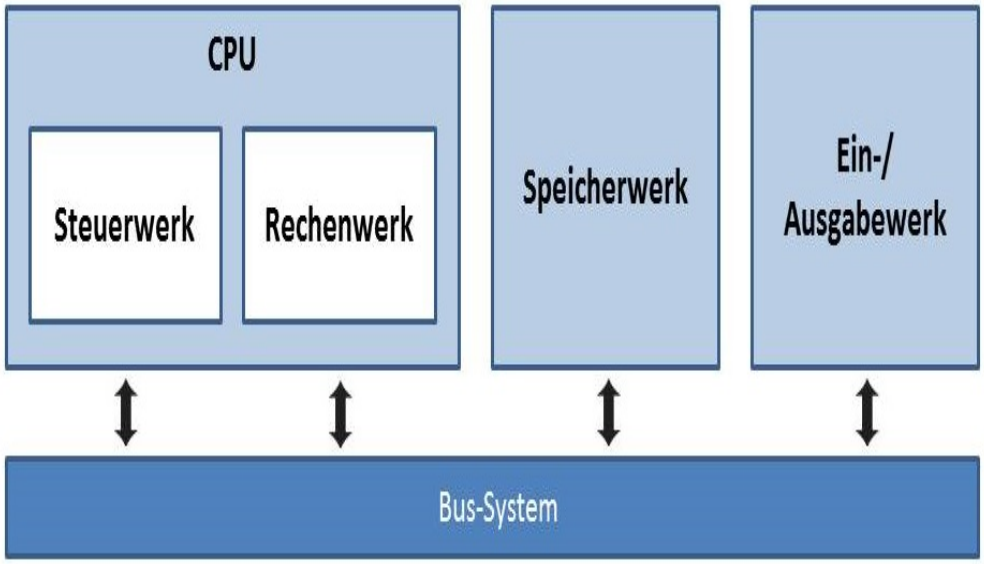


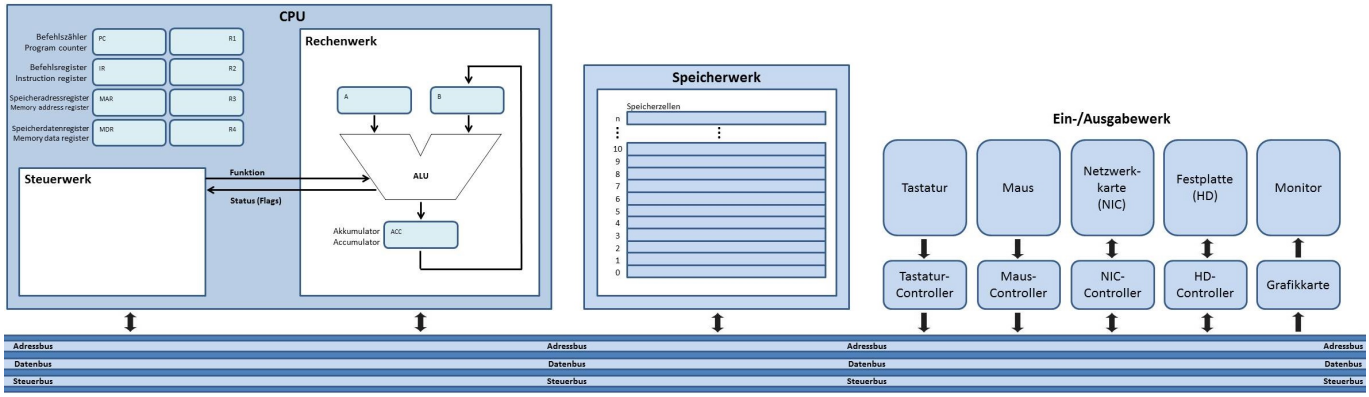
Er revolutionierte die bisherigen Computer, da durch seine Architektur verschiedene Programme auf derselben Hardware laufen konnten.

Komponenten

Ein Von-Neumann-Rechner beruht auf folgenden Komponenten, die bis heute in Computern verwendet werden:

- ALU (Arithmetic Logic Unit) – Rechenwerk, selten auch Zentraleinheit oder Prozessor genannt, führt Rechenoperationen und logische Verknüpfungen durch. (Die Begriffe Zentraleinheit und Prozessor werden im Allgemeinen in anderer Bedeutung verwendet.)
- Control Unit – Steuerwerk oder Leitwerk, interpretiert die Anweisungen eines Programms und verschaltet dementsprechend Datenquelle, -senke und notwendige ALU-Komponenten; das Steuerwerk regelt auch die Befehlsabfolge.
- Bussystem: Dient zur Kommunikation zwischen den einzelnen Komponenten (Steuerbus, Adressbus, Datenbus)
- Memory – Speicherwerk speichert sowohl Programme als auch Daten, welche für das Rechenwerk zugänglich sind.
- I/O Unit – Eingabe-/Ausgabewerk steuert die Ein- und Ausgabe von Daten, zum Anwender (Tastatur, Bildschirm) oder zu anderen Systemen (Schnittstellen).





Register

- **Befehlszähler (Program Counter - PC)**

Enthält die Speicheradresse vom Speicherwerk des aktuellen Befehls. (Startadresse = 0x0000). Wird nach jedem Befehl um 1 erhöht.

- **Befehlsregister (Instruction Register - IR)**

Speichert den vom Speicherwerk zurückbekommenen Befehl.

- **Speicheradressregister (Memory Address Register - MAR)**

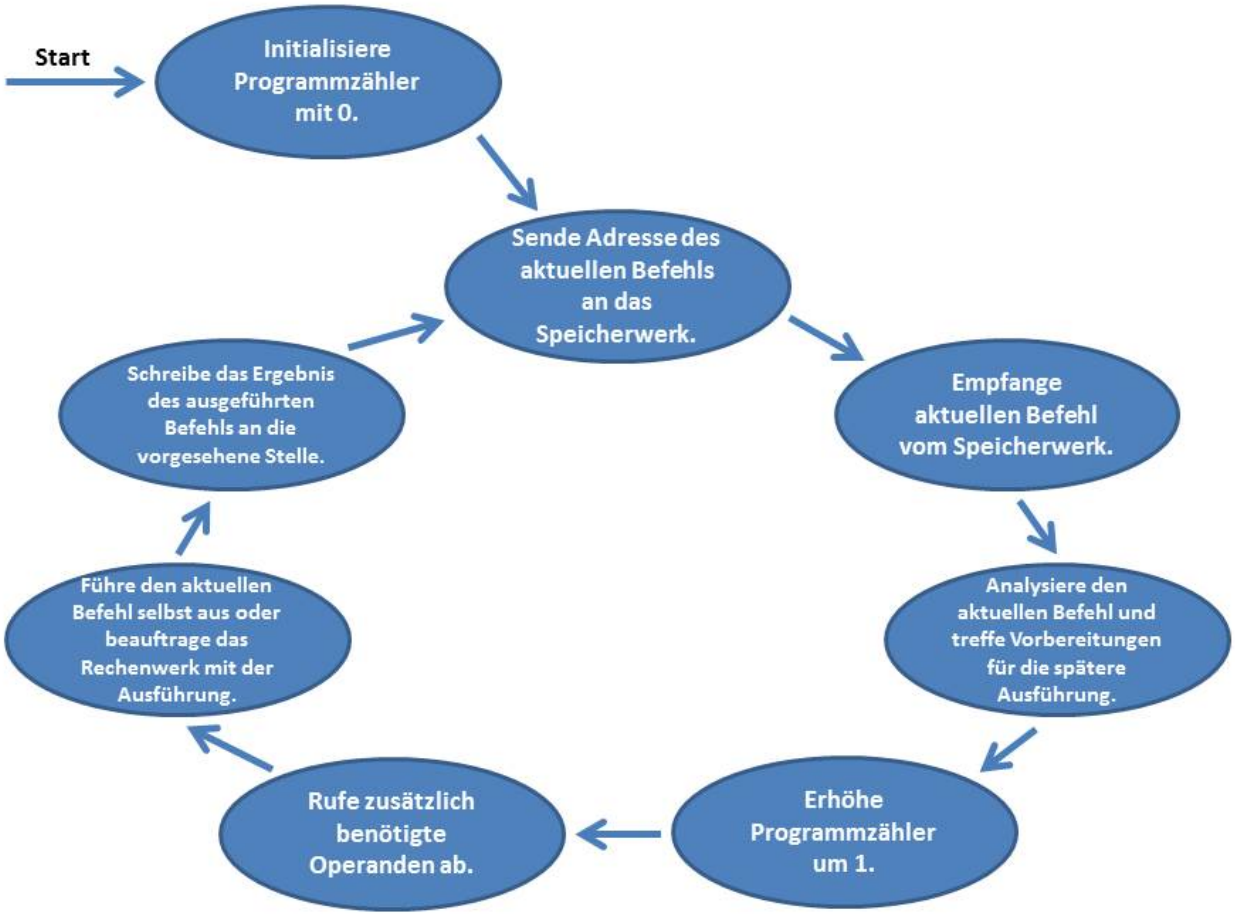
Ausschließlich für die Kommunikation zwischen Steuerwerk und Rechenwerk. Im MAR legt das Steuerwerk jeweils die Adresse ab, welche im Speicherwerk angesprochen werden soll.

- **Speicherdatenregister (Memory Data Register - MDR)**

Ausschließlich für die Kommunikation zwischen Steuerwerk und Rechenwerk. Bei einem Lesezugriff auf die Speicherzelle wird der vom Speicherwerk über den Datenbus bereitgestellte Wert im MDR abgelegt und kann von hier aus weiter verarbeitet werden. Bei einem Schreibzugriff muss sich im MDR der zu schreibende Wert befinden, so dass er über den Datenbus an das Speicherwerk übermittelt werden kann.

Prozesszyklus

1. Initialisiere das Befehlszählerregister (Program Counter- PC) mit 0 (Start)
2. Sende Adresse des aktuellen Befehls zum Speicherwerk
3. Empfange aktuellen Befehl vom Speicherwerk und speichere diesen in das Befehlsregister (Instruction Register - IR)
4. Analysiere aktuellen Befehl und treffe Vorbereitungen für die spätere Ausführung (Welcher Befehl und was ist dazu notwendig?)
5. Erhöhe den Befehlszähler (PC) um 1
6. Rufe zusätzlich benötigte Operanden ab (z.B.: Befehl ADD OP1 OP2)
7. Führe den Befehl selbst (Steuerwerk) aus oder beauftrage das Rechenwerk für die Ausführung
8. Schreibe das Ergebnis des ausgeführten Befehls an die vorgesehene Stelle



Arbeitsweise des Steuer- und Rechenwerks

Arbeitsweise des Speicherwerks

Befehlszähler und Befehlsregister im Zusammenspiel mit dem Bus-System

Arbeitsweise des Steuerwerks

Die 7 Prinzipien der Von-Neumann-Architektur

- Rechner besteht aus fünf Funktionseinheiten
- Struktur des Rechners ist unabhängig vom zu bearbeitenden Problem. Zur Lösung eines Problems muss Programm im Speicher abgelegt werden.
- Programme, Daten und Ergebnisse werden im selben Speicher abgelegt.
- Der Speicher ist in fortlaufenden nummerierten Zellen unterteilt. Über die Adresse einer Speicherzelle kann auf den Inhalt zugegriffen werden.
- Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgende Speicherzellen abgelegt.
- Durch Sprungbefehle kann von der gespeicherten Befehlsreihenfolge abgewichen werden.
- Es gibt zumindest
    - arithmetische Befehle (Addition, Subtraktion, Multiplikation)
    - logische Befehle (EQUAL, NOR, AND, OR)
    - Transportbefehle, z.B. von Speicher zu Rechenwerk und für Ein- und Ausgabe
- Alle Daten (Befehle, Adressen, usw.) werden binär kodiert

4.3) Hardware-Komponenten

Grundbestandteile

- Stromversorgung
- 4.3.1) Motherboard/Mainboard - Hauptplatine
- 4.3.2) Central Processing Unit (CPU) - zentrale Recheneinheit (Prozessor)
- 4.3.3) Random Access Memory (RAM) - Arbeitsspeicher

Massenspeicher

- 4.3.4) Datenspeicher


SAMSUNG
Solid State Drive
SAMSUNG
Solid State Drive


ASUS
QuieTrack

Erweiterungskarten

- 4.3.5) Erweiterungskarten

Peripheriegeräte

Eingabegeräte

- 4.3.6) Eingabegeräte

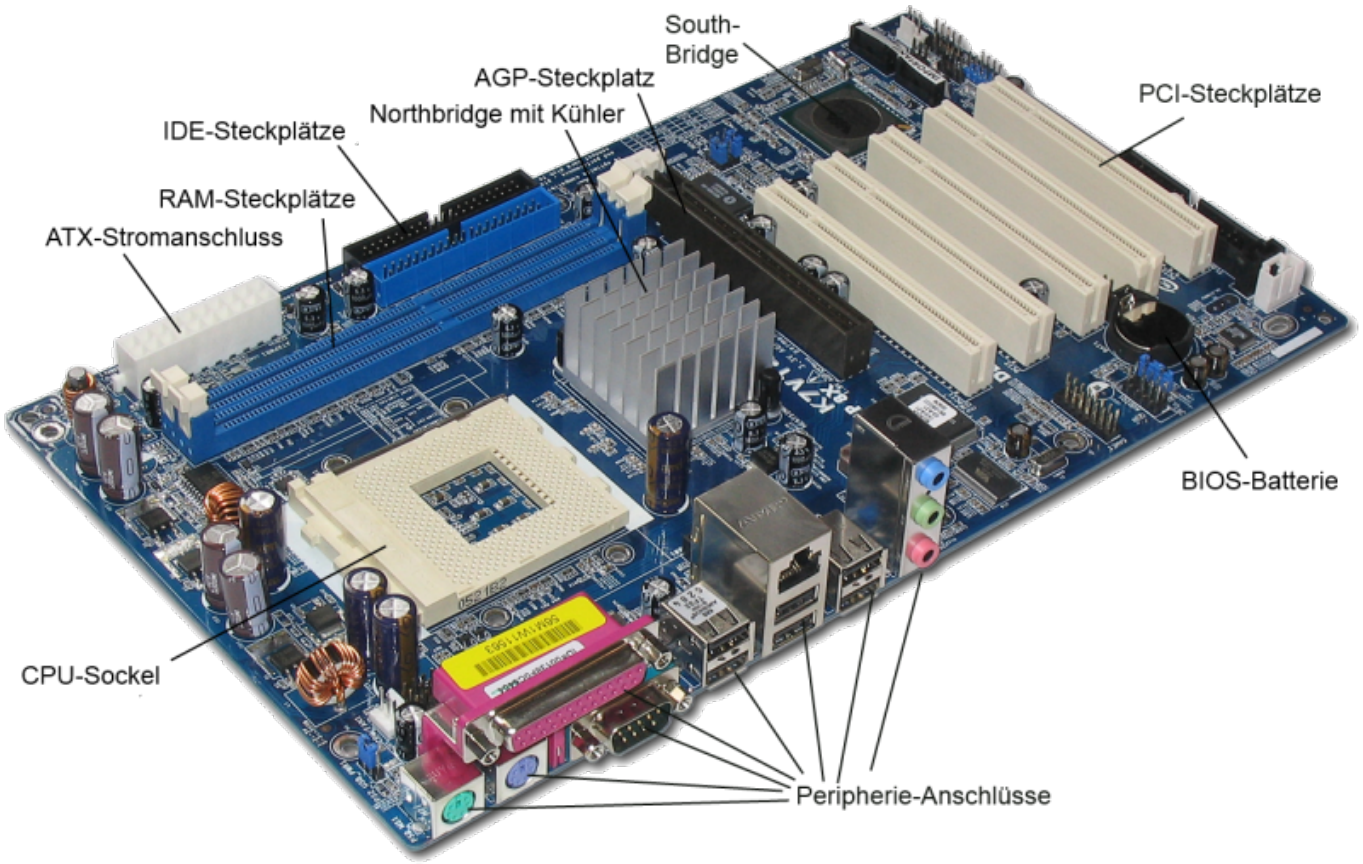
Ausgabegeräte

- 4.3.7) Ausgabegeräte

4.3.1) Motherboard / Mainboard / Hauptplatine

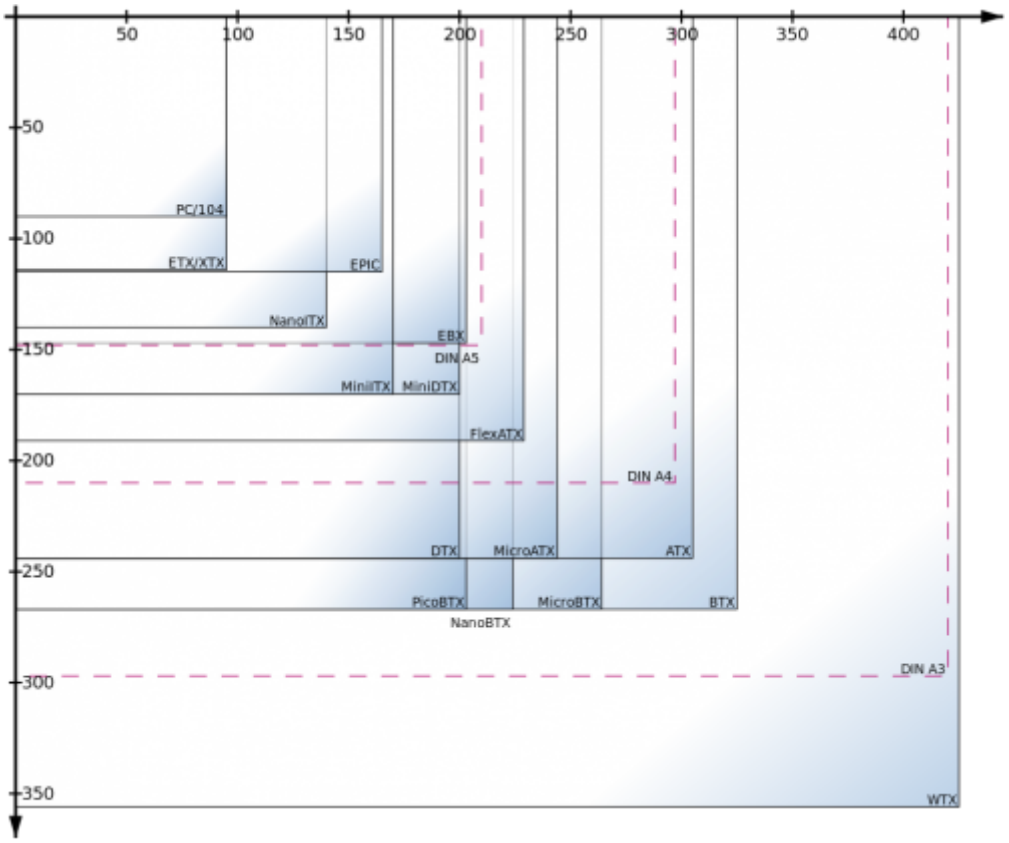
Das Motherboard ist der **Grundbaustein eines Computers**. Das Motherboard ist die Platine, auf der alle Systemkomponenten eines Computers eine **physikalische und logische Verbindung** erhalten. Die wichtigsten und einige leistungsbeeinflussenden Bauteile sind fest auf dieser Platine miteinander verbunden.

Die **Ausstattung des Motherboards bestimmt die System-Leistung, Erweiterbarkeit und Zukunftsfähigkeit eines Computersystems**. Die meisten Motherboards sind auf eine bestimmte Anwendung mit einem bestimmten Prozessor zugeschnitten. Man kann also nicht jeden beliebigen Prozessor auf jedem Motherboard verwenden. Der **Einsatz eines Prozessors hängt vom Motherboard bzw. vom Prozessorsockel und dem Chipsatz ab**.



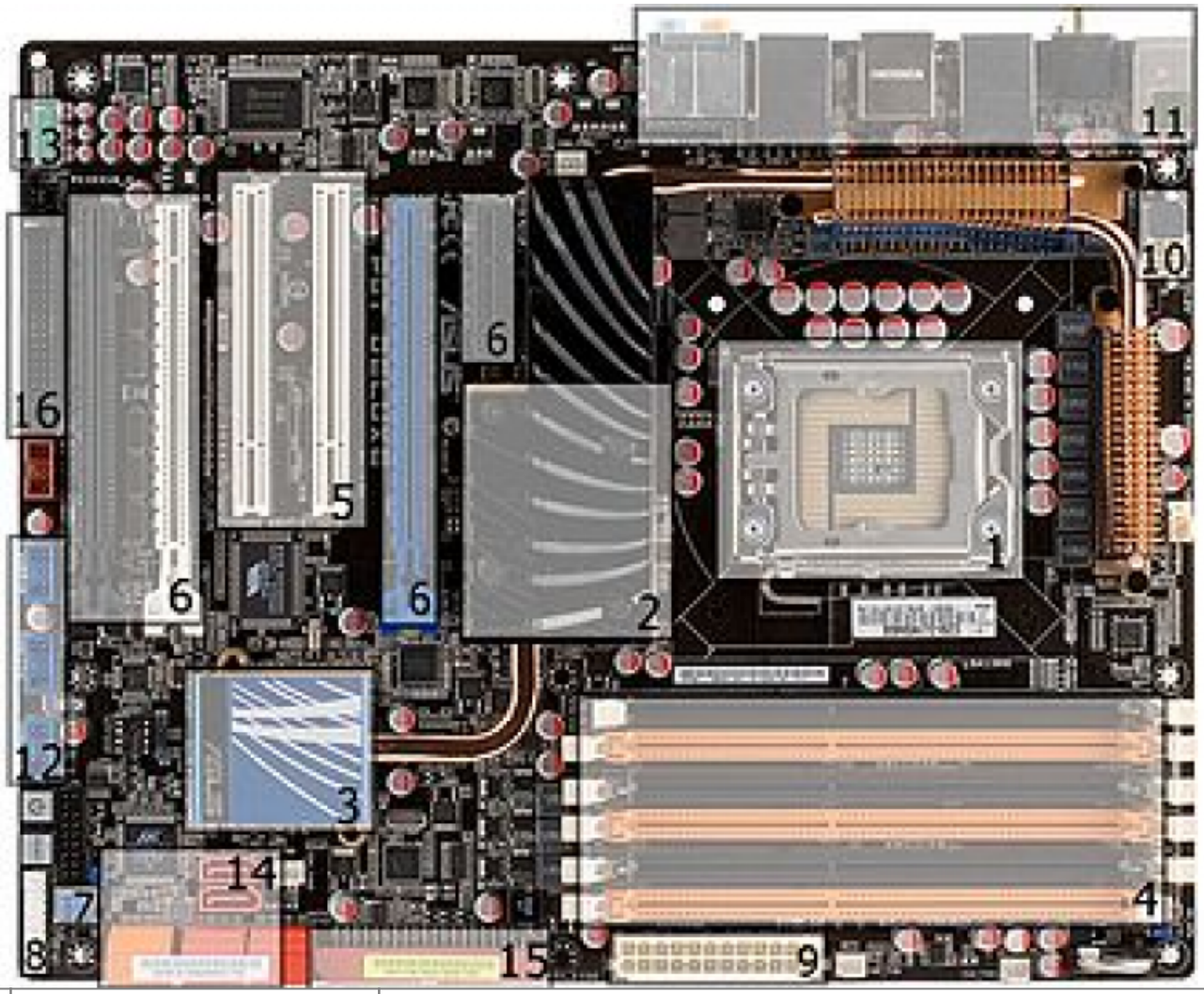
Bauformen

Das Format der Hauptplatinen wird nach dem **Formfaktor** unterschieden. Seit 1995 ist das **ATX-Format aktuell**. Es löste das AT-Format ab und brachte diverse Umstellungen an Gehäusen und Netzteilen mit sich. Es existieren diverse Variationen von ATX und AT, um auch kompaktere Geräte ohne proprietäre Formate bestücken zu können, etwa Baby AT oder µATX. Mini-ITX, Flex-ATX und Micro-ATX passen in ATX-Gehäuse. **Steckplätze, Schrauben** und das **Fenster für I/O-Shield** befinden sich an einheitlichen oder **derselben Position**.



Komponenten eines Motherboards

Auf einem Motherboard können je nach Bauform, Ausstattung und Integrationsdichte unterschiedliche Systemkomponenten zu finden sein.



Nr	Komponente	Beschreibung
1	**Prozessorsockel**	Der Prozessor, der auch als Hauptprozessor bezeichnet wird, hat auf dem Motherboard einen eigenen Steckplatz bzw. Sockel (z.B.: LGA775 oder LGA1155).
2	**Chipsatz - Northbridge**	Der Chipsatz ist das Bindeglied zwischen den einzelnen Systemkomponenten eines Computers. Egal was in einem Computer passiert, der Chipsatz hat immer damit zu tun. Er sorgt dafür, dass alle Komponenten über unterschiedliche Schnittstellen miteinander kommunizieren können. Dabei werden unterschiedliche Spannungspegel, Taktfrequenzen und Protokolle berücksichtigt und untereinander umgewandelt.
3	**Chipsatz - Southbridge**	
4	**RAM-Steckplatz**	Steckplätze für den RAM (z.B.: hier für DDR3)
5	**PCI-Steckplatz**	für Erweiterungskarten (z.B.: Audiokarte)
6	**PCI-Express-Steckplatz**	2 PCIe-x16 Slots (Grafikkarte) & 1 PCIe-x1-Slot (Erweiterungskarten)
7	**Jumper**	Kurzschlussstecker zur Aktivierung und Deaktivierung von Einstellungen (z.B. Übertakten)
8	**Anschlüsse Frontblende**	Hier werden die Schalter (Power, Reset, Lampen für FP) von vorne an das Mainboard angebunden
9	**24-poliger ATX-Connector**	Haupstromversorgung für das Mainboard
10	**8-poliger-ATX-Connector**	Anschluss für die Stromversorgung der CPU
11	**Externe Anschlüsse**	Anschlüsse ragen aus dem Gehäuse nach hinten hinaus und dienen als Anschluss für die Peripheriegeräte (Maus, Tastatur,..)
12	**Interne USB-Anschlüsse**	USB-Anschlüsse für die Vorderseite des Gehäuses

Nr	Komponente	Beschreibung
13	**AAFP-Soundanschluss**	An diesem Anschluss werden die Sound Ein- und Ausgänge der Front des Gehäuses angeschlossen
14	**Serial-ATA-Anschlüsse**	Ermöglicht die Verbindung von Festplatten oder optischen Laufwerken mit der Hauptplatine
15	**IDE-Anschluss**	Ermöglicht die Verbindung von Festplatten oder optischen Laufwerken mit der Hauptplatine
16	**Floppy-Disk-Stecker**	Anschluss des Diskettenlaufwerks

4.3.2) CPU - Central Processing Unit

Der Prozessor, Hauptprozessor oder die CPU ist heutzutage das **Herzstück eines jeden elektronischen Geräts**. Er wird in Smartphones, Taschenrechnern und in Computern, für die er eigentlich erfunden wurde, eingesetzt. Eine Welt ohne diese Rechengenies ist undenkbar.
Die bekanntesten Prozessoren stammen von den Unternehmen **Intel und AMD**.

In einem **Computersystem** kann es **mehrere Prozessoren** geben. Wenn man vom Prozessor spricht, dann ist damit in der Regel immer der Hauptprozessor gemeint. Wegen seiner **zentralen Stellung** wird die Bezeichnung **„Zentrale Verarbeitungseinheit“** verwendet.
Der Hauptprozessor ist die **Funktionseinheit** in einem Computer, der **die eigentliche Verarbeitungsleistung erbringt**. Der Hauptprozessor ist für die **Informationsverarbeitung und die Steuerung der Verarbeitungsabläufe zuständig**. Dazu holt sich der Prozessor **aus dem Speicher nacheinander die Befehle** und **veranlasst die Informationsverarbeitung**.
Neben dem Hauptprozessor gibt es noch **weitere Prozessoren**, die den Hauptprozessor von der Arbeit entlasten. Der **Grafikprozessor (GPU)** ist ein solcher Prozessor.

Prozessor-Hersteller

Es gibt 2 marktführende Prozessore Hersteller, **AMD** und **INTEL**.
Seit jeher gilt der Prozessor-Hersteller AMD als ewiger Zweiter im Kampf um die Marktanteile im PC-Prozessor-Markt. Dazu kommt, dass AMD lange Zeit von dem am stärksten wachsenden Markt, den Notebooks, abgekoppelt war. Mit wenigen Ausnahmen, waren die meisten AMD-Prozessoren in Low-Cost-PCs für Privatkunden verbaut. In den Bereichen, in denen richtig Geld verdient werden konnte, war Intel immer besser unterwegs.

Das Problem von AMD und INTEL ist, dass die Prozessoren von den beiden Herstellern jeweils komplett andere Sockel gebrauchen und somit ein Umstieg von einem Hersteller zum anderen auch meistens mit einem neuen Motherboard-Kauf einhergeht.

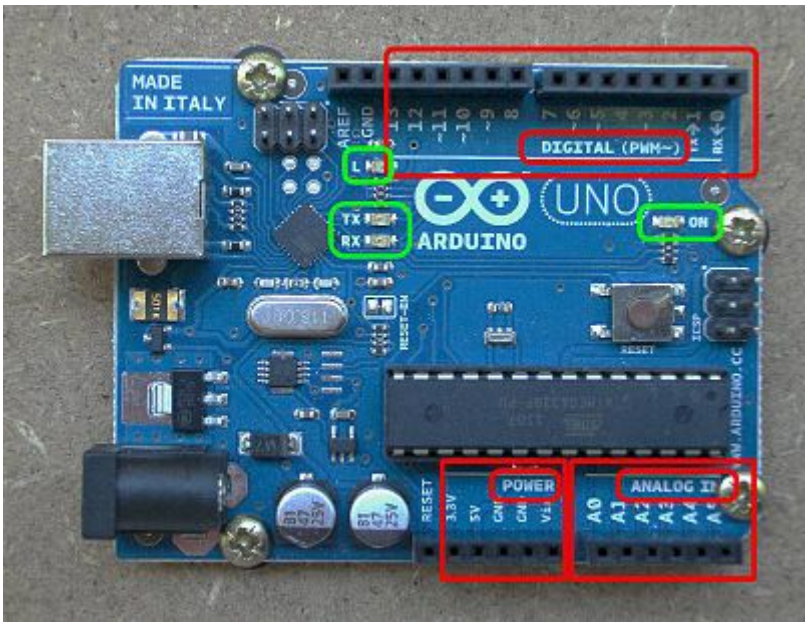
Mikroprozessor

Der Mikroprozessor ist ein Prozessor, der vollständig in einem einzigen Schaltkreis untergebracht ist. Der Prozessor im Personal Computer ist ein solcher Mikroprozessor.

⇒ Ein **Mikroprozessor ist die Vereinigung von Rechenwerk und Steuerwerk eines Computers mit den dazugehörigen Daten-, Adress- und Steuerleitungen auf einem Chip**

Mikrocontroller

Ein Mikrocontroller ist ein **Prozessor**, der über **zusätzliche digitale Ein- und Ausgänge** verfügt und für Steuerungsaufgaben vorgesehen ist. Er wird bereits **als vollständiger Computer** angesehen, der im Embedded-Bereich eingeordnet wird.



Moderne Prozessortechnik

Die **Entwicklung der Prozessoren** verlief lange Zeit nach einem scheinbar einfachen Gesetz. Man **optimierte den internen Aufbau**, **verkleinerte die Strukturen**, **senkte die Spannung**, **erhöhte die Taktfrequenz** oder verbesserte den Herstellungsprozess.
Schon war die **nächste Prozessor-Generation** geboren. Doch diesem Spiel sind enge Grenzen gesetzt. So muss mit **zunehmender Chipgröße das Taktsignal immer längere Wege** zurücklegen. Damit der Zeitunterschied der Taktflanken im akzeptablen Bereich liegt, muss der **Takttreiber immer leistungsstärker** werden. Dadurch **erhöht er die Verlustleistung** des Prozessors. Deshalb wird die **Chipfläche regelmäßig verkleinert**. Dabei werden zwischen den Schaltelementen **immer dünnere Verbindungen** eingesetzt. Dadurch **steigt der Widerstand der Verbindungen** und die Signale bewegen sich immer langsamer. Das führt unter Umständen dazu, dass die Signallaufzeit unter der Verarbeitungszeit der Gatter liegt. Die **physikalischen Grenzen zeigten sehr schnell**, dass **insbesondere die Taktfrequenz nicht unendlich weit gesteigert werden konnte**. Die Taktfrequenz bestimmt unter anderem die entstehende Verlustleistung und damit **die Lebensdauer des Prozessors**. Deshalb wurden sehr bald andere leistungssteigernde Techniken entwickelt.

Während vor Jahren die Geschwindigkeit eines Prozessors immens wichtig war, ist heute ein **ausgewogenes System aus Prozessor, Arbeitsspeicher und Chipsatz** das Maß **für einen schnellen Computer**. Immer weniger Anwendungen benötigen die volle Rechenleistung eines aktuellen Prozessors. Aus diesem Grund bieten die heutigen Prozessoren viel mehr als nur reine Rechengeschwindigkeit. Sie haben **mehrere Kerne, nutzen Befehlssatzerweiterungen, intelligente Zwischenspeicher, verfügen über Virtualisierungstechnik und Grafikfunktionen**.

Multi-Core-Prozessoren

Multi-Core bedeutet, dass **in einem Prozessor mehrere Prozessor-Kerne** eingebaut sind. Man bezeichnet diese Prozessoren als Multi-Core- oder Mehrkern-Prozessoren. Rein **äußerlich unterscheiden sich** Multi-Core-CPUs **nicht von Single-Core-CPUs**. Der Rechenkern ist bei Multi-Core-CPUs einfach mehrfach vorhanden. Innerhalb des **Betriebssystems** wird der **Multi-Core-Prozessor wie mehrere Recheneinheiten** behandelt.

Je nach Anzahl der Kerne gibt es abgewandelte Bezeichnungen, die darauf hindeuten, wie viele Kerne im Prozessor integriert sind.

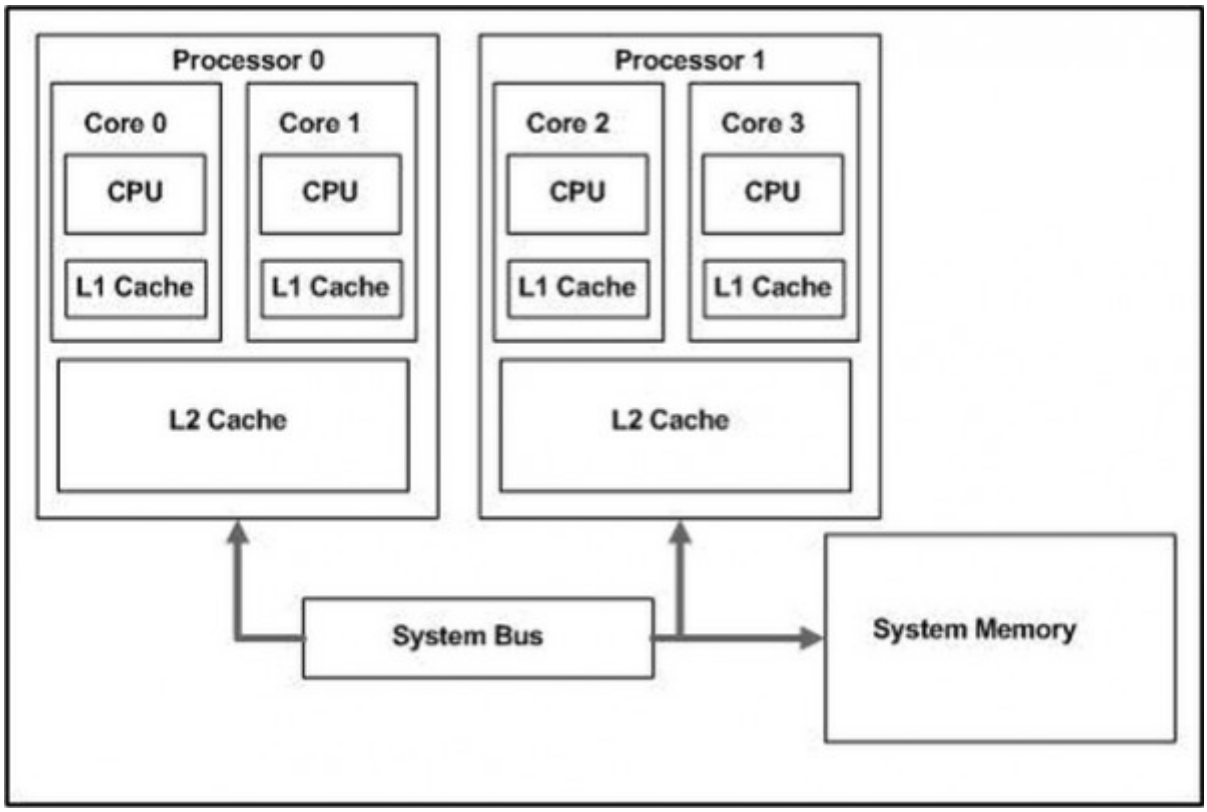
- Dual-Core (2)
- Triple-Core (3)
- Quad-Core (4)
- Hexa-Core (6)
- Octa-Core (8)
- ....

Unterschied Mehrprozessorsysteme vs. Mehrkernprozessor

Ein Mehrprozessorsystem ist ein Computersystem, das **mehr als einen Prozessorsockel** auf der

Multi-Prozessor-Hauptplatine besitzt, und in dem mehr als einer dieser Sockel auch bestückt ist.

Ein Mehrkernprozessor ist ein Prozessor mit mehreren Kernen.



In der obigen Abbildung ist erkennbar, dass dies 2 Prozessoren mit jeweils 2 Kernen sind. Also für das Betriebssystem insgesamt 4 Kerne zur Verfügung stehen!

Vom Takt-orientierten Prozessor zum Mehr-Kern-Prozessor

Um einen Prozessor schneller zu machen war die **Taktfrequenz lange Zeit der maßgebliche Faktor**, um **mehr Rechenleistung** aus einem Prozessor heraus zu bekommen. Leider hat die **Erhöhung der Taktfrequenz auch Nachteile**, die zu Folgeproblemen führen, die nur sehr schwer zu lösen sind.

- höhere Leistungsaufnahme
- höhere Wärmeentwicklung
- umfangreichere Kühlmaßnahmen
- lautere Computer durch aktive Kühlung

Schon eine **minimale Leistungssteigerung führt zu einem dramatisch höheren Energieverbrauch**. Die damit verbundene Leistungsaufnahme verhält sich proportional zur Taktfrequenz. Zusätzliche Transistoren und kleinere Halbleiterstrukturen erhöhen zusätzlich die Wärmeentwicklung. Die **Anforderungen an die Kühlung** sind mit den herkömmlichen Mitteln **nicht mehr zu leisten**. Gleichzeitig **verringert sich die Stabilität und Lebensdauer des Prozessors**.

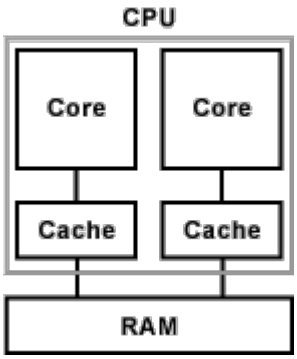
Eine **Alternative** zu immer höheren Taktraten sind **mehrere Rechenkerne**. Das bedeutet **mehr Leistung bei gleichzeitig geringerer Leistungsaufnahme**. Dabei werden die **einzelnen Kerne weit geringer getaktet**, als ein einziger Rechenkern. Insgesamt **steigt jedoch die**

Leistungsfähigkeit des Prozessors mit jedem weiteren Rechenkern.

Problematik

Grundsätzlich kann man die **Rechenleistung mehrerer Kerne nicht „addieren“**. Das würde voraussetzen, dass sich die vorliegenden Rechenaufgaben parallelisieren lassen. Die effektive Nutzung mehrerer Kerne erfordert es, dass die **Software so geschrieben ist, dass sie Daten parallel verarbeitet und so zum Beispiel mehrere Kerne gleichzeitig genutzt werden**. Das heißt, ein Problem bzw. die **Ausführung eines Programms** muss **in mehrere kleine Teilaufgaben zerlegt werden**, damit diese auf mehrere Kerne verteilt werden können. Doch leider sind die üblichen Anwendungen nicht auf parallele Ausführung ausgelegt und meist auch nicht notwendig.

Auswirkungen auf die Prozessor- und Computer-Architektur



In einer **Multi-Core-Architektur müssen** sich mehrere Rechenkerne die **vorhandenen Resourcen teilen**. Zum Beispiel **Arbeitsspeicher, Schnittstellen** usw. **Je mehr Kerne** ein Prozessor hat, **desto mehr Speicher und mehr Bandbreite zum Speicher ist erforderlich**. Aus diesem Grund ist der **Speicher-Controller nicht mehr im Chipsatz, sondern im Prozessor verankert**.

Arbeitsspeicher / Hauptspeicher

Gleichzeitig besteht das Problem, dass sich die **Rechenkerne darüber abstimmen** müssen, **wer gerade welche Daten im Cache** hält. In der Regel arbeiten die Rechenkerne mit einer **hierarchischen Cache-Struktur**. Dabei bekommt **jeder Kern einen eigenen L1- und L2-Cache**. Den **L3-Cache müssen sich die Kerne teilen**. Ein Cache-Kohärenz-Protokoll sorgt dafür, dass sich die Prozessorkerne bei der Nutzung des L3-Caches nicht in die Quere kommen.

Typische Anwendungen für Single-Core-Prozessoren

- Textverarbeitung
- Browser
- E-Mail
- Instant-Messaging
- Virenscanner
- MP3-Player

- einfache Bildbearbeitung (Schneiden, Skalieren, Farbkorrektur)

Typische Anwendungen für Multi-Core-Prozessoren

Gut parallelisierbare Anwendungen sind Schneiden von 4K-Videos, Computergrafiken mit Raytracing erzeugen oder aufwendige Softwareprojekte kompiliert. In diesen Fällen können im Prozessor nicht genug Kerne enthalten sein.

- CAD
- Simulation
- HD-Video
- Compiler
- 3D-Rendering
- professionelle Audio-Bearbeitung
- professionelle Bildbearbeitung
- Videoschnittprogramme

4.3.4) Datenspeicher

Ein Datenspeicher dient zum **Speichern von Daten bzw. Informationen**. **Als Datenspeicher dienen Datenträger**. Der Begriff Speichermedium wird auch als Synonym für einen konkreten Datenträger verwendet. Bei der Datenspeicherung geht es darum, Daten jeglicher Art auf Datenträger zu schreiben bzw. speichern und von dort auch wieder zu lesen.

Es wird unterschieden zwischen:

4.3.4.1) Magnetische Massenspeicher

Der magnetische Datenspeicher besteht aus einem **Datenträger**, der aus einem **magnetisierbaren Material** besteht. Das kann auf **Bänder, Karten, Papier oder Platten** aufgebracht werden. Der Datenträger wird dann mittels einem **Lese-Schreib-Kopf gelesen und beschrieben**.
Typischerweise wird das Speichermedium im **Datenträger gedreht oder rotiert**, während der **bewegliche oder unbewegliche Lese-Schreib-Kopf über das Speichermedium geführt** wird.
Die Daten können auf dem Speichermedium sowohl **digital als auch analog** gespeichert werden.

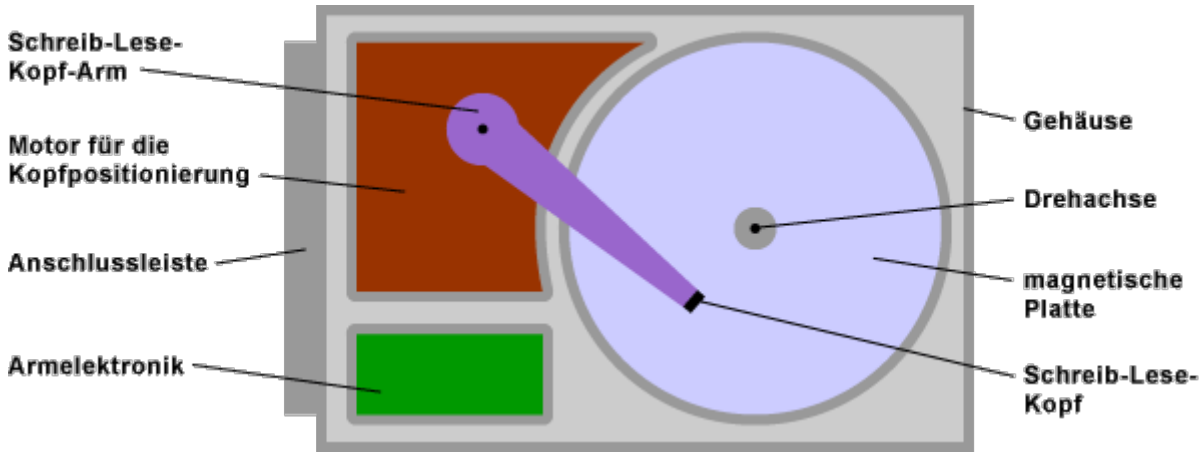
4.3.4.1.1) Festplatten - HDD (Hard Disk Drive)

Festplatten sind **Massenspeicher**, die auf einem **magnetischen Datenträger** beruhen, auf dem die **Daten fest gespeichert** werden und **auch ohne Energieversorgung gespeichert** bleiben. Festplatten werden typischerweise in einen Computer eingebaut. Auf Festplatten werden alle Daten und Anwendungen eines Computers gespeichert.

Der Begriff **Festplatte (engl. Harddisk bzw. Hard Density Disc, HDD)** kommt durch die Unterscheidung zur inzwischen veraltete Diskette (engl. Floppydisk, FDD), die als wechselbarer Datenträger lange Zeit vor der Festplatte verwendet wurde.
Festplatten **ersetzen den Festwertspeicher ROM und die Diskette**. Festplatten können **viel mehr Daten speichern als Disketten**. Im Gegensatz zum ROM kann man Daten von einer Festplatte nicht nur lesen, sondern auch darauf schreiben und jederzeit ändern. 1954 wurden Festplatten erstmals industriell eingesetzt.

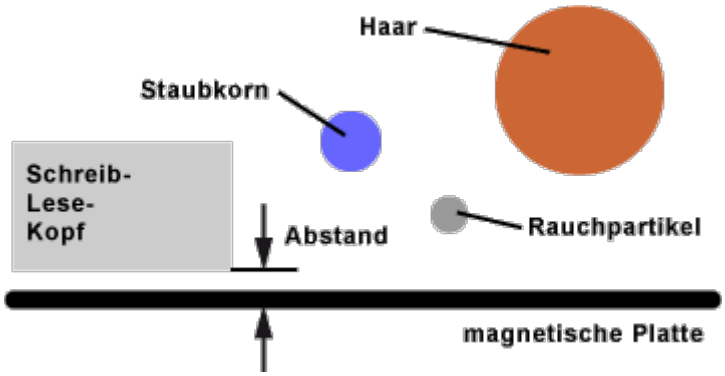
Festplatten unterscheiden sich in ihrer **Zuverlässigkeit, Geschwindigkeit, Speicherkapazität** und damit auch im Preis. Für unterschiedliche Anwendungsfälle haben die Festplatten-Hersteller **unterschiedliche Festplatten-Typen**. Im Vergleich zu anderen Datenspeichern haben Festplatten ein sehr gutes Preis-Leistungsverhältnis. Das bedeutet, **geringe Kosten pro Byte**. Momentan liegt die **größtmögliche Speicherkapazität bei 6 bis 8 TByte pro Einzellaufwerk**. Wobei es diese Festplatten nur für spezielle Anwendungen gibt. Bis **2020** soll die **Speicherkapazität auf 20 TByte** pro Einzellaufwerk steigen.

Aufbau einer Festplatte



In einem **geschlossenen Metallgehäuse** befinden sich alle Komponenten, die für das Funktionieren der Festplatte wichtig sind. Um das **Eindringen von Staub und Schmutz** in das Gehäuse zu

verhindern, sind die einzelnen Teile einer Festplatte in ein **nahezu luftdichtes Gehäuse** verschlossen. Als einziger Kontakt zum Computersystem dient eine **Anschlussleiste für eine Schnittstelle (IDE, SATA, SCSI, etc.)**, über die die Daten übertragen werden. Der eigentliche **Datenspeicher** einer Festplatte ist eine oder mehrere **Metallscheiben**, die mit einem **magnetisierbaren Material beschichtet** sind. Um die Speichermenge zu erhöhen liegen mehrere Scheiben übereinander. Die Scheiben sind um eine Drehachse mittels Halteklammern befestigt und dadurch voneinander getrennt. **Zwischen den Metallscheiben greifen die Schreib-Lese-Kopf-Arme** hinein. Auf diesen Armen befindet sich eine federnde Aufhängung. Auf dieser ist der Kopf befestigt, der zum Lesen und Schreiben der Daten dient.



Der **Abstand zwischen Kopf und Scheibe** ist **geringer, als ein Haar, Staub- oder Rauchpartikel**. Die **Berührung von Kopf und Scheibe führt zum Head-Crash**, der wiederum zum **Datenverlust** führt. Dabei wird der Datenträger zerstört, was die Festplatte unbrauchbar macht. Normalerweise können sich Kopf und Platte nicht berühren. Denn bei hohen Rotationsgeschwindigkeiten, bei der sich eine Festplatte dreht, bildet sich ein Luftpolster zwischen Kopf und Platte.

Geschwindigkeit einer Festplatte

Je schneller eine Festplatte ist, desto flüssiger laden die Daten und laufen die Programme. Besonders beim Start von Betriebssystem und Anwendungen spürt der Anwender eine schnelle Festplatte. Und nur mit einem **schnellen Massenspeicher speichert ein Computer große Datenmengen** mühelos. Folgende **vier Kriterien** machen eine Festplatte schnell:

1) Umdrehungsgeschwindigkeit

Die Umdrehungsgeschwindigkeit wird in Umdrehungen in der Minute (UPM, U/Min) angegeben. Je geringer die Drehzahl, desto länger dauert der Zugriff auf zufällig ausgewählte Sektoren. Üblich sind die Umdrehungsgeschwindigkeiten 10.000, 7.200 und 5.400 U/Min.

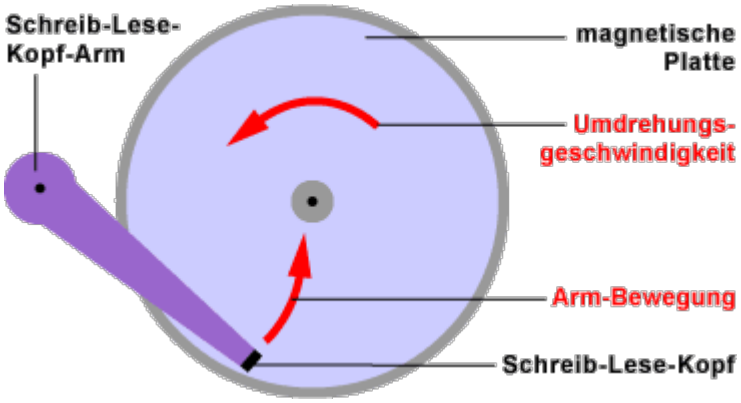
2) Anzahl der Datenscheiben

Je mehr Datenscheiben eine Festplatte hat, desto höher ist ihre Kapazität. Doch auch die Lese- und Schreibgeschwindigkeit steigt, wenn der Datenstrom über mehrere Lese- und Schreibköpfe summiert wird.

3) Datendichte auf den Datenscheiben

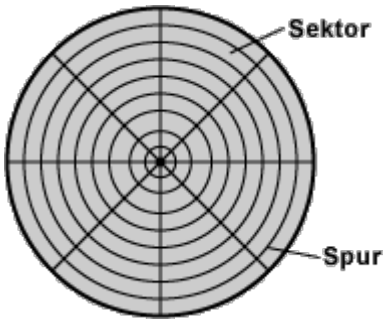
Je höher die Datendichte, desto mehr Bit ziehen pro Sekunde am Schreib-Lese-Kopf vorbei und können gelesen oder geschrieben werden.

4) Zugriffszeit / Access Time / Datenzugriffszeit



Die Zugriffszeit gibt an, wie lange es dauert, bis die Festplatte die gewünschten Daten auf ihren Datenschreiben gefunden hat und die ersten Bit liefert. Die Zugriffszeit ist die Summe aus Such- und Latenzzeit und wird hauptsächlich von der Umdrehungsgeschwindigkeit der Festplatte bestimmt. Je schneller sich die Platte dreht, desto geringer ist diese Zeit.

Organisation der Daten



Damit die Daten, die auf den magnetischen Platten abgelegt sind, wieder gefunden werden, ist es notwendig eine **Einteilung der Magnetscheiben** vorzunehmen. Als erster Schritt wird eine herstellerseitige Low-Level-Formatierung vorgenommen. Dazu werden auf den Scheiben **Spuren** angelegt. Es handeln sich dabei um **konzentrische Kreise**, die auf allen Magnetscheiben gleich sind. Die Spuren werden vom **äußeren Rand der Platte nach innen**, beginnend bei 0, **durchnummeriert**. Der Abstand der Spuren, die Spurdichte, bestimmt die Speichermenge. Diese Dichte wird in Spuren pro Zoll (Tracks per Inch, TPI) angegeben.
Die **Anordnung mehrerer Spuren** (durch **übereinander gelagerte Magnetscheiben**) nennt man **Zylinder**. Die **Spuren** werden wiederum in **kleinere Abschnitte** eingeteilt. Dieser Abschnitt nennt sich Sektor und entspricht einem **Kreisausschnitt**.

Spuren

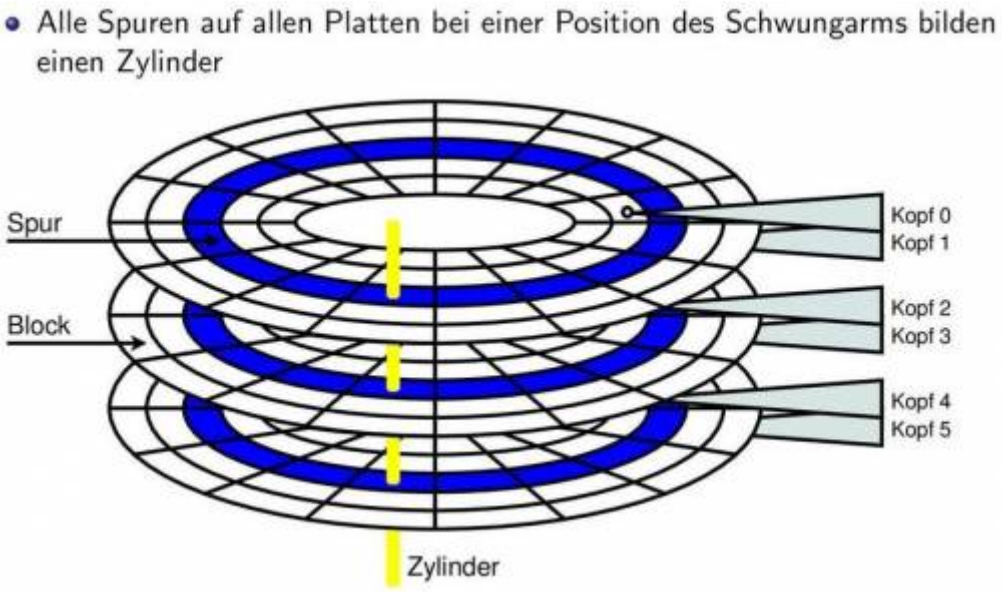
Die **Spuren** sind konzentrische Kreispfade, die auf jede Scheibenseite geschrieben werden, wie auf einer Schallplatte oder einer CD. Die Spuren werden mit einer Nummer identifiziert, die mit Spur 0 am äußeren Rand einsetzt.

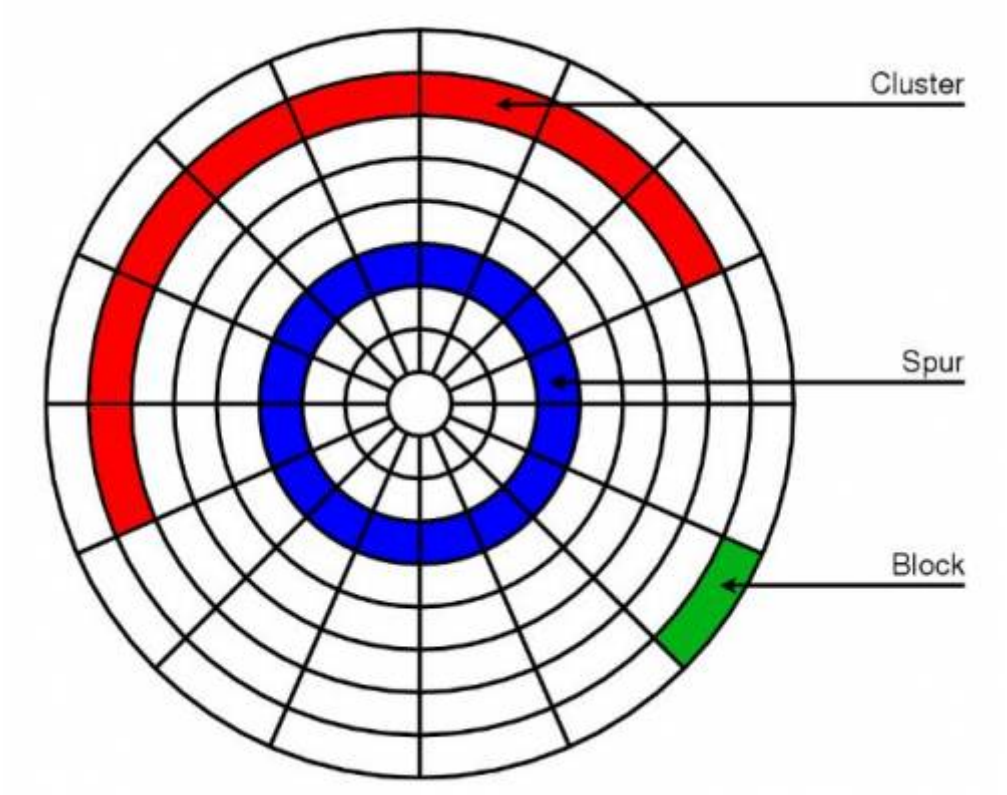
Zylinder

Der Spurensatz, der auf allen Seiten der Platte im gleichen Abstand von der Mitte angelegt ist, wird als "**Zylinder**" bezeichnet. Hardware und Software arbeiten häufig mit diesen Zylindern.

Sektoren

Die Spuren sind in Bereiche, sogenannte "**Sektoren**" oder „**Blöcke**" unterteilt, in denen eine festgeschriebene Datenmenge gespeichert wird. Die Sektoren werden in der Regel für eine Speicherkapazität von 512 Byte (ein Byte besteht aus 8 Bit) formatiert.





4.3.4.1.2) Bandlaufwerke / Tapes

Streamer sind Bandlaufwerke zum Beschreiben und Lesen von Tapes, Magnetbändern oder einfach nur Bändern.

Die Bezeichnung Streamer wird vom Aufzeichnungsverfahren abgeleitet. Die Aufzeichnung findet linear statt, weshalb sie sich nur für das Sichern bzw. Speichern zusammenhängender Daten eignen. Zum Beispiel bei einer Datensicherung. Da Backups archiviert werden, ist die Datenmengen bis zu 10 mal größer als die eigentliche Datenmenge. Magnetbänder bzw. Tapes sind das einzige Speichermedium, das große Datenmenge vernünftig (Zeit, Aufwand, Kosten) sichern kann. Da es sich bei den Tapes um Wechselspeicher handelt, benötigt das gelagerte Speichermedium keinen Strom. Obwohl optische Speichermedien, wie CD, DVD und Blu-ray in der Computertechnik als Massenspeicher populär sind, hat das an der Bedeutung von Tapes und Bändern als externe Speichermedien im professionellen Bereich nichts geändert.

4.3.4.2) Optischer Datenspeicher

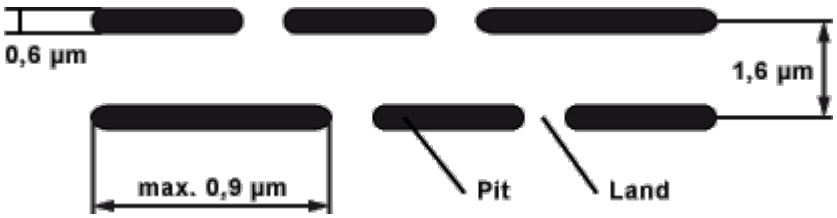
Optische Datenspeicher **speichern Daten und Informationen optisch veränderter Form**. Bei der optischen Speicherung **nutzt** man die **Reflexions- und Beugungseigenschaften des Speichermediums** aus. Zum Lesen und Schreiben der Daten auf den Datenträger wird ein **Laserstrahl** verwendet. Die **Speicherform ist ausschließlich digital**.

4.3.4.2.1) CD-ROM

Die CD-ROM ist ein Massenspeicher mit **650 bis 879 MByte** Speicherkapazität. Als Vorlage diente die **Compact-Disc** (CD), die bereits Anfang der 80er Jahre als digitales Medium für Musik entwickelt wurde. Im aufkommenden Multimedia-Zeitalter wurde es nötig die umfangreichen Computerdaten sinnvoll zu speichern. Dazu reichte die **alte Diskette mit 1,4 MByte** nicht mehr aus.

Aufbau einer CD-ROM

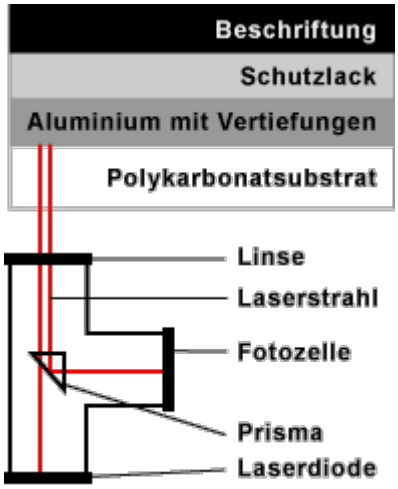
Der **Durchmesser** einer CD-ROM beträgt **12 cm**. Die CD besteht aus **Polycarbonat** mit einer Dicke von **1,2 mm**. Darin liegt eine **reflektierende Aluminiumschicht**. Sie ist **einseitig mit Daten beschrieben**.



Die Daten sind auf einer **spiralförmigen, von innen nach außen, führenden Spur in Vertiefungen gespeichert**. Diese **Vertiefungen werden Pits (Täler)** genannt. Sie sind 0,2 µm tief, 0,6 µm breit und maximal 0,9 µm lang. Der Abstand zwischen den Spuren beträgt 1,6 µm. Die **Zwischenräume zwischen den Pits werden Lands** genannt.

Die Daten auf einer CD-ROM werden in **gleich großen Sektoren** gespeichert. Diese Sektoren werden auf einer einzigen Spur, die spiralförmig angelegt ist, untergebracht.

Lesen der Daten



Die **Laserdiode erzeugt einen Laserstrahl**, der zusätzlich durch eine **Linse gebündelten** wird. Dieser Laserstrahl trifft auf die Unterseite der CD-ROM, **durchdringt die Schutzschicht** und **trifft auf die Aluminium-Schicht**. Dort wird er **reflektiert**. Das **Prisma leitet** den **Laserstrahl zur Fotozelle** weiter. Der Laserstrahl, der die Fotozelle trifft, erzeugt eine **geringe elektrische Spannung**.
Trifft der Laserstrahl auf einen **Übergang zwischen Pit und Land**, dann wird der **Laserstrahl abgelenkt**, was eine **andere elektrische Spannung** in der Fotozelle verursacht. Auf diese Weise entsteht eine **Reihe von Einsen und Nullen.**

Lesegeschwindigkeit

Eine Audio-CD wird mit einer konstanten Datenrate von 150 kByte/s ausgelesen. Dieser Datendurchsatz wird mit „1x“ bezeichnet. Laufwerke, die einen 52-fachen Datendurchsatz haben, erreichen 7800 kByte/s.

Umdrehungs-geschwindigkeit	max. Übertragungsrate	mittlere Zugriffszeit
1x	150 kByte/s	400 ms
2x	300 kByte/s	300 ms
4x	600 kByte/s	150 ms
8x	1200 kByte/s	100 ms
12x	1800 kByte/s	100 ms
16x	2400 kByte/s	90 ms
24x	3600 kByte/s	90 ms
32x	4800 kByte/s	85 ms
44x	6600 kByte/s	85 ms
52x	7800 kByte/s	-

Fehler

Jede CD-ROM erhält im Laufe der Zeit und durch unsachgemäße Behandlung fehlerhafte Stellen (Kratzer, Risse). Diese Stellen bringen ein CD-ROM-Laufwerk zum häufigen Wiederholen des Lesevorgangs mit reduzierter Geschwindigkeit. Arbeitet die Fehlerkorrektur gut, sind diese Vorgänge seltener. Wobei sich das positiv auf die Lesegeschwindigkeit auswirkt.

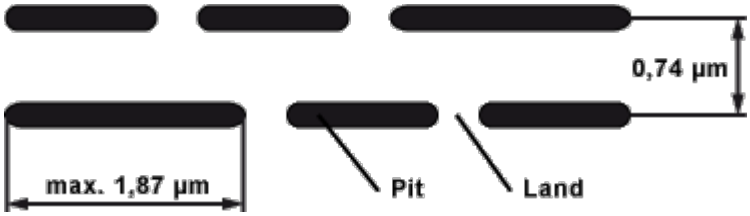
Formate

Format	Beschreibung
CD+G	Compact Disc and Graphics für Audio und Grafik mit einer sehr geringen Verbreitung.
CD+Midi	Compact Disc mit digitaler Schnittstelle für Musikinstrumente für Audio- und Midi-Informationen mit sehr geringer Verbreitung.
CD-DA	Compact Disc Digital Audio ist die klassische Musik-CD mit 74 Minuten Spieldauer.
CD-Extra	Enhanced Music Compact Disc ist die verbesserte Musik-CD. Nachfolger der CD-Plus für Audio und Daten.
CD-I	Compact Disc Interactive ist die Multimedia-CD mit der Möglichkeit in den Ablauf aktiv einzugreifen (interaktiv) für Audio, Video, Fotos und Spiele.
CD-MO	Magnetooptische Daten-CD mit geringer Verbreitung.
CD-R	Compact Disc Recordable mit der Möglichkeit einmal mit Audio oder Daten zu beschreiben.
CD-RW	Compact Disc Rewriteable mit der Möglichkeit mehrfach mit Audio oder Daten zu beschreiben.
CD-ROM	Compact Disc Read Only Memory ist ein Nurlesespeicher für Daten.
CD-ROM XA	CD-ROM eXtended Architecture ist ein Nurlesespeicher für Daten, Text, Grafik, Bild, Audio und Video.
Photo-CD	Foto-CD für die Ablage von digitalen Bildern in Echtfarben.
Video-CD	Video-CD für Filme mit max. 74 Minuten Spieldauer im Audio- und Videoformat MPEG. Videoqualität im VHS-Format.

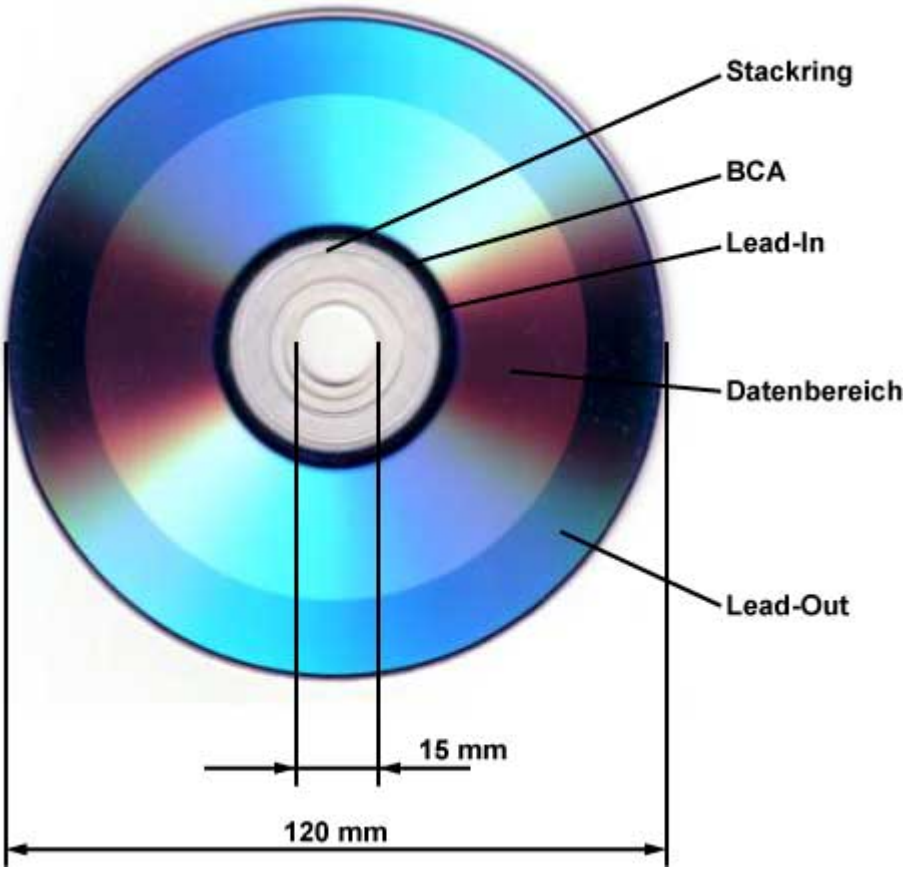
4.3.4.2.2) DVD - Digital Versatile Disc

Die DVD ist ein Massenspeicher, ähnlich wie die CD-ROM. Mit einer Speicherkapazität von **4,7 GByte (Single Layer) bzw. 8,5 GByte (Dual Layer)** konnten viel mehr Daten gespeichert werden, als auf einer CD-ROM. Parallel dazu wurden **beschreibbare und wiederbeschreibbare DVD-Varianten** entwickelt. Ursprünglich stand DVD für Digital Video Disc. Ein Großteil der Anwendungen liegen jedoch im Computerbereich.

Aufbau einer DVD



Die Daten auf einer DVD werden in **Erhöhungen (Pits) und Vertiefungen (Lands)** in der Reflektionsschicht gespeichert. Der **Übergang von einem Pit zu einem Land** bzw. umgekehrt wird als **logische 1** gewertet. Findet **kein Wechsel** statt, wird das als **logische 0** gewertet. Um mehr Daten auf ein gleich großes Medium wie die CD unterzubringen, werden die Pits und deren **Abstand zueinander verkürzt** und die Daten auf **vier übereinanderliegende Schichten** verteilt. Dadurch entsteht ein Speichermedium, das bis zu **17 GByte** speichern kann.
Um die kleineren Pits lesen zu können, ist die **Wellenlänge des roten Lasers auf 635 nm bzw. 650 nm** verkürzt. Zum Vergleich: die **CD-ROM** wird mit einem **infraroten Laser und einer Wellenlänge von 780 mm** gelesen.



Speicherkapazität

Die Speicherkapazitäten einer DVD sind eine **Vorgabe der Filmindustrie**. Diese geht von einer Standardlänge pro Film von 135 Minuten aus. Durch Addieren der Datenrate pro Sekunde kommt man mit Bild und Ton in mehreren Sprachen und Untertitel auf 4,69 MBit. In der Minute sind das 281,4 MBit. Bei 135 Minuten ergibt das eine Datenmenge von ca. 38 GBit, was umgerechnet etwa **4,7 GByte** entspricht. Das ist die Speicherkapazität einer **DVD-5 (Single Layer)**. Diese wird für Video-DVDs verwendet. Wenn die Speicherkapazität nicht ausreicht, dann greift man auf die **doppellagige DVD-9 (Dual-Layer)** zurück.

Typen

DVD-Typ	Seiten	Schichten	Speicherkapazität		Besonderheit
			binär	dezimal	
DVD-5	1	1	4,38 GibiByte	4,7 GByte	-
DVD-9	1	2	7,92 GibiByte	8,5 GByte	-
DVD-10	2	1	8,76 GibiByte	9,4 GByte	-
DVD-14	2	1/2	12,3 GibiByte	13,2 GByte	-
DVD-18	2	2	15,84 GibiByte	17 GByte	-
DVD-plus	2	1	4,28 GibiByte	4,7 GByte	DVD-5/CD-ROM
DVD-1	1	1		1,5 GByte	8 cm Durchmesser
DVD-2	1	2		2,7 GByte	8 cm Durchmesser
DVD-3	2	1		2,9 GByte	8 cm Durchmesser
DVD-4	2	2		5,3 GByte	8 cm Durchmesser

Formate

Formate	Beschreibung
DVD-Video	Mehrstündige Videos mit hochqualitativen, interaktiven Videosequenzen mit mehreren Soundspuren und Untertiteln. Die Video-Daten werden mit MPEG2 komprimiert.
DVD-Audio	Musik mit 24-Bit-Auflösung und 96 kHz Sampling-Frequenz.
DVD-ROM	Speichermedium für Computer, ähnlich der CD-ROM.
DVD-R	Einmalbeschreibbare DVD wie die CD-R.
DVD-R DL	Einmalbeschreibbare DVD mit zwei Aufnahmeschichten, wie bei einer DVD-9
DVD-RW	Wiederbeschreibbare DVD des DVD-Forums, ähnlich der CD-RW.
DVD+R	Einmalbeschreibbare DVD und Konkurrenz zur DVD-R.
DVD+R DL	Einmalbeschreibbare DVD mit zwei Aufnahmeschichten, wie bei einer DVD-9
DVD+RW	Wiederbeschreibbare DVD und Konkurrenz zur DVD-RW.
DVD-RAM	Wiederbeschreibbare DVD ähnlich wie DVD-RW und DVD+RW.
Recordable CSS	Filme mit dem „DVD Download“-Siegel lassen sich auf spezielle „DVD-R for CSS Managed Recording“ brennen.

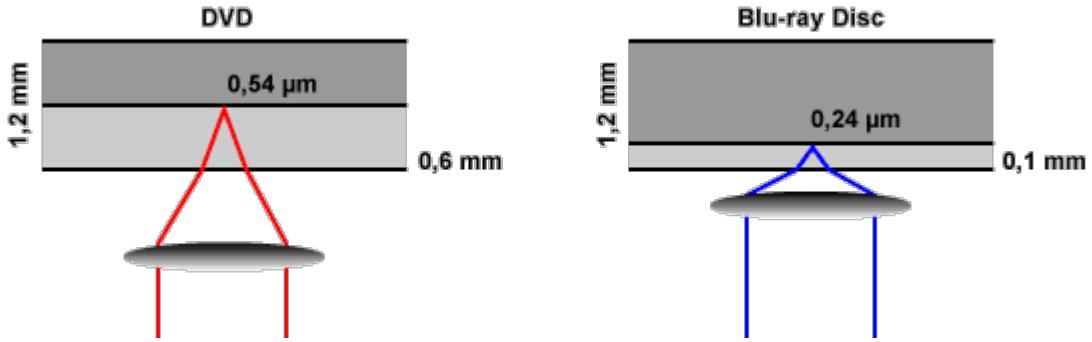
4.3.4.2.3) BD - Blu-ray Disc

Die Blu-ray Disc (BD) wurde für die **Speicherung von hochauflösenden Videodaten (HDTV)** entwickelt. Die Speicherung von **mehr als zwei Stunden Audio- und Videodaten** ist das maßgebliche Kriterium bei der Entwicklung gewesen. Die Speicherkapazität einer **DVD reicht für diese Datenmenge nicht aus**.

Die **Bezeichnung** Blu-ray Disc ist von der **Farbe des Lasers abgeleitet**. Weil sich eine Farbe nicht als Warenzeichen schützen lässt, hat man einfach das „e“ aus dem Wort „Blue“ weggelassen. Deshalb findet man ab und zu fälschlicherweise die Bezeichnung „Blue-ray Disc“.

Aufbau der Blu-ray Disc

Mit Hilfe eines blauen Lasers mit einer **Wellenlänge von 405 nm können 25 GByte** auf einer einlagigen, **einseitig beschreibbaren Blu-ray Disc** gespeichert werden. **Zweilagige Blu-ray Discs** haben sogar die **doppelte Kapazität (50 GByte)**. **Denkbar wären Medien mit 8 Schichten, also 200 GByte**. Allerdings wäre die Produktion dieser Medien sehr schwierig.



Um auf dem gleichen Medium, wie die CD bzw. DVD, eine höhere Speicherkapazität möglich zu machen, wurde die **Struktur auf der Blu-ray Disc verkleinert**. Dazu wird ein **Laser im blauvioletten Bereich** verwendet und die Wellenlänge auf **405 nm** abgesenkt. Blaues Licht lässt sich feiner fokussieren und ermöglicht so eine **höhere Datendichte auf der Scheibe**. Dadurch muss aber auch der **Abstand zwischen Scheibe und Schreib-/Leseoptik verringert** werden.

Formate

Format	Speicherkapazität
BD-ROM SL	25 GByte
BD-ROM DL	50 GByte
BD-R (Recordable)	25 GByte
BD-RE SL (Rewriteable)	25 GByte
BD-RE DL (Rewriteable)	50 GByte
BDXL	100 und 128 GByte (40 MByte/s)

4.3.4.2.4) UHD-BD - Ultra-HD Blu-ray Disc

Die Ultra-HD Blu-ray Disc, kurz UHD-BD, ist ein optisches Speichermedium, dass auf der BDXL-Disc der Blu-ray Disc Association basiert. Allerdings sind UHD-BD und BDXL nicht vollständig identisch, weshalb nicht alle BDXL-Brenner UHD-BDs abspielen können. Wie gewohnt hat die Scheibe einen **Durchmesser von 12 cm**. Eine UHD-BD hat **zwei Schichten auf die zusammen 66 GByte** passen. Auf einer zweiten Variante mit **drei Schichten finden rund 100 GByte** Platz.

Die Ultra-HD Blu-ray dient **zum Speichern von 4K-Filmen und Abspielen auf TV-Geräten mit Ultra-HD-Auflösung (3.840 × 2.160 Pixel)**. Die entsprechenden UHD-BD-Hüllen sind schwarz und mit einem silbernen Logo versehen.

4.3.4.3) Elektronischer Datenspeicher (Halbleiterspeicher)

Halbleiterspeicher dient sowohl der **zeitlich begrenzten** als auch der **unbegrenzten Aufbewahrung von Daten, Zuständen und Programmen** in **Form von digitalen Signalen**.

Elektronische Datenspeicher fassen **Halbleiterbauelemente, vorwiegend Transistoren und Kondensatoren, zu integrierten Schaltkreisen zusammen**, um Daten und Informationen zu speichern. Dazu werden **spezielle nicht-leitende Materialien**, wie **zum Beispiel Silizium, gezielt verunreinigt**, um sie **unter bestimmten Bedingungen in einen leitenden oder nicht-leitenden Zustand zu versetzen**. Die Zustände leitend und nicht-leitend können dabei die beiden binären Zuständen „0“ und „1“ abbilden.

4.3.4.3.1 ROM - Read Only Memory

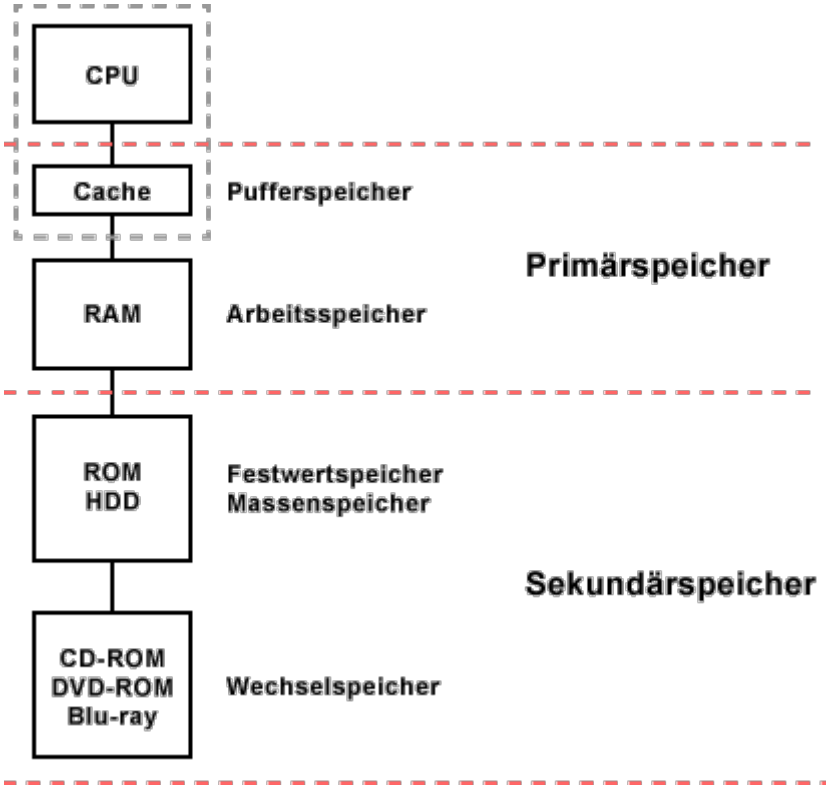
Das ROM ist ein **digitaler Festwertspeicher bzw. permanenter Halbleiterspeicher**, in dem **Daten dauerhaft und unveränderlich gespeichert werden**. Der **Speicherinhalt kann also NICHT neu programmiert oder beschrieben werden**. Typischerweise enthalten permanente Halbleiterspeicher **Betriebssysteme, Anwendungsprogramme und Firmware** an denen **während des Betriebs keine Änderungen** vorgenommen werden müssen. In der Regel werden ROM- und PROM-Bausteine nicht fest die Platinen gelötet, sondern werden in Sockel gesteckt und sind somit austauschbar.
Die Herstellung von ROM-Bausteinen war schon immer **relativ teuer**. Wenn überhaupt **lohnt sich das nur bei Massenprodukten**.

Heute dienen **Flash-Speicher als ROM- und PROM-Ersatz**. Vor allem deshalb, weil sich der **Speicherinhalt von Flash-Memory jederzeit**, auch während des Betriebs, **überschreiben** lässt. Speicher in der Form „Read Only Memory (ROM)“ sollte man nur noch als eine Form eines Datenspeichers sehen, der in der Praxis so nicht mehr vorkommt. Höchstens nur noch dort, wo man keine Änderung des Speicherinhalts wünscht. In der Praxis ist es jedoch wichtig, dass ein Halbleiterspeicher jederzeit veränderbar ist.

4.3.4.3.2) RAM (Random Access Memory)

RAM bezeichnet einen **Speichertyp dessen Speicherzellen über ihre Speicheradressen direkt angesprochen** werden können. In diesem Zusammenhang wird auch von **„wahlfrei“** gesprochen, was sich auf **„random“** bezieht und **nichts mit „Zufall“ oder „zufälligem Zugriff“ zu tun** hat. In diesem Zusammenhang spricht man von **„Speicher mit wahlfreiem Zugriff“** oder „Direktzugriffsspeicher“. Auf **andere Speicherarten (zum Beispiel Flash) kann man nur blockweise zugreifen**. **RAM erlaubt den Zugriff auf jede einzelne Speicherzelle**. Bei ROM (Read-Only-Memory, Nur-Lese-Speicher) funktioniert das genauso. Bei RAM funktioniert es lesend, wie auch schreibend. **Doch wird die Stromversorgung abgeschaltet gehen die Daten im RAM verloren**.

RAM wird in **Computer- und Mikrocontroller-Systemen als Arbeitsspeicher** eingesetzt. Weil in der Regel nur RAM als Arbeitsspeicher verwendet wird, wird **RAM gerne als Abkürzung für Arbeitsspeicher** verwendet. In diesem **Arbeitsspeicher werden Programme und Daten von externen Speicherträgern und Festplatten geladen**. Zur schnellen Verarbeitung kann der Prozessor darauf zugreifen, verarbeiten und danach wieder in den Arbeitsspeicher schreiben.



Prinzipiell **unterscheidet** man zwischen **statischem RAM und dynamischem RAM**. SRAM und

DRAM sind flüchtige Halbleiterspeicher. Sie verlieren nach dem Ausschalten ihren Speicherinhalt.

- **SRAM - Static Random Access Memory** (Statisches RAM)
- **DRAM - Dynamic Random Access Memory** (Dynamisches RAM)

SRAM - Static Random Access Memory

SRAM ist ein statischer Halbleiterspeicher, was bedeutet, dass der **Speicherinhalt mittels Flip-Flops gespeichert wird und so nach dem Abruf des Speicherinhaltes erhalten bleibt**. Dadurch ist der **Stromverbrauch sehr hoch**, was aber zu einem **schnellen Arbeiten innerhalb des Speichers** führt. Aufgrund seines **hohen Preises** und des **großen Stromverbrauchs** wird SRAM **nur als Cache- oder Pufferspeicher** mit geringen Kapazitäten verwendet.

- Speicherung erfolgt in Flip-Flops
- sehr schnell
- kein Refresh nötig
- hoher Stromverbrauch
- Einsatz als L1-, L2- und L3-Cache

DRAM - Dynamic Random Access Memory

DRAM ist der **einfachste und billigste Speicher**. Im Computer-Bereich ist **DRAM als Arbeitsspeicher bzw. Hauptspeicher der bevorzugte Halbleiterspeicher**, den es in verschiedensten Varianten und Weiterentwicklungen gibt. Heute ist der SDRAM der am weitesten verbreitete Halbleiterspeicher in der Computertechnik. Im Gegensatz zum SRAM muss der **Speicherinhalt beim DRAM zyklisch aufgefrischt werden (Refresh)**. Dies ist normalerweise in Abständen von **einigen zig Millisekunden** erforderlich. Das Auffrischen des Speichers erfolgt zeilenweise.

- Kondensator als Speicherelement
- Speicherhaltung durch Refresh der Speicherzellen
- langsam
- geringer Stromverbrauch
- Einsatz als Arbeitsspeicher oder Hauptspeicher

Eine DRAM-Speicherzelle besteht aus einem Transistor und einem Kondensator (1T1C), der das eigentliche Speicherelement ist. In einer DRAM-Speicherzelle wird ein Bit durch die Ladung des Kondensators gespeichert. Die Messung der Spannung am Speicherkondensator (Lesen) und dessen anschließende Aufladung (Schreiben) benötigen eine gewisse Zeit.

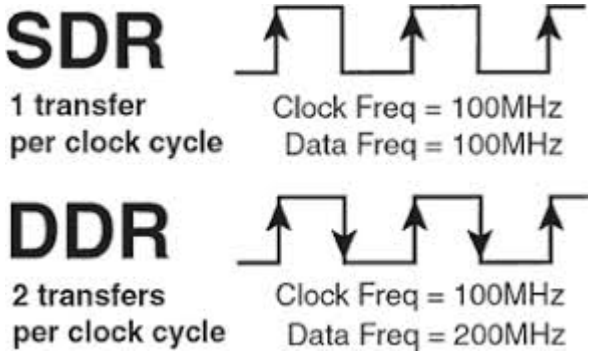
SDRAM - Synchronous DRAM

SDRAM ist der am **häufigsten verwendete Arbeitsspeicher** bzw. Hauptspeicher **in Computersysteme**n. SDRAM hat die Eigenschaft, dass er seine **Schreib- und Lesezugriffe am Systemtakt orientiert**. Das bedeutet, er arbeitet **synchron mit dem Speicherbus**. Woraus die Bezeichnung „Synchronous DRAM“ abgeleitet wird. Im Gegensatz dazu arbeitet normales DRAM asynchron. Die synchrone Arbeitsweise vereinfacht und beschleunigt die Ansteuerung des Speichers. SDRAM kann programmiert und so die Art des Zugriffs gesteuert werden. Auf diese Weise lässt sich

SDRAM an jede beliebige Anwendung anpassen.

DDR-SDRAM - Double Data Rate SDRAM (DDR1 / DDR2 / DDR3 / DDR4)

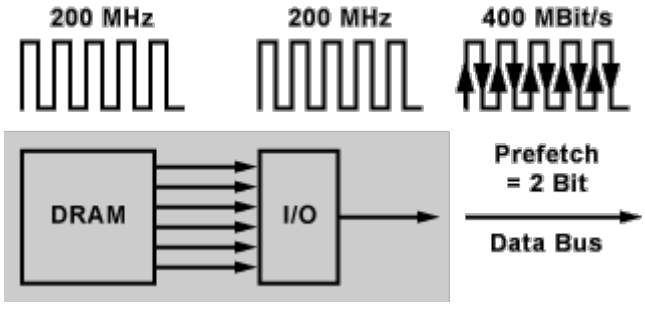
DDR-SDRAM und die Varianten DDR2, DDR3 und DDR4 sind gängige Halbleiterspeicher für Computersysteme. **DDR-SDRAM wird als Arbeitsspeicher bzw. Hauptspeicher** verwendet. Dieser Halbleiterspeicher wird nicht nur in Computern, sondern auch in Kraftfahrzeugen, Netzwerken, Kommunikationstechnik, medizinischen Apparaten und in der Unterhaltungselektronik eingesetzt.



DDR-SDRAM entspricht dem normalen SDRAM, jedoch mit einer kleinen Modifikation: Bei der Übertragung der Daten wird **nicht nur die ansteigende Flanke, sondern auch die abfallende Flanke des Taktsignals zur Datenübertragung genutzt**. In der Praxis entspricht das einer **Taktverdopplung**. Rein rechnerisch entsteht so eine **Verdopplung der Übertragungsrate**.

Double Data Rate bedeutet, dass **pro Übertragungszyklus (eine Taktrate) zweimal bzw. die doppelte Menge an Daten übertragen** wird. Damit die beteiligten Komponenten des Speicherbusses synchron arbeiten orientieren sie sich am Speichertakt und nutzen durch die DDR-Technik sowohl die steigende als auch die fallende Taktflanke. Normalerweise würde man nur die steigende Taktflanke nehmen. Ein Speicherbus, der mit 100 MHz arbeitet würde bei DDR rein rechnerisch mit 200 MHz arbeiten. Soviel zur grundsätzlichen Funktionsweise von Double Data Rate (DDR).

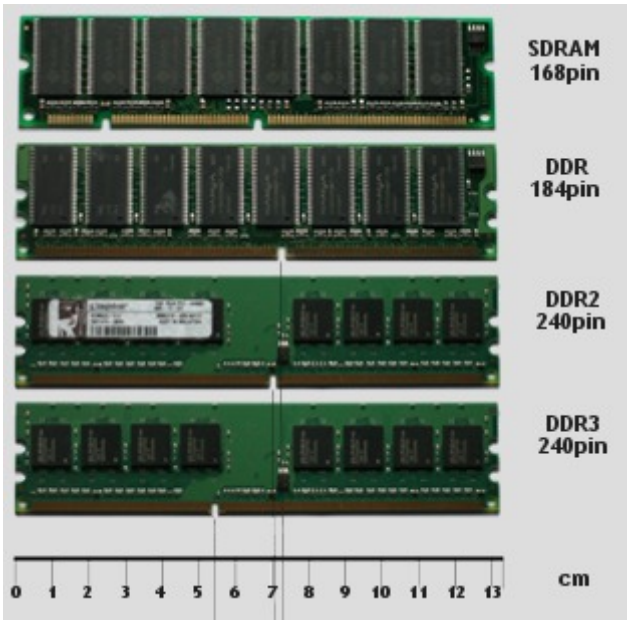
Eine weitere Erhöhung der Übertragungsrate wäre dann nur noch durch die Erhöhung der Taktrate möglich. Doch Vorsicht, man **unterscheidet zwischen dem internen Speicher-Takt in den Speicherchips und dem externen Speicherbus-Takt**. Das bedeutet, wenn der Speicherbus-Takt mit 100 MHz und durch DDR rechnerisch mit 200 MHz arbeitet, bedeutet das nur, dass sich die DDR-Technik als Taktverdopplung von 100 auf 200 MHz auswirkt. Denn intern arbeitet der Speicher nur mit 100 MHz. Double Data Rate bezieht sich hier nur auf den Speicherbus, nicht auf die Speicherchips.

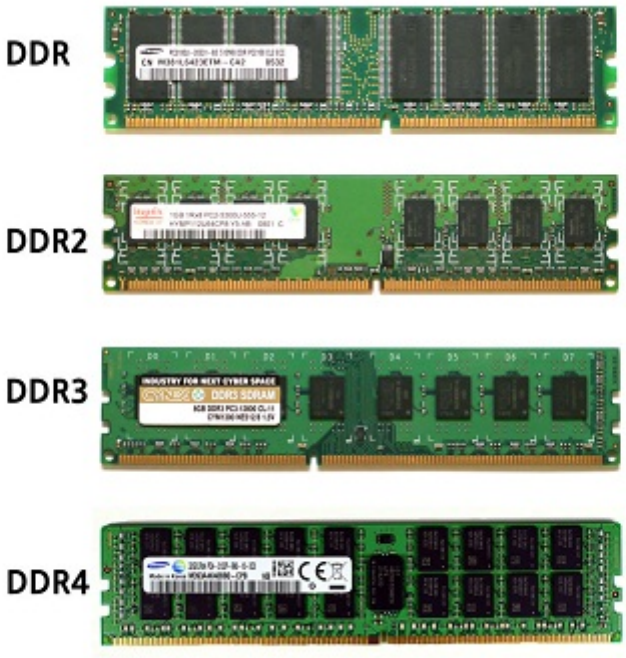


Das bedeutet, die Verdopplung der Übertragungsrate auf dem Speicherbus ist nur wenig sinnvoll,

wenn der eigentliche Speicher die Daten in dieser Geschwindigkeit nicht liefern kann. Die Frage ist, wie schafft man es, dass die Verdopplung der externen Datentransferrate auch wirklich genutzt werden kann. Das erreicht man nur durch **Prefetching** innerhalb des Speichers. Dazu werden **einfach zwei (DDR) oder mehr (DDR2, DDR3) Datenbits auf einmal ausgelesen**. Die Zugriffsbeschleunigung durch **Prefetching funktioniert** aber **nur dann**, wenn der **Speicher-Controller hintereinander liegende Adressbereiche aus der gleichen Speicherfeldzeile anfordert** oder wenn die **angeforderten Daten auf unterschiedlichen Speicherbänken** liegen.

DDR SDRAM standard	Release year	Bus clock (MHz)	Internal memory clock (MHz)	Prefetch (min burst)	Transfer rate (MT/s)	Voltage (V)	DIMM pins	SO-DIMM pins	MicroDIMM pins
DDR1	2000	100–200	100–200	2n	200–400	2.5/2.6	184	200	172
DDR2	2003	200–533.33	100–266.67	4n	400–1066.67	1.8	240	200	214
DDR3	2007	400–1066.67	100–266.67	8n	800–2133.33	1.5/1.35	240	204	214
DDR4	2014	800–1600	200-400	8n	1600–3200	1.2/1.05	288	256	—





4.3.4.3.3) Flash-Memory / Flash-Speicher

Flash-Speicher bzw. Flash-Memory **kombiniert die Vorteile von Halbleiterspeicher und Festplatten**. Wie jeder andere Halbleiterspeicher kommt Flash-Speicher **ohne bewegliche Teile** aus. Und die **Daten bleiben** wie bei einer Festplatte auch **nach dem Abschalten der Energieversorgung erhalten**.

Computer, deren Speicher rein auf Flash-Speicher basieren, sind der Traum eines jeden Software-Entwicklers und Anwenders. Der Computer müsste nie mehr minutenlang beim Starten booten, sondern wäre innerhalb weniger Sekunden sofort betriebsbereit. Genauso schnell wäre er auch ausgeschaltet.

Vorteile von Flash-Speicher

- Die gespeicherten Daten bleiben auch bei fehlender Versorgungsspannung erhalten. Auf eine Erhaltungsladung kann verzichtet werden. Somit ist auch der Energieverbrauch und die Wärmeentwicklung geringer.
- Wegen fehlender beweglicher Teile ist Flash geräuschlos, unempfindlich gegen Erschütterungen und magnetische Felder.
- Im Vergleich zu Festplatten haben Flash-Speicher eine sehr kurze Zugriffszeit. Lese- und Schreibgeschwindigkeit sind über den gesamten Speicherbereich weitestgehend konstant.
- Die erreichbare Speichergröße ist durch die einfache und platzsparende Anordnung der Speicherzellen nach oben offen.

Nachteile von Flash-Speicher

- begrenzte Schreib- bzw. Löschvorgänge
- begrenzte Speicherkapazität
- hoher Preis

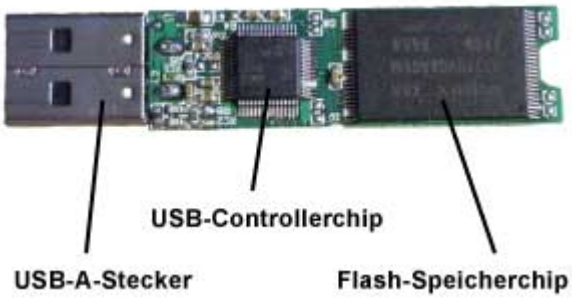
Anwendungen von Flash-Speicher

Flash-Speicherchips sind allgegenwärtig. Sie stecken in vielen Geräte des täglichen Gebrauchs.

- 4.3.4.3.3.1) USB-Stick
- 4.3.4.3.3.2) Speicherkarten
- 4.3.4.3.3.3) SSD - Solid State Drive
- 4.3.4.3.3.4) SSHD - Solid State Hybrid Drives
- Handy
- MP3-Player

4.3.4.3.3.1 USB-Stick

Ein USB-Speicherstick bzw. USB-Stick ist ein **Wechselspeicher**, der direkt an einen **USB-Port eines Computers steckbar** ist. Der Datenspeicher auf einem USB-Stick besteht aus Flash-Halbleiterspeicher.



USB-Sticks zeichnen sich durch eine **hohe Speicherkapazität und Zugriffsgeschwindigkeit** aus. Dadurch sind sie eine Alternative zu Disketten, wiederbeschreibbaren CDs oder DVDs. Weil die USB-Schnittstelle eine **Standard-Schnittstelle** ist, die jeder Computer hat, eignen sich **USB-Sticks als Wechselspeicher**. Der Zugriff ist so einfach, wie bei einer Festplatte. Der USB-Stick ersetzt damit auch die beliebten beschreibbaren CDs und DVDs, wenn es um den Transport und die Archivierung großer Dateimengen geht.

4.3.4.3.3.2) Speicherkarten

Speicherkarten sind **kleine Datenspeicher, die als Wechselspeicher in mobilen Geräten** eingesetzt werden. Speicherkarten verwenden Flash-Speicher als Speichertechnik. Die Flash-Speicherzellen behalten die Daten auch ohne Stromversorgung. Die **kleine und leichte Bauweise** erlaubt den Einsatz in kleinen mobilen Geräten. Auch der **Stromverbrauch hält sich in Grenzen**. Das ist **besonders bei Akku-betriebenen Geräten wichtig**. Zum Lesen und Beschreiben der Speicherkarten gibt es Lesegeräte in den unterschiedlichsten Ausführungen. Egal ob als Gehäuse-Einbaukit, als Karten-Adapter für USB oder fest eingebaut in Notebooks.

In der Regel werden Speicherkarten für **Digitalkameras und Handys** verwendet. Für Handys gibt es auch ganz kleine Speicherkarten, die nur so groß wie ein Fingernagel sind. Noch vor ein paar Jahren war die Auswahl an Speicherkarten sehr groß. Man musste zwischen CompactFlash, Memory Stick, Secure Digital, xD und MultiMediaCard wählen.

Übersicht

Bezeichnung	Abkürzung	Breite	Länge	Höhe	Kapazität	Bemerkung
CompactFlash I	CF	42,8 mm	34,4 mm	3,4 mm	bis 32 GByte	
CompactFlash II	CF	42,8 mm	34,4 mm	5 mm	bis 32 GByte	
Secure Digital High Capacity	SDHC	24 mm	32 mm	2,1 mm	bis 32 GByte	
SDXC	SDXC	24 mm	32 mm	2,1 mm	bis 2 TByte	
miniSD	miniSD	20 mm	21,5 mm	1,4 mm	bis 2 GByte	veraltet
miniSDHC	miniSDHC	20 mm	21,5 mm	1,4 mm	bis 8 GByte	veraltet
microSD	microSD	11 mm	15 mm	1 mm	bis 2 GByte	veraltet
microSDHC	microSDHC	11 mm	15 mm	1 mm	bis 16 GByte	
microSDXC	microSDXC	11 mm	15 mm	1 mm	bis 64 GByte	

4.3.4.3.3.3) SSD - Solid State Drive

Ein **Solid State Drive, kurz SSD, ist ein Massenspeicher**, vergleichbar mit einer Festplatte. Im **Gegensatz zur Festplatte hat eine SSD keine beweglichen Teile**. Bei der SSD ist das Speichermedium ein Flash-Speicher (z. B. NAND-Flash).
Flash-Speicher arbeiten vollkommen **geräuschlos**, hat **kurze Zugriffszeiten** und **schont den Akku, weil keine mechanischen Teile durch Motoren bewegt werden** müssen.
Die **SSD** wird **wie** eine **herkömmliche Festplatte angesprochen**. Um einen bestimmten Sektor zu lesen, muss eine Festplatte die Köpfe auf die richtige Spur bewegen. Diese Bewegung unterliegt einer gewissen mechanische Trägheit, die überwunden werden muss. Dann vergeht noch eine geringe Zeit, bis der gewünschte Sektor am Lesekopf vorbeidreht. Dieser ganze Zeitverlust fällt bei der SSD weg. SSDs zeichnen sich also durch eine **sehr hohe Lese- und Schreibgeschwindigkeit**, sowie einen **geringeren Energieverbrauch** aus.

Vorteile von SSD (Solid State Drive)

- hohe Transferraten
- kurze Zugriffszeiten, vor allem beim Lesen
- niedrige Leistungsaufnahme
- geräuschloser Betrieb

SSD-Schnittstellen

Wegen immer schnellerer Flash-Speicher und -Controller nimmt die **Geschwindigkeit von SSDs unaufhörlich** zu. Da SSDs als Festplatten-Ersatz dienen, ist die **SATA- bzw. SAS-Schnittstelle als Massenspeicher-Schnittstelle** hier maßgeblich im Einsatz. Im Vergleich zur Weiterentwicklung von SSDs bleibt die **Weiterentwicklung von SATA leider zurück**. Während es schon **SSDs** gibt, die **Daten mit 2 GByte/s** schaufeln können, hängt **SATA 6G bei 600 MByte/s** bzw. **SAS bei 1,2 GByte/s** fest. In der **Praxis können SATA-6G-Desktop-Festplatten** Daten linear mit **ca. 180 MByte/s lesen**. Sehr **schnelle Server-Festplatten erreichen ca. 250 MByte/s**. Mit der bisherigen Übertragungstechnik und den dazugehörigen Steckverbindern ist es leider nicht möglich, die Datenrate von SATA zu steigern.

SSD oder Festplatte?

Obwohl die SSD in den letzten Jahren sich immer mehr durchgesetzt hat, hat die klassische Festplatte immer noch ihre Daseinsberechtigung. Trotz der typischen Vorteile, wie hohe Geschwindigkeit bei

nicht-sequenziellen Zugriffen und niedrigen Energieverbrauch ist eine SSD nicht zwangsläufig der bessere Datenspeicher. Mit einem Blick auf **Speicherkapazität, Preis und Zuverlässigkeit spricht mehr für die herkömmliche Festplatte**.

Hohe Performance erreichen SSDs bisher **nur mit rechenstarken Controllern**, die ausgefeilte Wear-Leveling-Algorithmen beherrschen, viele Flash-Chips parallel anbinden und SDRAM-Cache nutzen. Eine **Zwischenlösung** zwischen Festplatten und SSDs sind **Hybrid-Festplatten (SSHD, Solid-State Hybrid Drives)**, die neben dem Plattenspeicher auch einen Chip-Speicher haben, der das Booten des Betriebssystems und den Start von Anwendungen beschleunigen kann.

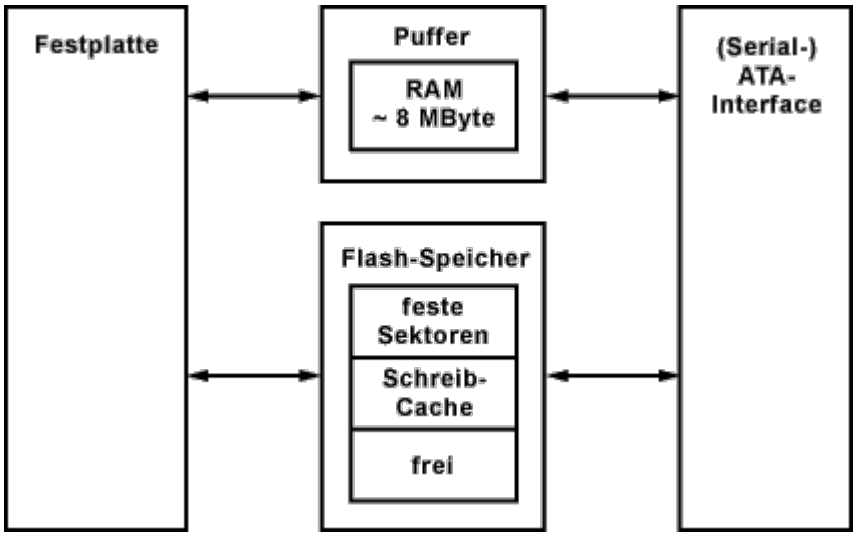
4.3.4.3.3.4) SSHD - Solid-State Hybrid Drives (Hybrid-Festplatten)

Solid-State Hybrid Drives, kurz SSHDs, sind **Hybrid-Festplatten**, die aus einer **herkömmlichen Festplatte mit einem zusätzlichen Flash-Speicher** bestehen. Der **Flash-Speicher dient als Datenpuffer für Schreib- und Lesezugriffe** und soll für **kürzere Zugriffszeiten** sorgen. Im Flash-Speicher speichert der Festplatten-Controller **häufig von den Magnetscheiben gelesene Daten** zusätzlich ab, so dass diese von dort mit SSD-Tempo gelesen werden können. Gleichzeitig werden Schreibvorgänge mit geringem Datenumfang in den Flash-Speicher und später am Stück auf die Magnetscheiben geschrieben. Das spart Strom, weil die Magnetscheiben in der Festplatte seltener anlaufen müssen.

Zusammenfassen kann man sagen, dass dieses Verfahren die **Leistungsaufnahme von Festplatten reduziert und die Lese- und Schreibgeschwindigkeit erhöht**. Hierbei gilt, **je größer der Flash-Speicher, desto besser funktioniert es**. Denkbar wäre, dass ein Computer vom Flash-Cache aus bootet oder aus dem Ruhezustand aufwacht. So könnte ein Computer in weniger als einer Sekunde betriebsbereit sein.

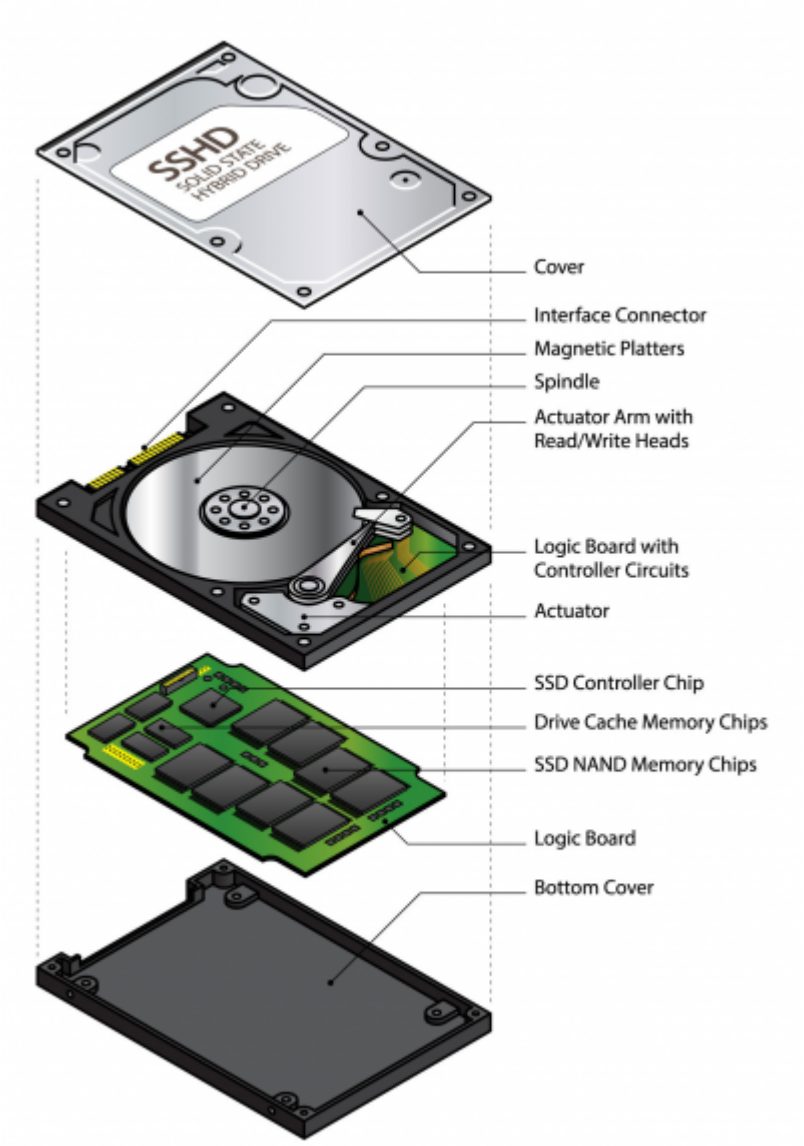
Beim Lesen und Schreiben arbeiten Hybrid-Festplatten rund **20% schneller**, wodurch die **Systemleistung aber nur um ca. 10%** steigt. Hinweis: Geschwindigkeitssteigerungen von unter 20% bemerkt der Anwender in der Regel kaum. **In der Regel lohnt sich der Einsatz von Hybrid-Festplatten nicht**.

Aufbau und Funktionsweise einer SSHD



Ein Teil des **Flash-Speichers** wird zum **Puffern von Schreibzugriffen** verwendet. Anstatt die Daten gleich auf die Hybrid-Festplatte zu schreiben, werden die **Daten im Flash-Speicher abgelegt**. Der

Speicher wird so lange beschrieben, **bis er voll ist**. In dieser Zeit bleibt das Laufwerk im Stromsparmodus. Erst dann, wenn der Flash-Speicher voll ist, wird das Laufwerk aufgeweckt und der **Inhalt des Flash-Speichers auf den Magnetspeicher** übertragen.
Auf diese Weise **beschleunigt der Flash-Speicher die Schreibzugriffe**. Vor allem die nichtlinearen Zugriffe, bei denen der Schreib-/Lese-Kopf der Festplatte sich mehrmals über die Platte bewegt. Das ist sehr zeitintensiv. Beim Schreiben auf den Flash-Speicher fallen die mechanischen Vorgänge erst mal weg. Auch beim Lesevorgang können Geschwindigkeitsvorteile entstehen.
Interessant sind **Hybrid-Festplatten vor allem in Notebooks**. Hier kommt es auf **geringe Leistungsaufnahme und schnelle Startzeiten** an. Es gibt ein spezielles ATA-Kommando, mit dem eine Hybrid-Festplatte angewiesen werden kann den Flash-Speicher zum Stromsparen zu verwenden. Dabei wird das Laufwerk möglichst oft schlafen gelegt und alle Schreibzugriffe erst mal im Flash-Speicher abgelegt.



4.3.5) Erweiterungskarten

Erweiterungskarten sind, ganz grob ausgedruckt, nichts weiter als Teile der Hardware, die benötigt wird, um den Computer ohne Austausch von ganzen Komponenten (z.B. Mainboard) um mehr Leistung oder Funktionen zu erweitern.

Gängige Erweiterungskarten sind:

4.3.5.1) Grafikkarte

Die Grafikkarte **berechnet visuelle Daten** und **gibt sie über eine Schnittstelle aus**, an der ein Bildschirm angeschlossen ist. Auf dem Bildschirm wird dann die **Benutzeroberfläche von Betriebssystem und Anwendungsprogrammen dargestellt**.

Grafikkarten bzw. deren Funktionen **bestimmen maßgeblich die Systemleistung und Eigenschaften** eines Computers. Wobei die Leistung der Grafikkarte in der **Hauptsache für Computerspiele** und **umfangreiche Grafikdarstellungen** relevant ist. In **komplexen Spielszenen berechnet die Grafikkarte detaillierte Objekte und die richtige Beleuchtung**. Während der **Hauptprozessor allgemeine Berechnungen** erledigt und zum Beispiel das **Verhalten von Spielfiguren** berechnet, übernimmt die **Grafikkarte die Berechnung** beispielsweise von **physikalischen Phänomenen, Explosionen und einstürzenden Gebäuden**. Wenn die Bildschirmdarstellung bei diesen Bewegungen oder Verschiebungen ruckelt, dann ist die Grafikkarte, insbesondere der Grafikprozessor (GPU), bei der aktuell gewählten Auflösung und Detailtiefe nicht leistungsfähig genug.

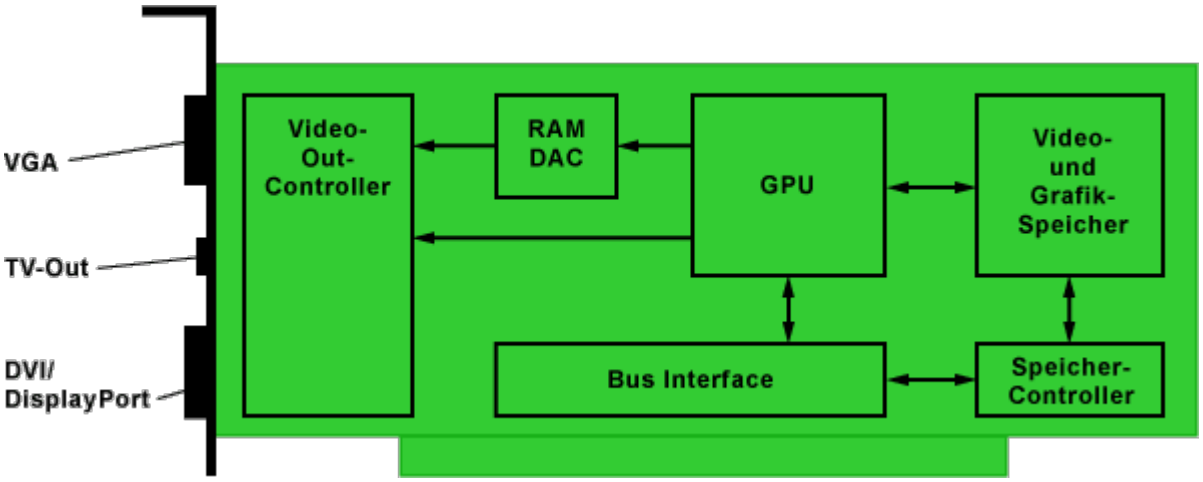
Zunehmend wird die **Grafikverarbeitung** von der Grafikkarte **in den Hauptprozessor** verlagert. Das bedeutet jedoch nicht, dass die Zeit der Grafikkarten vorbei ist. Es ist eher so, dass die Grafikfunktionen, die bereits „onboard“ im Chipsatz integriert sind, gleich in den Prozessor zu integrieren. Konsequenterweise muss die Grafikausgabe dann auch vom Prozessor aus erfolgen.

Anders als man denkt **profitieren Bildbearbeitungsprogramme** vor allem **von einer schnellen CPU** und **viel Arbeitsspeicher**. Die **Grafikkarte** bzw. der Grafikprozessor **verbessert** in der Hauptsache die **Geschwindigkeit der Bildschirmdarstellung**. Zum Beispiel beim **Verschieben von Fenstern und Bildelementen**, stufenloses **Zoomen und Drehen der Zeichenfläche**. Nur in bestimmten Fällen unterstützt die Grafikkarte das Computersystem direkt bei der Verarbeitung von Bild- und Videodaten. Zum Beispiel beim **Abspielen und Transcodieren von Videos**.

Aufbau einer Grafikkarte

Eine Grafikkarte besteht aus folgenden mechanischen und elektronischen Teilen:

- Abdeckung
- Lüfter
- Kühlkörper
- Grafikprozessor
- Speicher
- Platine
- Slotblech
- weitere Abdeckung



Aufgrund des **hohen Datenaufkommens zwischen Prozessor und Grafikkarte**, werden Grafikkarten mit einem **eigenen Prozessor** ausgestattet. Er soll den **Hauptprozessor** mit parallel laufenden Rechenoperationen **entlasten**. Das Herz der Grafikkarte ist der **Grafikprozessor (GPU)**, der die **wesentlichen Leistungsmerkmale einer Grafikkarte bestimmt**. Es gibt auch Grafikkarten, die zwei GPUs haben und die aufgrund der hohen Verlustleistung mit einem Kühlkörper oder sogar Lüfter gekühlt werden müssen.
Um die enormen Datenmengen verarbeiten zu können haben **Grafikkarten** einen **eigenen Arbeitsspeicher** und eine **spezielle Speicheranbindung mit hohen Taktraten**. Der **Grafikspeicher** kann durchaus die **Größe des normalen Arbeitsspeichers** haben.

Schnittstellen für Grafikkarten

Anfangs wurden Grafikkarten **über den internen Systembus** betrieben. Mit steigender Auflösung, Forderung nach Multimedia und 3D-Darstellung wurden die **Datenmengen immer größer**, die zur Grafikkarte transportiert werden mussten. **Um die steigenden Datenmengen zwischen Prozessor, Arbeitsspeicher und Grafikkarte bewältigen** zu können, hat Intel den **AGP-Steckplatz** für Grafikkarten eingeführt. Dieser Steckplatz war ausschließlich für Grafikkarten gedacht. Ein paar Jahre später wurde der **AGP** zusammen mit dem PCI **durch den PCI Express (PCIe) ersetzt**. Die heutigen Grafikkarten sind PEG-Grafikkarten (PCIe).

- PCI-X / PCI
- AGP - Accelerated Graphics Port
- PCI Express / PCIe
- PEG - PCI Express for Graphics

Anschlüsse für Bildschirm/Monitor

Ist die Darstellung schlecht, dann wird fast immer der Bildschirm dafür verantwortlich gemacht. Wird der Bildschirm analog, also über den VGA an die Grafikkarte angeschlossen, dann kann die **Signalqualität des VGA für das schlechte Bild verantwortlich** sein. Obwohl man annehmen könnte, dass das heute kein Problem mehr sein sollte, weisen manche Grafikkarten eine jämmerliche Signalqualität auf. Insbesondere die VGA-Anschlüsse haben diese Probleme. **Vorzugsweise sollte man den DVI- oder DisplayPort-Anschluss verwenden**.

VGA

DVI

DisplayPort

Obwohl mit **DVI** eine **digitale Schnittstelle für qualitativ hochwertige Bild- und Video-Signale** verfügbar ist macht die Einführung neuer Schnittstellen durchaus Sinn. Denn **DVI ist für hohe Auflösungen nicht geeignet**. Der **DisplayPort soll DVI** (Digital Visual Interface) im PC **und HDMI (High Definition Multimedia Interface)** in der Unterhaltungselektronik **ersetzen**.
Der **DisplayPort** ist elektrisch **kompatibel** zu den bestehenden Digitaleingängen **DVI und HDMI**. Er unterstützt auch den HDCP-Kopierschutzmechanismus und beherrscht zusätzlich ein eigenes Protokoll namens DPCP (Display Port Content Protection).

Auswahl einer Grafikkarte

Grafikkarten **unterscheidet** man im Hinblick auf ihre **3D- bzw. Spiele-Tauglichkeit, ihre Performance und Funktionsvielfalt**. Die Grafikkarte ist **nach den Anforderungen des Einsatzzwecks auszuwählen**. Man unterscheidet Grafikkarten nach ihrem Preis in **Einsteiger-, Mittelklasse-, Performance- und High-End-Modellen**. Je nach Preis unterscheiden sich Grafikkarten in Ihrer 3D-Leistung und Ausstattung.
Typische Leistungsmerkmale sind

- Taktfrequenz
- Shader-Anzahl
- Speichertakt
- Speicherbandbreite.

Bei einer Einsteiger-Grafikkarte ist die Spieletauglichkeit eingeschränkt, die Performance schwach und die Funktionsvielfalt begrenzt. Einfache Büro- und Internet-Anwendungen kommen mit einer günstigen Grafikkarte oder Onboard-Lösung aus. Bei einer High-End-Grafikkarte ist die Spieletauglichkeit unschlagbar, die Performance enorm und die Funktionsvielfalt lässt keine Wünsche offen. Insbesondere (Action-)Spiele erfordern zwangsläufig eine leistungsfähige Grafikkarte.

	Einsteiger	Mittelklasse	Performance	High-End
Preis	bis 75 EUR	bis 150 EUR	bis 300 EUR	bis 1000 EUR
Anzahl Shader	bis 160	bis 800	bis 2000	bis 4000
Speichergröße	512 MByte	1 GByte	2 GByte	4 GByte
Speichertyp	DDR2, DDR3, GDDR5	DDR3, GDDR5	GDDR5	GDDR5
typische 3D-Leistungsaufnahme	bis 50 Watt	bis 90 Watt	bis 170 Watt	bis 300 Watt oder mehr
Netzteil	min. 300 Watt	min. 350 Watt	min. 450 Watt	min. 500 Watt oder mehr

4.3.5.2) Soundkarte

Die Soundfunktion, in Form einer Soundkarte oder im Chipsatz integriert, ist ein fast unersetzlicher Teil eines Computers. Erst mit der Möglichkeit Sprache und Musik wiederzugeben und aufzunehmen, wird ein Computer zum Multimedia-Computer. In den **Anfangszeiten der Soundkarte waren Erweiterungskarten für den ISA-Bus** üblich. Diese Karten mussten per Jumper konfiguriert werden (Interrupt, DMA, Speicher) und gaben den Ton mit einer Auflösung von 8 Bit wieder. Das entsprach 256 möglichen Zuständen. Mehr als Piepsen war den Lautsprechern nicht zu entlocken. **Später** kamen **Soundkarten mit einer Auflösung von 16 Bit**. Daraus resultierten 65.536 Klangabstufungen. Das **entspricht Hifi-Qualität**. Da der ISA-Bus am Aussterben war, haben führende Soundkarten-Hersteller auf PCI-Karten (mit Plug & Play) umgestellt. Neben der automatischen Konfiguration durch das BIOS und Betriebssystem waren auch höhere Datenraten zwischen Soundkarte und Prozessor möglich. **Inzwischen ist es üblich, dass Soundfunktionen auf dem Motherboard integriert sind (onboard)**. Anfangs hat die Klangqualität sehr zu wünschen übrig gelassen. Heute ist die **Klangqualität vollkommen ausreichend und genügt auch anspruchsvollen Ohren**. Fast jedes Motherboard bietet heute einen integrierten Sound-Chip, der eine ausreichend gute Klangqualität liefert.

4.3.5.3) Netzwerkkarte (NIC - Network Interface Card)

Eine Netzwerkkarte ermöglicht es, auf ein Netzwerk zuzugreifen und arbeitet auf der Bitübertragungsschicht (Schicht 1) des OSI-Schichtenmodells. Jede **Netzwerkkarte hat eine Hardware-Adresse (Format: XX-XX-XX-XX-XX-XX), die es auf der Welt nur einmal gibt**. Anhand dieser Adresse lässt sich eine Station auf der Bitübertragungsschicht adressieren.

An einer Netzwerkkarte ist nicht nur die **RJ45-Buchse herausgeführt**, sondern meist **auch zwei LEDs**, die den **Status der Verbindung anzeigen**. Üblich sind RJ45-Buchsen, die zwei integrierte Status-LEDs in den **Farben Grün und Orange haben**.
Die **grüne LED zeigt an, dass eine hardwareseitige Verbindung besteht**. Dazu muss der Computer nicht eingeschaltet sein (bei Formfaktor ATX).
Die **orangene LED zeigt den Status der Übertragung** an. Wenn diese LED blinkt oder flackert, dann werden gerade Daten übertragen.
Bei manchen Netzwerkkarten sind diese Funktionen etwas anders. Wenn die grüne LED flackert, dann werden gerade Daten übertragen. Ansonsten ist sie ständig grün, wenn eine Verbindung besteht. Leuchtet die orangene LED (manchmal ist sie auch gelb), dann besteht eine 100-MBit-Verbindung.

4.3.6) Eingabegeräte

Als Eingabegeräte werden **alle Geräte** bezeichnet, **über die einem Computer Informationen zugeführt werden** können, sodass Interaktion mit den Computerprogrammen möglich ist.

Einige Eingabegeräte sind hier aufgelistet:

- Maus
- Tastatur
- Touchscreen
- Grafiktablett
- Mikro
- Webcam
- Scanner
- Joystic
- PC-Controller


Y
X
B
A


Why?

4.3.7) Ausgabegeräte

Als Ausgabegeräte werden in der Computertechnik **alle Geräte** bezeichnet, **die das Ergebnis einer Operation oder eines Programms der Außenwelt zugänglich machen**.

Dies sind insbesondere:

- Bildschirm/Beamer zur flüchtigen sichtbaren Ausgabe,
- Drucker/Plotter zur permanenten sichtbaren Ausgabe,
- Lautsprecher/Headset zur hörbaren Ausgabe,
- Braillezeile zur fühlbaren Ausgabe.

In Verbindung mit einem Eingabegerät ist Interaktion mit einem Computerprogramm möglich.

4.4) Schnittstellen (PC)

Eine Schnittstelle **verbindet Systeme, die unterschiedliche physikalische, elektrische und mechanische Eigenschaften** besitzen. Die **Definition oder Spezifikation einer Schnittstelle enthält gemeinsame Eigenschaften**. Dazu gehört auch ein **Protokoll** für die **Kommunikation** und den **Datenaustausch**. Schnittstellen befinden sich überall dort, wo unterschiedliche Systeme miteinander verbunden werden müssen. Die Schnittstellen bilden den **Übergang von einem System in ein anderes System**. Dieser Übergang kann zur Kommunikation oder dem Datenaustausch verwendet werden.

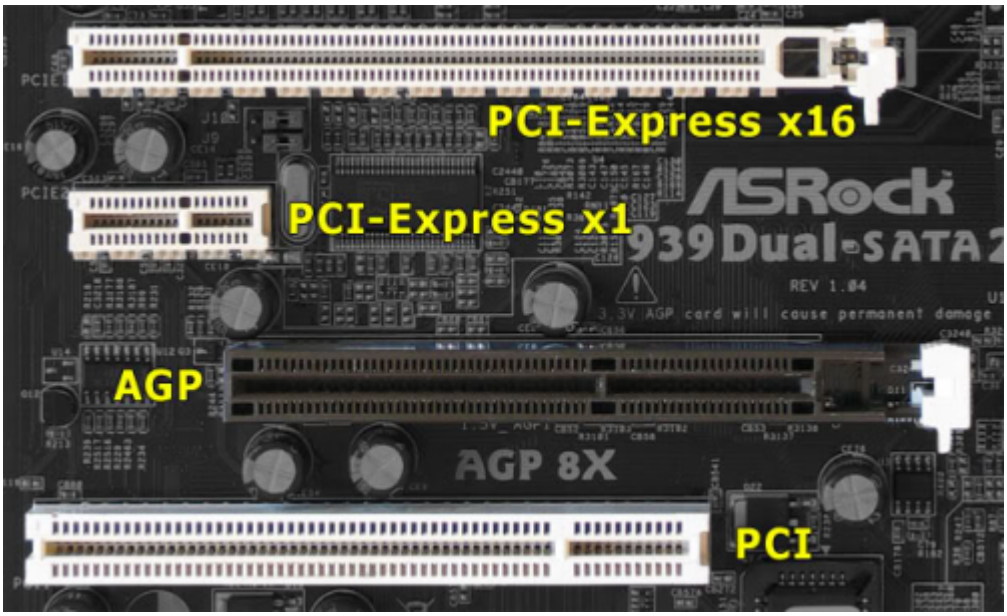
Ein Computer hat **interne** Schnittstellen, die sich **im Computer-Gehäuse** befinden und **externe** Schnittstellen, die **aus dem Computer-Gehäuse herausgeführt** sind.

4.4.1) Interne Schnittstellen

Interne Schnittstellen verbinden Systeme innerhalb eines Computers. Diese Schnittstellen werden meist auf dem **Motherboard als Sockel oder Slot** herausgeführt. Dort werden dann Erweiterungskarten direkt oder interne Laufwerke über Kabel angeschlossen. Einige andere Schnittstellen werden nicht herausgeführt, sondern befinden sich auf der Hauptplatine zwischen den einzelnen Controllern.

4.4.1.1) Interne Schnittstellen für Erweiterungskarten

AGP (Accelerated Graphics Port)



Der braun gefärbte AGP- Steckplatz wurde als **Schnittstelle für Grafikkarten** entwickelt und ist **doppelt so schnell, wie der PCI-Bus**. Mit der letzten Spezifikation „AGB 3.0“ kann eine **maximale Taktrate von 2133 MHz** erreicht werden. Aufgrund der **Nachfrage nach höher getakteten Grafikkarten** - vor allem für die Spieleindustrie - wurde dieser Slot **von der PCI Express Schnittstelle abgelöst.**

PCI - Peripheral Component Interconnect

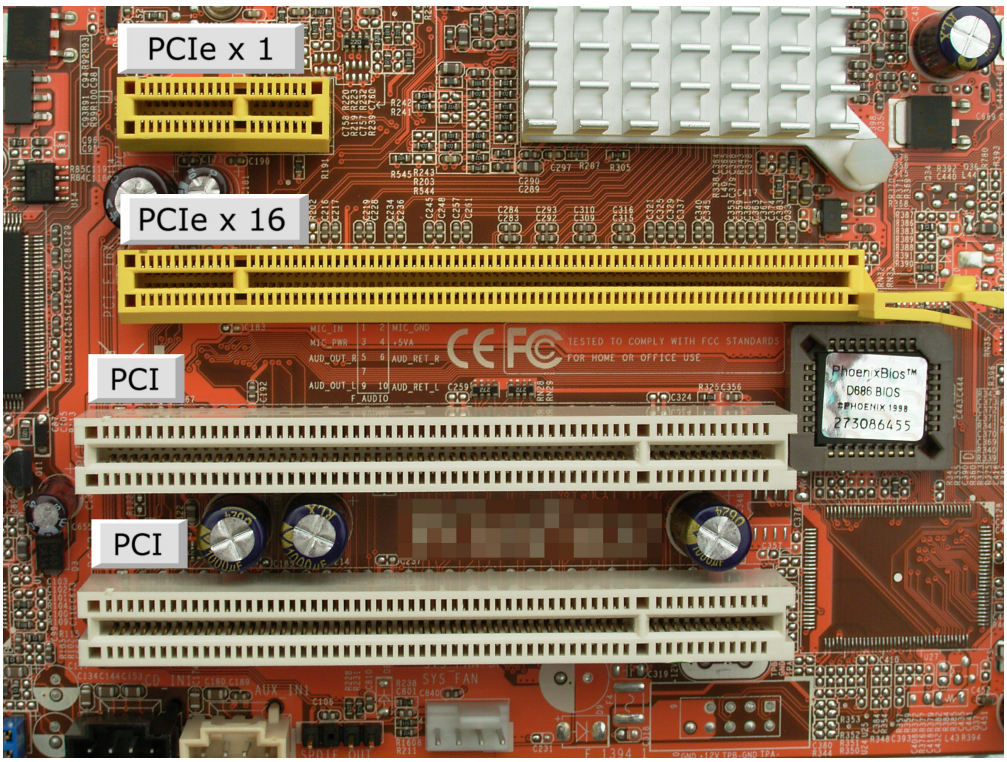
Der **PCI-Bus ist Industriestandard** und war viele Jahre fester Bestandteil von IBM-kompatiblen PCs, Macs von Apple und Alpha-Workstations von Digital. Das von Intel im Jahr 1993 entwickelte Bus-System ist bis ins Detail normiert, so dass andere Computerhersteller den PCI-Bus nachbauen können.

Der PCI-Bus kann **32 oder 64 Signalleitungen** haben. Der PCI-Bus mit 32 Bit teilen sich Adress- und Datenbus die Signalleitungen. Die Signalleitungen werden im Multiplex-Betrieb genutzt. Das bedeutet, mit einem Takt wird zuerst die Adresse und in einem zweiten Takt das Datenwort gesendet. So werden 32 Signalleitungen eingespart. In Servern kommt der PCI-Bus mit 64 Bit zum Einsatz. dort stehen jeweils 32 Adress- und Datenleitungen zur Verfügung.

Sie sind an ihrer weißen Färbung zu erkennen. Eine Erweiterung des Computers mittels Soundkarten, Netzwerkkarten, TV-Karten o.ä. ist über diese Slots möglich.

Spezifikation	Bus-Breite	Taktfrequenz	Datentransferrate	Signalspannung	Geräte pro Bus	Einführung
PCI 2.0	32 Bit	8 bis 33 MHz	0,12 GByte/s	5 V	6	1993
PCI 2.3	32 Bit	33 MHz	0,133 GByte/s	5V	6	2002
PCI 2.3	64 Bit	33 MHz	0,266 GByte/s	5V	6	2002
PCI 2.3	32 Bit	66 MHz	0,266 GByte/s	3,3V	3	2002
PCI 2.3	64 Bit	66 MHz	0,533 GByte/s	3,3V	3	2002
PCI-X 1.0	64 Bit	66 MHz	0,533 GByte/s	3,3V	4	1999
PCI-X 1.0	64 Bit	100 MHz	0,800 GByte/s	3,3V	2	1999
PCI-X 1.0	64 Bit	133 MHz	1,066 GByte/s	3,3V	1	1999
PCI-X 266 (2.0)	64 Bit	133 DDR	2,133 GByte/s	1,5V	1	2003
PCI-X 533 (2.0)	64 Bit	133 QDR	4,266 GByte/s	1,5V	1	2003

PCI-E(xpress)



PCI Express (PCIe) ist eine schnelle interne Schnittstelle für Erweiterungskarten in Computer-Systemen. Mit der **Einführung von PCIe im Jahr 2004 wurde dem AGP als Grafikkarten-Schnittstelle ein Ende gesetzt** und auch der **PCI als internes Computer-Bussystem abgelöst**.

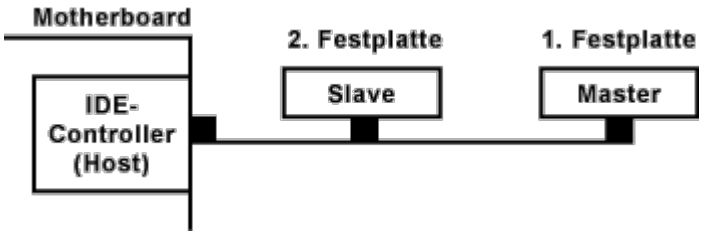
Die **Übertragungsgeschwindigkeit** bei PCIe **orientiert sich an der Version und der Anzahl der Links bzw. Lanes**. Je höher die Version und je mehr Links, desto höher die Bandbreite und desto höher ist die Übertragungsgeschwindigkeit. Die Bandbreite gibt dabei an, wie viel Kapazität für die Datenübertragung theoretisch bzw. maximal zur Verfügung steht. Die tatsächliche Datenrate liegt jedoch darunter.

PCIe	Bandbreite pro Link		PCIe x1	PCIe x4	PCIe x8	PCIe x16	Kodierung/Balast	Verfügbar seit
1.0	2,5 GT/s	2,5 GBit/s	250 MByte/s	1 GByte/s	2 GByte/s	4 GByte/s	8b10b / 20%	2004
2.0	5 GT/s	5 GBit/s	500 MByte/s	2 GByte/s	4 GByte/s	8 GByte/s	8b10b / 20%	2008
3.0	8 GT/s	10 GBit/s	0,9846 GByte/s	3,938 GByte/s	7,877 GByte/s	15,754 GByte/s	128b/130b / <2%	2011
4.0	16 GT/s	20 GBit/s	1,969 GByte/s	7,877 GByte/s	15,754 GByte/s	31,508 GByte/s	128b/130b / <2%	2017
5.0	32 GT/s		3,9 GByte/s	15,8 GByte/s	31,5 GByte/s	63 GByte/s	128b/130b / <2%	?

4.4.1.2) Interne Schnittstellen für Massenspeicher

IDE - Integrated Drive Electronics

IDE bedeutet offiziell **Integrated Device Elektronics**. Bei IDE handelt sich um eine **alte Festplatten-Schnittstelle**, bei der die Steuerungselektronik bzw. der Controller in das Festplattengehäuse integriert ist. Nachfolger von IDE ist Enhanced IDE (EIDE), was auch als ATA (PATA) bezeichnet wird, eine deutlich höhere Übertragungsgeschwindigkeit hat und den Anschluss von CD-ROM- und DVD-Laufwerken ermöglicht. IDE-Controller war Anfangs eine ISA-Steckkarte für den AT-Bus. Er war jedoch eher ein sogenannter Host-Adapter, der nur die notwendigen Systembussignale mit Pufferung zur Festplattenelektronik weiterleitete. Das **40-adrige IDE-Flachbandkabel** stellte praktisch die **Verlängerung des Systembusses** dar. Später wurde der IDE-Controller fest auf dem Motherboard integriert.

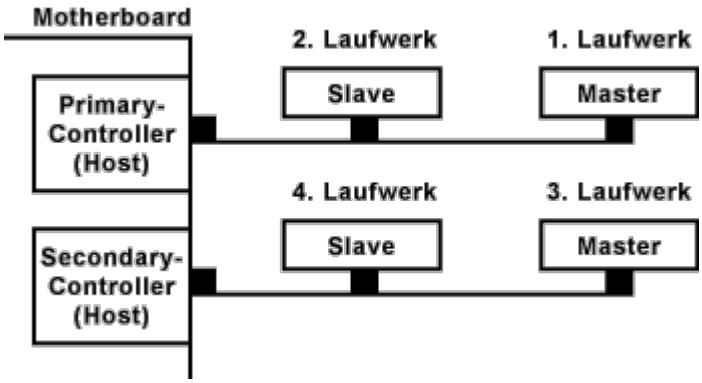


Das **IDE-Flachbandkabel hat drei Steckerleisten**. Die eine ist für den **Hostanschluss** auf dem IDE-Controller. Die anderen beiden Steckerleisten sind für das **Master- und Slave-Endgerät**. **Pro IDE-Controller** lassen sich **zwei Festplatten** betreiben. Weil die eigentliche Steuerung auf den Festplatten sitzt, muss die eine Festplatte, am besten die schnellste, als Master und die andere als Slave konfiguriert werden. Dazu müssen Jumper oder Dip-Schalter gesetzt werden. Die Master-Slave-Konfiguration sorgt dafür, dass beim Systemstart der Master die höhere Priorität hat. Bevor er Funktionsbereitschaft an das Bios meldet, wartet er auf die Bestätigung der Slave-Festplatte. Beide **Festplatten (Laufwerke) arbeiten unabhängig voneinander**. Sie belegen aber die gleichen Adressen im Computersystem.

P-ATA / Ultra-ATA / EIDE

EIDE bzw. ATA sind **alte Schnittstellen für den Anschluss von Festplatten und Wechselspeicher-Laufwerken**, wie CD-ROM, DVD oder Streamer in einem Computer. **EIDE wurde von SATA abgelöst**.

Die **EIDE-Schnittstelle (Enhandced Integrated Drive Electronics)** ist eine **Weiterentwicklung des IDE-Standards**. Die EIDE-Schnittstelle bezeichnet man auch als **ATA-Schnittstelle**. ATA steht für **Advanced Technology (AT) Attachment**. Die Bezeichnung EIDE wird nur noch selten verwendet. Mit dem Aufkommen von **Serial ATA (S-ATA)** wurde die Bezeichnung **P-ATA** immer gebräuchlicher. Wobei das **P für parallel** steht.



S-ATA / Serial-ATA

Serial-ATA, kurz SATA oder S-ATA, ist eine **Schnittstelle zum Anschluss von Massenspeichern, wie Festplatten und Wechselspeicher-Laufwerken**. Schnittstellen für Massenspeicher waren **ursprünglich immer Bussysteme mit parallel geführten Signalleitungen** in Leiterbahnen und Anschlusskabeln. Mit **zunehmender Übertragungsgeschwindigkeit** ergaben sich **technische Schwierigkeiten**, die für die Übertragungsrate eine obere Grenze setzten. So blieb auch die ATA (EIDE)-Schnittstelle nicht davon verschont, dass sie auf eine **seriellen Betriebsart umgestellt** wurde.

Im **Jahr 2000** setzten sich mehrere Firmen aus dem IT-Sektor zusammen, um eine **Spezifikation über Serial-ATA (Serielles ATA) zu erstellen**. Im Jahr 2001 wurde die erste Version von Serial-ATA

vorgestellt. Anfang 2003 waren bereits die ersten Controller und Festplatten erhältlich. Bis zur vollständigen Marktdurchdringung hat es noch bis zum Jahr 2004 gedauert. Mit **150 MByte/s hat SATA** direkt an die parallele EIDE-Schnittstelle (P-ATA) mit 133 MByte/s angeknüpft. Die **SATA-Schnittstelle unterstützt 1,5 GBit/s** bei einer **Nettodatenrate von ca. 150 MByte/s**. **Festplatten mit 10.000 Umdrehungen** in der Minute (U/m) liefern rund **75 MByte/s** an Daten. Mit **SATA 6G** erreichen herkömmliche Festplatten **fast 500 MByte/s (Schreibgeschwindigkeit)**.

Zwar wurde SATA mit SATA-II und SATA 6G noch zwei mal auf maximal 600 MByte/s beschleunigt. Für **Datenspeicher mit Flash-Memory (SSD, Solid State Drive)** ist das aber **nicht schnell genug**. Allerdings gibt es SSDs, die Daten mit weit über 1.000 MByte/s lesen und schreiben können. Dafür bedarf es auch einer Schnittstelle, die diese Datenmenge bewältigen kann. SATA kann das nicht leisten. Deshalb wird **SATA durch SATA Express (SATAe) oder PCI Express (PCIe) als Massenspeicher-Schnittstelle abgelöst**.

Schnittstelle	Bezeichnung	Transferrate		Reichweite	Geräteanzahl	Einführung
Serial-ATA	SATA	1,5 GBit/s	150 MByte/s	1 m	4	2003
Serial-ATA-2	SATA-II	3 GBit/s	300 MByte/s	1 m	16	2005
Serial-ATA-3	SATA 6G / SATA-600	6 GBit/s	600 MByte/s	1 m	16	2007
Serial Attached SCSI (SAS)	SAS	3 GBit/s	300 MByte/s	1 m	16.384	2004
Serial Attached SCSI 2 (SAS 2)	SAS 6G	6 GBit/s	600 MByte/s	1 m	16.384	2007
Serial Attached SCSI 3 (SAS 3)	SAS 12G	12 GBit/s	1.200 MByte/s	1 m	16.384	2010

SATAe / Serial-ATA Express

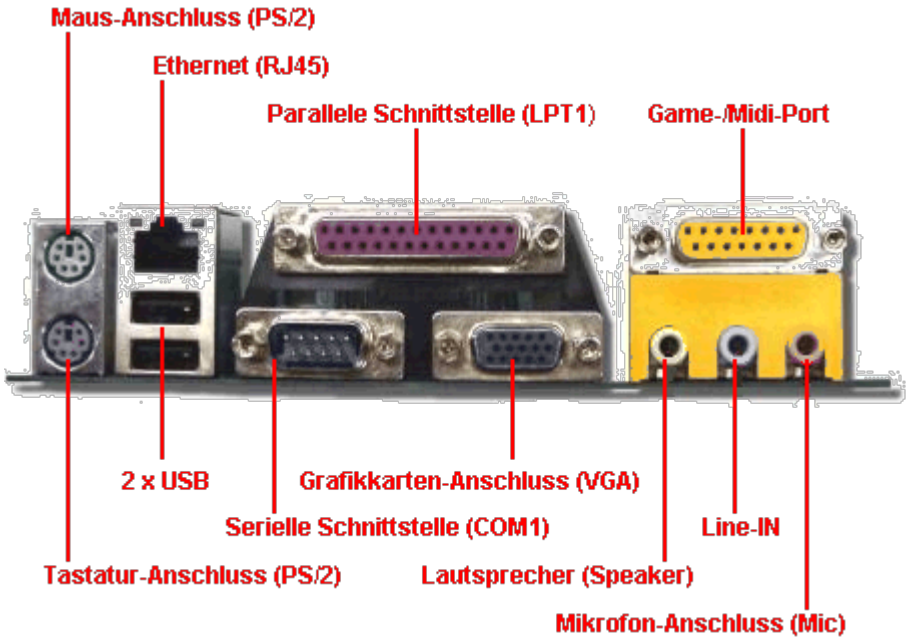
SATA Express ist der **offizielle Nachfolger von SATA 6G** und nutzt einen zu SATA 6G abwärtskompatiblen Steckverbinder. Dieser Steckverbinder bündelt **zwei SATA-Ports** die **wahlweise für die SATA-6G- oder PCIe-Übertragung** genutzt werden können. Denn **SATA Express beherrscht sowohl SATA-6G, als auch PCIe**. Die SATAe-SSD hat dabei die Form einer herkömmlichen Festplatte. Mit PCIe 2.0 bringt SATA Express aber nur 1 GByte/s und mit PCIe 3.0 wären es auch „nur“ 2 GByte/s. Das ist für viele Anwendungen viel zu wenig. **SATA Express** scheint

schon vor seiner breiten Nutzung veraltet.

Es ist davon auszugehen, dass es sich bei SATAe nur um Übergangslösungen handelt. Mittelfristig wird sich für **SSDs der PCI Express als Standardschnittstelle** durchsetzen.

4.4.2) Externe Schnittstellen

Externe Schnittstellen werden aus dem Computer-Gehäuse herausgeführt. Sie **verbinden Systeme oder Peripherie-Geräte** mit dem Computer. Die Verbindung wird mit einer **Kombination aus Stecker und Buchse** realisiert.



Serielle Schnittstelle

An jeder seriellen Schnittstelle kann nur ein weiteres Gerät angeschlossen werden. Die Bits werden nacheinander (seriell) über eine einzige Leitung übertragen, deshalb entsteht hier nur ein sehr geringer Kostenaufwand, aber auch eine niedrige Übertragungsrate. Klassische Endgeräte einer seriellen Schnittstelle sind Maus und Modem aber auch zahlreiche technische Einrichtungen, wie zB. Messgeräte, haben diese Schnittstelle, um mit Computern verbunden werden zu können.

Parallele Schnittstelle

Diese Schnittstelle wurde ursprünglich vom Drucker-Hersteller Centronics (daher oft Centronics-Schnittstelle genannt) entwickelt und war eine der ersten Schnittstellen für Drucker. Hier werden bereits 8 Bit gleichzeitig über jeweils eine eigene Leitung übertragen was im Vergleich zur seriellen Schnittstelle zu einer höheren Übertragungsrate führt. Je nach der Qualität des Kabels kann problemlos eine Leitungslänge von 5m für eine fehlerlose Datenübertragung genutzt werden.

USB (Universal Serial Bus)

Der USB ist eine universelle, externe Schnittstelle für alle Peripheriegeräte, die an einem Computer angeschlossen werden. Egal ob Tastatur, Maus, Modem, Drucker, Mikrofon, Lautsprecher, Kamera oder Scanner. Mit dem USB sind die Anwender unabhängig von der Anzahl der verfügbaren Schnittstellen und Steckplätze für Erweiterungskarten. Der USB lässt sich durch Steckkarten oder USB-Hubs fast beliebig erweitern.
Die Identifikation der Geräte wird vom USB-Hostadapter im Computer durchgeführt, der auch das Laden der Treiber und die Grundkonfiguration vornimmt. Zusätzlich verbessert sich durch Hot-Plugging, das Hinzufügen und Entfernen von Peripherie-Geräten im laufenden Betrieb, die Bedienerfreundlichkeit.

Der USB erfüllt folgende Anforderungen:

- eine einheitliche Schnittstelle für alle Peripherie-Geräte
- eine mechanisch stabile und einfache Steckverbindung
- kleine platzsparende Stecker und Buchsen

Zusätzlich weisen alle USB-Spezifikationen folgende Eigenschaften auf:

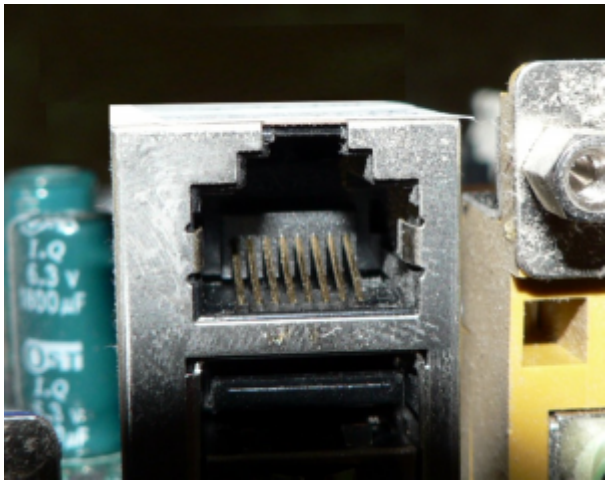
- billig
- abwärtskompatibel

USB-Version	USB 1.0/1.1		USB 2.0	USB 3.0	USB 3.1	USB 3.2
	Low Speed	Full Speed	High Speed	Super-Speed	Super-Speed-Plus	
Symbolrate	1,875 MBi/ts	15 MBit/s	600 MBit/s	5 GBit/s	10 GBit/s	20 GBit/s
Datenrate (brutto)	1,5 MBit/s	12 MBit/s	480 MBit/s	4 GBit/s	-	-
Datenrate (theoretisch)	188 kByte/s	1,5 MByte/s	60 MByte/s	600 MByte/s	1.200 MByte/s	
Datenrate (netto)	ca. 150 kByte/s	ca. 1 MByte/s	ca. 36 bis 44 MByte/s	ca. 480 MByte/s	ca. 800 MByte/s	ca. 1.600 MByte/s
Interface	UHCI/OHCI	UHCI/OHCI	EHCI	xHCI	-	-
Leitungslänge	5 m	5m	5m	3 m	1 m	1 m
Anwendungen	Maus Tastatur	Audio	Video, Speichermedien			

FireWire

Firewire wurde ursprünglich von Apple entwickelt und wurde dann aus urheberrechtlichen Gründen von Sony in i.LINK umbenannt. Eine weitere Bezeichnung für diese Schnittstelle ist der vom Standardisierungsgremium IEEE festgelegte Name IEEE 1394. Die besondere Fähigkeit von FireWire ist der schnelle Datenaustausch zwischen externem Gerät und Computer, mit einer Übertragungsrate von bis zu 800 MBit/s. Durch diese hohe Geschwindigkeit eignet sich diese Schnittstelle besonders gut zur Übertragung von Bildern und Videos und zum Anschluss von Videokameras, Festplatten und DVD-Brennern. Im Gegensatz zu USB 2.0 wird die theoretische Datenübertragungsrate nahezu erreicht.

Ethernet (RJ45 - Netzwerk)



Dient zum Senden und Empfangen von Daten über das Netzwerk.

PS/2

PS/2 ist eine **Computer-Schnittstelle bzw. ein Anschluss für Tastaturen oder Computer-Mäuse**. Die Abkürzung PS/2 stammt ursprünglich aus dem Jahr 1987, als IBM einen Personal Computer mit der Bezeichnung PS/2 auf den Markt brachte. Dieser PC hatte erstmals diesen Schnittstellen-Typ, der für die Tastatur verwendet wurde. Die Computer-Maus wurde an einer seriellen Schnittstelle angeschlossen. Später bekamen die PCs eine zweite PS/2-Schnittstelle für die Maus dazu.

PS/2 ist im Prinzip eine **serielle Schnittstelle**, die trotz der Universal-Schnittstelle USB nicht auszusterben scheint. Der Grund warum viele Motherboards immer noch PS/2-Schnittstellen für Maus und Tastatur haben liegt in der Nachfrage begründet.

Der **USB überträgt** von der Tastatur zum Rechner immer **sechs Befehle gleichzeitig**. Für einen schnellen Tipper ist das zu wenig. **PS/2 überträgt die Tastaturanschläge kontinuierlich an den Rechner**. Dadurch ist die Latenz niedriger. Die meisten Tastaturschreiber merken diesen Unterschied natürlich nicht. Auch aus Sicherheitsgründen kann PS/2 interessant sein. Während man einen **USB-Port** für alle möglichen Geräte **„missbrauchen“** kann, eignet sich der **PS/2 eben nur für den Anschluss einer Maus oder Tastatur**. Computer, bei denen USB-Ports elektrische oder mechanisch

blockiert sind, sind weniger anfällig für lokale Manipulationen durch „böse“ USB-Endgeräte. Leider kann man eine Tastatur am PS/2 im **laufenden Betrieb nicht ausstecken, ohne Abstürze oder ein stehendes System zu provozieren**. Hier ist USB einfach flexibler.

Eingabemedien

1. Nenne 4 Eingabegeräte!

Tastatur

1. Wie erhältst du das Eurozeichen?
2. Wozu dient die NumLock-Taste?
3. Was versteht man unter Scancode?

Rechner

Das Mainboard

1. Nenne 5 Komponenten des Motherboards!
2. Welche Mainboard-Formate kennst du?
3. Wozu dient der AGP-Slot? Wie heißt seine Weiterentwicklung?
4. Was versteht man unter dem Chipsatz?
5. Wie heißt die Schnittstelle(n) für Erweiterungskarten?
6. Wozu dient der Sockel? Welche verschiedene Sockelarten kennst du?

Der Prozessor (CPU - Central Processing Unit)

1. Was versteht man unter einem Mikroprozessor?
2. Was versteht man unter einem Mikrocontroller?
3. Wozu dienen die Register?
4. Welche Aufgabe hat die ALU?
5. Was ist der Unterschied zwischen Mehrkernprozessoren und Mehrprozessorsystemen?
6. Wie misst man die Geschwindigkeit von CPUs und was versteht man darunter?
7. Nenne 2 Hersteller von CPUs!
8. Nenne 4 typische Anwendungen für Single-Core CPUs!
9. Nenne 4 typische Anwendungen für Mehrkernprozessoren!
10. Warum gibt es Mehrkernprozessoren überhaupt?
11. Was ist die Problematik bei Mehrkernprozessoren bzgl. Leistungssteigerung?
12. Wie heißt der schnelle Zwischenspeicher zwischen CPU und RAM und was bedeuten dabei die verschiedenen Levels?

Speichermedien

Speicherkategorien

1. Nenne die drei Arten von Speichermedien und jeweils 2 Beispiele dafür!
2. Gib an, welche Speicherart am schnellsten und welche am langsamsten ist!

Magnetische Speicher

1. Wie lautet die Abkürzung für eine magnetische Festplatte und für was steht die Abkürzung!
2. Wie ist eine magnetische Festplatte aufgebaut?
3. Wie groß war die Kapazität einer 3 1/2- Zoll-Diskette?
4. Wieso sollte man Festplatten während des Betriebs keinen Stößen aussetzen bzw. warum sollte kein Staubkorn in das Innere einer magnetischen Festplatte gelangen?
5. Welche 4 Kriterien sind entscheidend für die Geschwindigkeit einer magnetischen Festplatte (HDD)?
6. Wie sind die Daten auf einer magnetischen Festplatte organisiert? Erkläre dabei die Begriffe Spuren, Zylinder und Sektoren (=Blöcke)!

Elektronische Speicher

1. Welche zwei elektronische Bauteile sind in einem elektronischen Speicher am häufigsten verbaut?
2. Wofür stehen die Abkürzung ROM?
3. Wofür stehen die Abkürzung RAM?
4. Was ist der Unterschied zwischen RAM und ROM?
5. Welche 2 Arten von RAM gibt es?
6. Welche 4 Arten von SDRAM gibt es?
7. Warum kommt es bei einem DDR-SDRAM zu einer verdopplung der Taktrate im Vergleich zum SDRAM?
8. Wie hoch sind die aktuellen Taktraten von den neuesten DDR4-SDRAM-Speicher?
9. Welche RAM-Bausteine sind heutzutage bei Computern in Verwendung?
10. Warum und worauf muss ich beim Motherboardkauf auf den RAM-Speicher Rücksicht nehmen?
11. Nenne 2 Primärspeicher und 2 Sekundärspeicher!
12. Welche Flash-Speicher kennst du?
13. Nenne 4 Vorteile und 2 Nachteile von Flash-Speicher!
14. Nenne 4 Anwendungen von Flashspeicher!
15. Warum sind SSDs schneller als HDDs?
16. Was ist eine SSHD und wie ist diese aufgebaut?
17. Welche Speicherkartenarten kennst du? Welche wird heutzutage in Handys verwendet?

Optische Speicher

1. Erkläre wofür die Abkürzungen CD, DVD, BD und UHD-BD stehen!
2. Wie viele MB kann eine CD speichern?
3. Wie viele GB kann eine DVD speichern?
4. Wie viele GB kann eine BD speichern?
5. Wie viele GB kann eine UHD-BD speichern?
6. Wieso kann eine DVD bei gleicher Größe mehr Daten fassen?
7. Wie bezeichnet man die Erhebungen und Vertiefungen auf einer CD bzw. DVD?
8. Wozu werden BD zumeist verwendet?

Schnittstellen

1. Was ist der Unterschied zwischen internen und externen Schnittstellen?
2. Welche interne Schnittstellen gibt es für Massenspeicher? Was sind die Unterschiede zwischen den einzelnen Schnittstellen?
3. Welche Geräte werden/wurden an die parallele Schnittstelle angeschlossen?
4. Was ist der große Vorteil von USB?
5. Für welche Geräte dient die externe DVI-Schnittstelle als Verbindung zum Computer?
6. Was ist der Unterschied zur externen VGA-Schnittstelle?

5) Präsentationstechniken

Unter **Präsentationstechnik** versteht man die Grundsätze, die eine Präsentation erfolgreich machen. Die Präsentation ist eine zweckbestimmte und empfangsorientierte Informationsbeschreibung, welche versucht, den Kommunikationsfluss zu verbessern und Expertenwissen anderen zugänglich zu machen.

Vorbereitung einer Präsentation

- Wem genau stelle ich meine Arbeit vor?
- Welche Einblicke soll das Publikum in meine Arbeit erhalten?
- Welche Kernbotschaften sollen sich schließlich im Gedächtnis festsetzen?
- Wodurch kann sich meine Präsentation von den anderen abgrenzen
- Welche Fragen könnte das Publikum stellen?

Zusätzlich ist zu beachten, dass pro Minute der Präsentation etwa 5 bis 30 Minuten an Vorbereitung einzuplanen sind. Je kürzer die Präsentation, desto kritischer ist die Vorbereitung.

Gliederung

Einstiegsphase

- kurze Begrüßung und Selbstvorstellung
- knappe Erläuterung des persönlichen Zugangs zum Thema
- Vorstellung der Fragestellung und der Präsentationsziele

Je interessanter und überraschender die Präsentation beginnt, desto einfacher gelingt die Kontaktaufnahme mit der Prüfungskommission und desto angenehmer ist die Gesprächsatmosphäre. Die Möglichkeiten sind vielfältig: Gemeinsamkeit mit Zuhörern herstellen, sachlich und ernst beginnen, persönliche Erfahrungen einbringen, Vergleiche herstellen, Anschauungsmaterial einsetzen, historische Rückschau halten, rhetorische Frage stellen, auf aktuelles Geschehen hinweisen, ein Zitat voranstellen, …

Hauptteil

- 3 - 5 Module
- Vorstellung der Vorgangsweise/Methodik
- Vermittlung der Kernbotschaften

Ausstiegsphase

- Zusammenfassung der Kerninhalte
- Ausblick

- Dank an das Publikum und Überleitung zur Diskussion

Der Schluss bleibt am längsten in Erinnerung und ist daher mindestens ebenso wichtig wie die Einstiegsphase.

Dimensionen der Verständlichkeit

- Einfachheit – Vermeide Kompliziertheit!
- Gliederung, Ordnung – Vermeide Unübersichtlichkeit!
- Kürze, Prägnanz – Vermeide Weitschweifigkeit!
- Zusätzliche Stimulanz – Vermeide einkehrende Langeweile und wecke Emotionen!

„Nichts ist schwerer, als bedeutende Gedanken so auszudrücken, dass jeder sie verstehen muss.“ (Schopenhauer)

- kurze Sätze verwenden
- Einschiebungen vermeiden
- Fremdwörter sparsam dosieren
- Füllwörter ausmerzen
- Vergleiche/ Beispiele bringen
- Standardsprache verwenden

PowerPointPräsentation – praktische Tipps

Zielsetzung

- Sichtbarmachen einer Gliederung
- Schaffung von Orientierung
- Visualisierung (komplexer) Inhalte durch Grafiken, …

Layout

- angemessenes Foliendesign
- dezente Übergänge
- sparsame Effekte
- einheitliches Muster (in Bezug auf Aufzählungszeichen, Einrückung, Farbe, Schriftgröße, …)
- ansprechende Gestaltung der Titelfolie (evtl. grafischer Eyecatcher)
- bewusster Verzicht auf „Danke für die Aufmerksamkeit“-Folien

Reduktion und Übersichtlichkeit

- „Weniger ist mehr“
- überschaubare Anzahl von Folien
- wenig Text auf den einzelnen Folien (Stichwörter!)
- „Ein Bild sagt mehr als tausend Worte“

Korrektheit

- inhaltlich einwandfrei
- stilistisch ansprechend (Ausdruck)
- sprachlich richtig (Rechtschreibung, Grammatik)

Wirkung durch Körpersprache

- authentisch
- Blickkontakt mit dem Publikum (→ Präsenz)
- Mimik: Ein Lächeln wirkt Wunder.
- Gestik: Was du sagst, „liegt auf der Hand“.
- Körperhaltung: Der Körper ist der Übersetzer der Seele ins Sichtbare.
- äußere Erscheinung: You never get a second chance to make a first impression!

Stimme macht Stimmung

Haltung

Eine gute Grundhaltung sorgt für eine funktionierende Atmung und somit für eine natürliche, klangvolle Stimme.

Artikulation

Um Nuscheln vorzubeugen und den Mundraum zur Verstärkung des Stimmklangs zu nutzen, bleibt der Unterkiefer locker und der Mund beim Sprechen geöffnet.

Stimmdynamik

Dynamisch ist eine Sprechweise, die sehr lebendig ist und auch Sprechpausen beinhaltet. Wörter werden sinnvoll betont (laut/leise), die Sätze werden melodiös gesprochen (hohe/tiefe Stimme), Satzteile werden manchmal kurz, manchmal gestreckt gesprochen (dynamisches Sprechtempo).

Notizen

- PowerPointPräsentation, Flipchart, … als Stichwortgeber
- allenfalls Verwendung von Karteikarten (DIN A6, Karton)
- ein Leitgedanke und drei bis fünf weiterführende Stichworte pro Kärtchen
- Beschreibung der Kärtchen auf der Vorderseite
- Durchnummerierung der Kärtchen
- freie Rede anhand der Leitgedanken
- Vergewisserung anhand der Stichworte

- Tipps für eine gute Präsentation

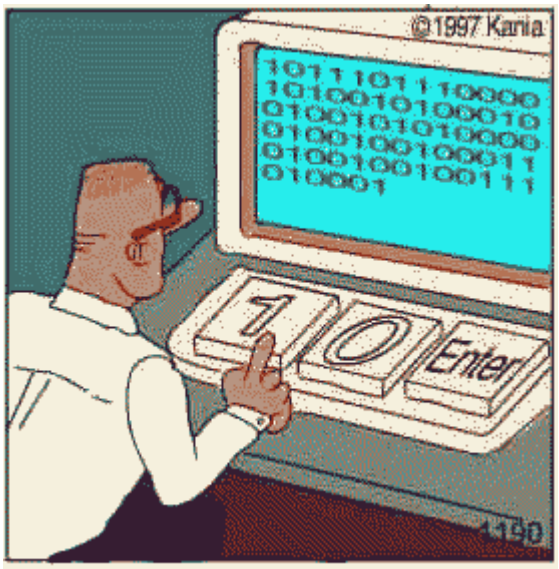
Powerpoint

- Übungen zu Powerpoint

Prezi

- http://prezi.com/
- Prezi-ähnliche Systeme

6) Algorithmik und Programmierung



Real programmers code in binary.

6.1) Grundlagen zur Programmierung

Programm

Im Alltag sind uns Programme geläufig, wie z.B.:

- das Programm einer Feier,
- ein Kochrezept,
- oder das Programm einer politischen Partei.

Diese beschreiben einen Ablauf d.h. was wie zu tun ist.
In der Informatik wird der Begriff enger gefasst und man spricht von der Programmierung.

Programmierung

Unter Programmierung versteht man das Schreiben solcher Programme in Form von Quellcode d.h. Befehle in einer Text Datei.

Das Verfassen von Quellcode erfolgt in einer Programmiersprache (C++, Java, Python, Fortran, …)

Diese Programmiersprachen werden von Softwareprogrammen in Maschinensprache übersetzt, welche ein Computer versteht und ausführen kann.

Somit kann man mit Programmierung das Verhalten eines Computers steuern.

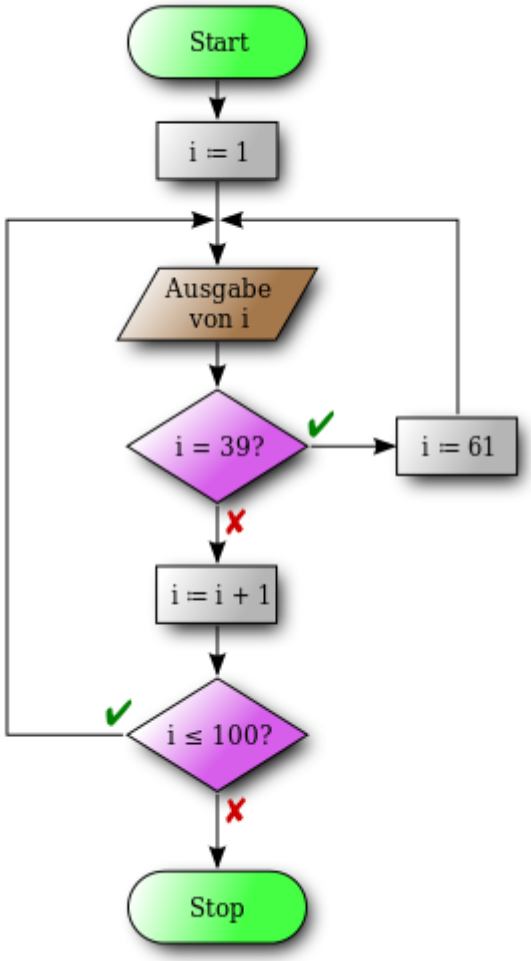
Algorithmus

Ein Algorithmus ist eine eindeutige Abfolge von Anweisungen (Befehlen) zur Lösung eines Problems.

Beispiel eines Algorithmus:

```
ALGORITHMUS mach_was()
   ANFANG mach eine Aufgabenliste
   SOLANGE was zu tun ist
   WENN nicht erledigt
      mach weiter
   SONST
      geh zur nächsten Aufgabe
   WIEDERHOLE
ENDE
```

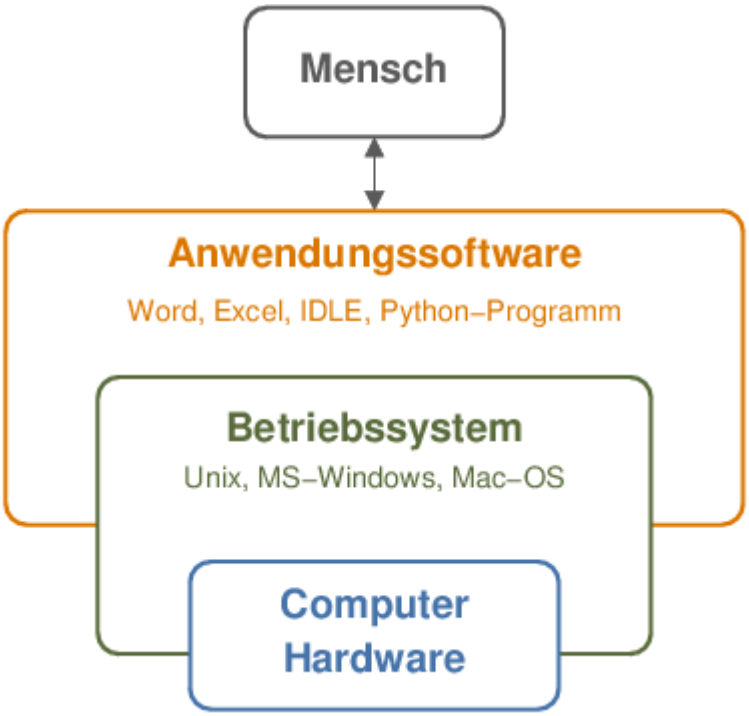
Ein Algorithmus kann auch graphisch in Form eines Flussdiagramms dargestellt werden.



Ein Programm besteht im Allgemeinen aus vielen Algorithmen!

Computersystem

Ein Computersystem kann als Schichtmodell gesehen werden.



6.2) Programmiersprachen

Eine Programmiersprache zeichnet sich - wie jede Sprache - durch Syntax und Semantik aus.

Syntax

Legt fest in welcher Folge Zeichen den Programmtext bilden.

Z.B. Python-Syntax zur Definition einer Variablen:

```
a = 1.843
```

Nicht erlaubt wäre z.B.:

```
a 1.843
```

Bei anderen Programmiersprachen würde dies wie folgt aussehen:

```
real a = 1.843     # Fortran
float a = 1.843;   # C++, Java
$a = 1.843;        # Perl
```

Semantik

Beschreibt die Bedeutung eines Programmtextes

```
print "Alles Python oder was?"
```

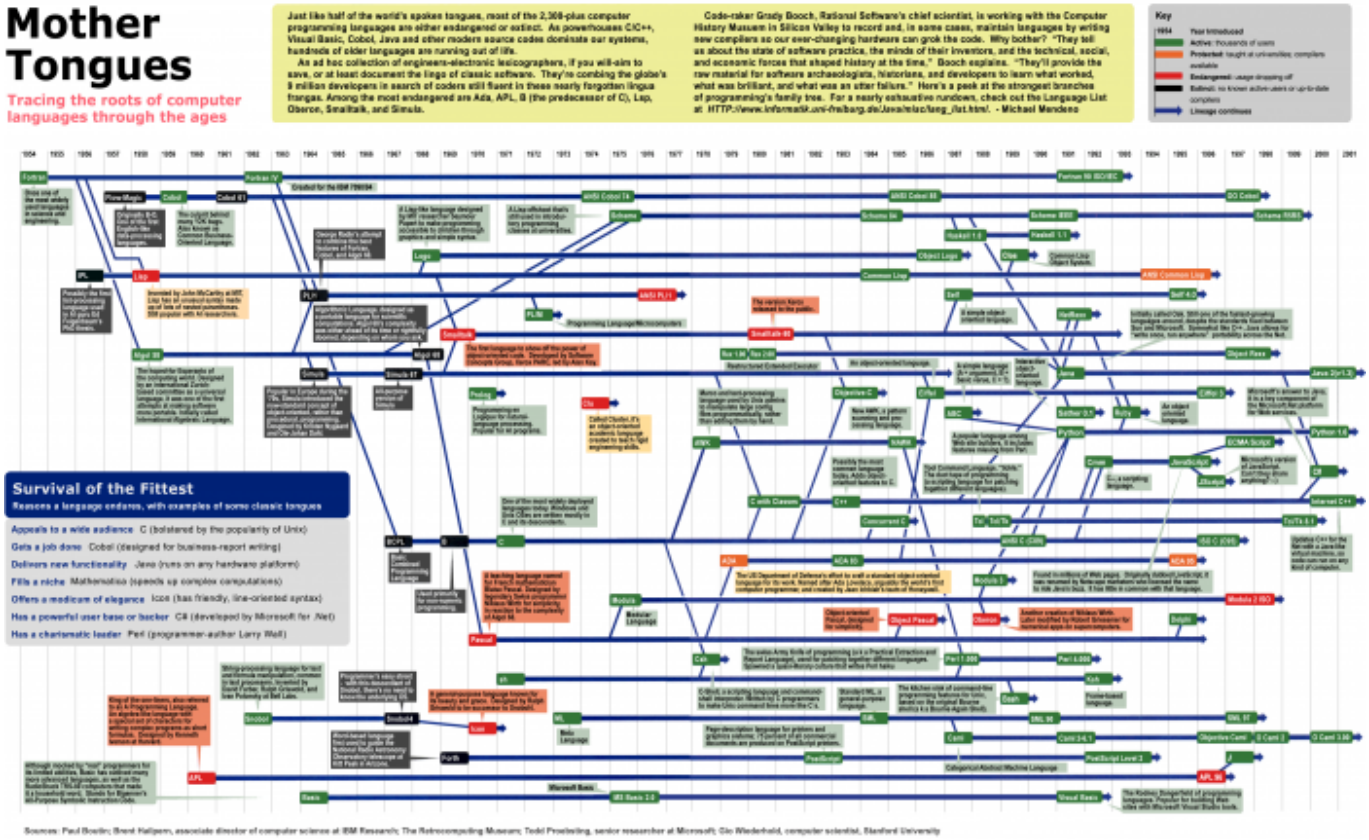
Dieser Programmtext gibt einen Text am Bildschirm aus. Ein weiteres Beispiel :

```
if a < 0 :
  a = a*(-1)
```

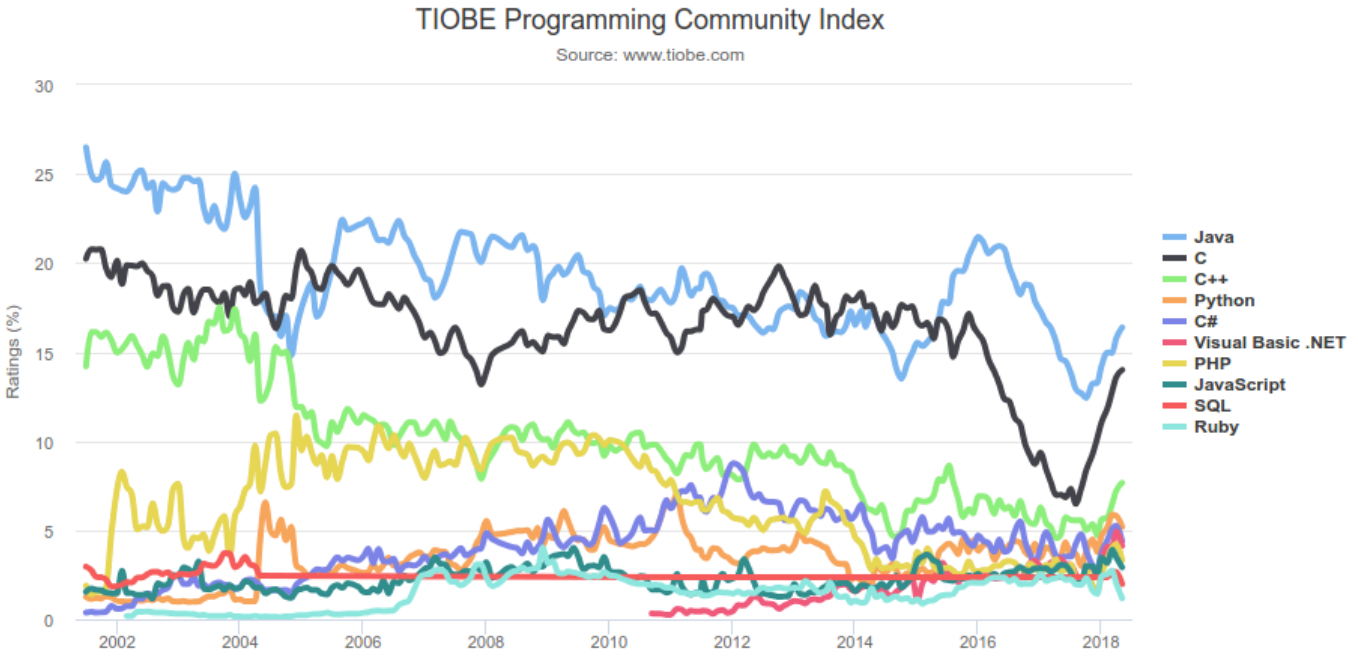
Dieser Code macht aus a eine positive Zahl.

Wichtige Programmiersprachen

Die Liste der Programmiersprachen ist lang. Anbei eine Abbildung über die Entwicklung wichtiger Programmiersprachen.



Anbei noch eine Abbildung über die Popularität von Programmiersprachen



Aber auch Experten wie **Bjarne Stroustrup** - der **Erfinder von C++** meint dazu im Video zu den 5 wichtigsten Programmiersprachen:

Video

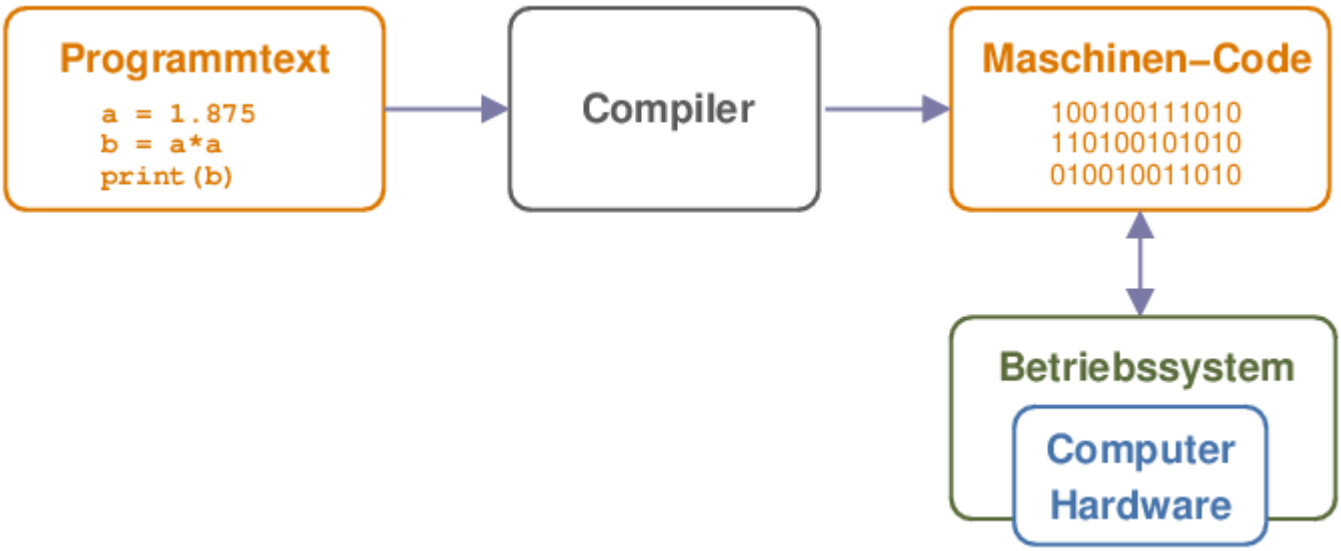
… Experten kennen mehr als nur eine Sprache, kennt man diesen Cluster, kennt man alle anderen!

6.3) Übersetzung von Quellcode in Maschinensprache

Es gibt **2 unterschiedliche Methoden** wie **Programmtext (Quelltext, source code) in Computersprache übersetzt** wird d.h. vom Computer **verstanden** wird.

Compiler

Der Programmtext wird nach Erstellung komplett in Maschinen-Code (executeable) übersetzt (siehe Arbeitsweise eines Compilers) und ausgeführt.

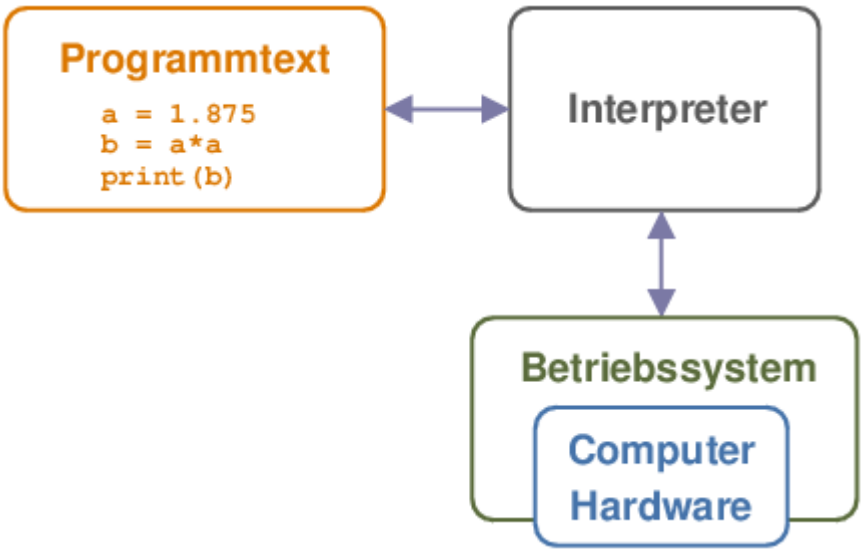


Vor- und Nachteile

- benötigt Compiler (=eigene Anwendungssoftware)
- läuft nur auf dem Betriebssystem, wo Programm kompiliert wurde
- i.a. sehr schnell (high performance)
- Quelltext nach kompilieren nicht einsehbar
- typische Sprachen: Fortran, C++

Interpreter

Der Programmtext wird Zeile für Zeile an das Betriebssystem geschickt und ausgeführt (siehe Arbeitsweise eines Interpreters).



Vor- und Nachteile

- benötigt Interpreter (=eigene Anwendungssoftware)
- unabhängig vom Betriebssystem
- schnell implementiert, langsam exekutiert
- Quelltext einsehbar, schnell änderbar
- typische Sprachen: JavaScript, Python

6.4) Programmierstile

Quellcode kann **imperativ (z.B. prozedural)** oder **objektorientiert** geschrieben werden.

Imperativ

Kommt aus dem lateinischen imperare = anordnen d.h. einzelnen Befehle werden **sequentiell hintereinander** geschrieben mittels:

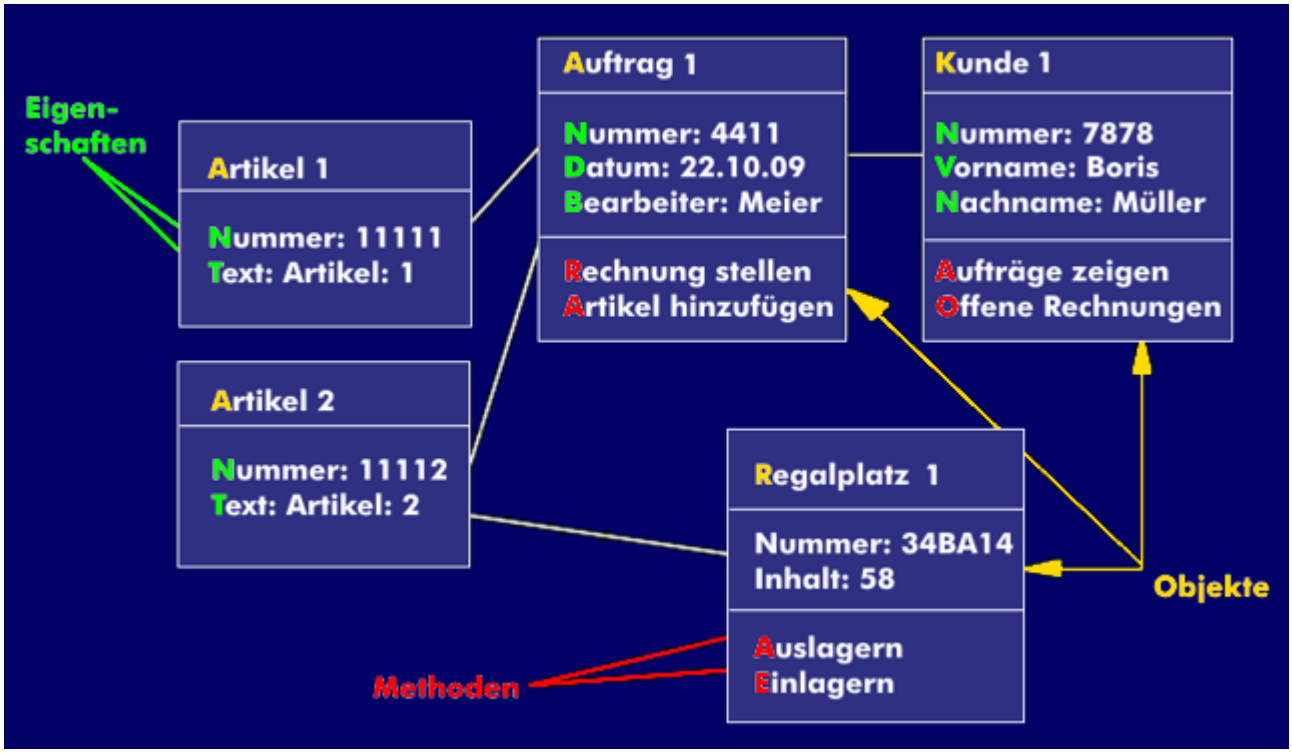
- Kontrollstrukturen = Strukturierte Programmierung
- Überschaubare Unterprogrammen = Prozedurale Programmierung
- Logisch zusammengefassten Unterprogrammen = Modulare Programmierung

Vor- und Nachteile

- für kleine Programme geeignet
- i.a. sehr schnell, da Compiler den Code gut optimieren kann
- wird schnell unübersichtlich, schwer wartbar, ...

Objektorientiert

Das Programm wird als **System von Objekten** dargestellt, die **miteinander kommunizieren** (siehe Beispiel objektorientierte Programmierung).



Erklärungen:

- Zunächst definiert man **sogenannte Klassen** (z.B. Klasse “Artikel”). Ein Programm kann beliebig viele unterschiedliche Klassen haben (Artikel, Auftrag, Kunde, ...).
- Erzeugt man nun einen bestimmten “Artikel” (z.B. Artikel 1, anders gesagt gibt man diesen spezifische Daten), spricht man von einem **Objekt**.
- Diesen Vorgang bezeichnet man auch als **das Objekt (Artikel 1) wird von der Klasse (Artikel) instantiiert**. Man kann viele Objekte einer Klasse erzeugen (Artikel 1, Artikel 2, ..).
- Jedes Objekt besitzt (anhand der Klassendefinition) **Eigenschaften (Attribute)**, dies sind die Daten (z.B. Nummer, Text, ...) und **Methoden (Algorithmen)**, welche diese Daten ändern, anzeigen, bereitstellen, etc (z.B. auslagern, einlagern, ...).
- Ein Hauptprogramm enthält z.B. alle hier dargestellten Objekte in Form von Listen und steuert den Programmablauf inklusive Daten Ein- und Ausgabe.

Vor- und Nachteile

- für große Programmierprojekte gut geeignet
- Konzepte wie Vererbung, Datenkapselung, Polymorhismus, etc umsetzbar

Darstellung von Algorithmen bzw. von Programmlogiken

Es gibt verschiedene Darstellungsformen, um algorithmische Grundbausteine und Algorithmen darzustellen. An dieser Stelle soll auf die drei wichtigsten eingegangen werden:

- Pseudocode
- Struktogramm
- Programmablaufplan

Pseudocode

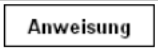
Der Pseudocode stellt einen **Algorithmus in einer Schreibweise** in der Art einer Programmiersprache dar, die aber in der Regel **näher an der natürlichen Sprache ist als am formalen Programmcode**. Es gibt **keine Standardisierung bzw. Vorschrift**, wie ein Pseudocode auszusehen hat; die Ablaufstrukturen werden grob durch Texte und fest vorgeschriebene Schlüsselwörter beschrieben. Guter Pseudocode zeichnet sich durch eine **kurze präzise Beschreibung des Algorithmus** aus.

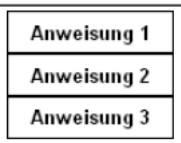
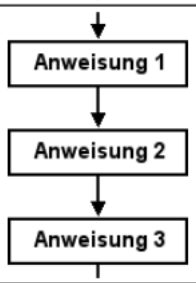
Struktogramm

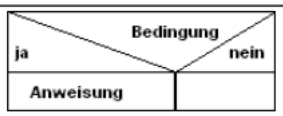
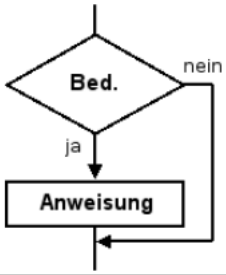
Struktogramme (auch nach ihren Entwicklern **Nassi-Shneidermann-Diagramme** genannt) ist eine **genormte (DIN 66261) Entwurfsmethode für Algorithmen**. Die Methode **zerlegt ein Gesamtproblem**, das man mit dem gewünschten Algorithmus lösen will, in immer **kleinere Teilprobleme** bis schließlich nur noch **elementare Grundstrukturen wie Sequenzen und Kontrollstrukturen** zur Lösung des Problems übrig bleiben.

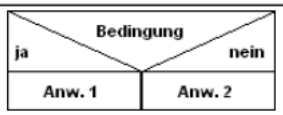
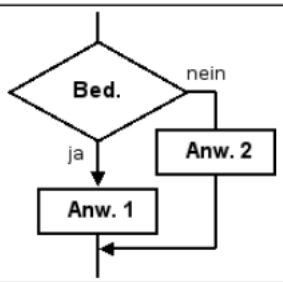
Programmablaufplan (PAP)

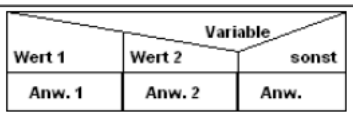
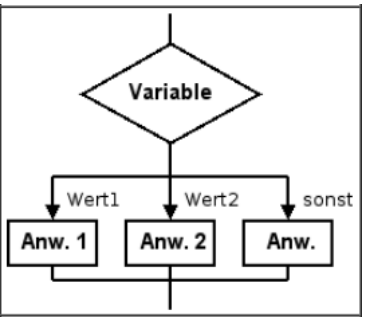
Der **Programmablaufplan (auch Flussdiagramm oder Programmstrukturplan)** ist eine **graphische Darstellung zur Umsetzung eines Algorithmus** in einem Programm und beschreibt die **Folge von Operationen zur Lösung einer Aufgabe**. Die Symbole für Programmablaufpläne sind in der DIN 66001 genormt.

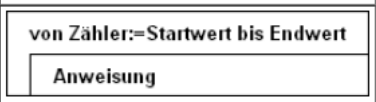
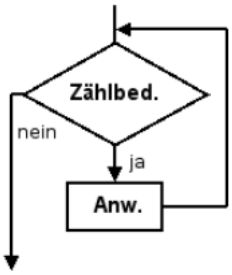
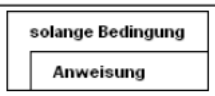
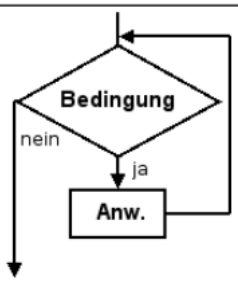
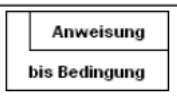
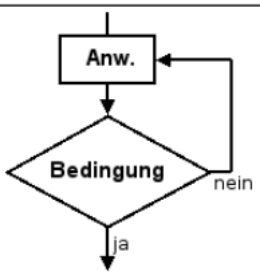
Strukturelement	Pseudocode	Struktogramm	Programmablaufplan
Verarbeitung (Elementarblock)	`Anweisung`		

Reihenfolge (Sequenz)	`Anweisung1`
	`Anweisung2`
	`Anweisung3`
			

Verzweigung (Alternative)	`wenn Bedingung` `  dann Anweisung`		

	`wenn Bedingung`
	`  dann Anweisung1`
	`  sonst Anweisung2`
			

Fallunterscheidung	für Variable
	  Wert 1 : Anweisung1
	  Wert 2 : Anweisung2
	  ...
	  [sonst Anweisung]
			

Zählschleife	von Zähler:=Startwert bis Endwert Anweisung		
anfangsgeprüfte Schleife	solange Bedingung Anweisung		
endgeprüfte Schleife	wiederhole Anweisung bis Bedingung		

Code.org

Code.org ist eine 2013 in den USA gegründete Non-Profit-Organisation und gleichnamige Website, die möglichst viele Menschen und insbesondere Kinder für das Thema Informatik (im englischen „Computer Science“) und Programmieren begeistern will.

- Kurs 3
- Kurs 4

Scratch

Mit Scratch kannst du deine eigenen interaktiven Geschichten, Spiele und Animationen programmieren und deine Kreationen mit anderen in der Gemeinschaft online teilen.

Scratch hilft jungen Leuten, kreativ zu denken, systematisch zu schließen und miteinander zusammenzuarbeiten — grundlegende Fähigkeiten für das Leben im 21. Jahrhundert.

Beispielhafte Scratch-Projekte

- Scratch-Games

C++ bzw. CPP

C + + (zeh plus plus ausgesprochen) ist eine kompilierte Allzweck Programmiersprache. Ihre Komplexität liegt zwischen Einsteiger und Fortgeschritten, da sie sowohl High-Level als auch Low-Level Sprachelemente vereint. Sie bietet imperative, objekt-orientierte und generische Programmiermerkmale.

C + + ist eine der beliebtesten Programmiersprachen und ist auf vielen Hardwareplattformen und Betriebssystemem verfügbar. Da sie besonders effizient ist, wird es für Systemprogramme, Applikationen, Gerätetreiber, eingebettete Software, Server-Client und Unterhaltungsprogramme, z. B. Videospiele, eingesetzt. Verschiedene Hersteller bieten Open Source und proprietäre C + + Compiler an, bspw. FSF, LLVM, Microsoft und Intel.

Entwicklungsumgebung DEV C++

- Download DEV C++

Los gehts!

- Einstieg in C++
- WikiBook C++
- Learn C++
- C++ spielerisch lernen

Programmierung in C++

- 6.8.1) Grundgerüst
- 6.8.2) Variablen
- 6.8.3) Datentypen
- 6.8.4) Verarbeitung
- 6.8.5) Ein- und Ausgabe
- 6.8.6) Ablaufsteuerung

6.8.1) Grundgerüst eines Programms

Ein **Programm** ist eine **Aneinanderreihung von Befehlen**, die dem Computer zur Abarbeitung zugeführt werden können. Wie schon erwähnt, wird ein **C++-Programm in einem Texteditor** geschrieben und dann von einem **Compiler in ausführbaren Code übersetzt**. Danach kann es gestartet werden.

Hauptprogramm - main()

Ein C++-Programm besteht **mindestens** aus der **Funktion main()**. Wichtig ist, dass Sie in **jedem C++-Programm irgendwo den Namen main** mit einem **Klammerpaar** finden werden. In manchen Fällen stehen **zwischen den Klammern Variablen für die Parameterübergabe**. Aber auch das sollte Sie zu Anfang nicht stören. Ignorieren Sie es von ganzem Herzen, bis Sie wissen, wozu Sie es brauchen.
Die **Hauptfunktion beginnt direkt nach einer öffnenden geschweiften Klammer und endet mit der schließenden Klammer**. **Zwischen den geschweiften Klammern** stehen die **Befehle der Hauptfunktion**. Sie werden nacheinander ausgeführt. Diese Hauptfunktion enthält nichts und ist damit das kleinste denkbare C++ Programm.

```
int main()   //Hauptfunktion main()
{

   return 0;  //Optionale Rückgabe an das Betriebssystem
}
```

Dieses Programm tut gar nichts. Wenn es etwas täte, stünden die Befehle zwischen den geschweiften Klammern.

Kommentare

Wenn wir schon einmal ein Programm haben, das nichts tut, wollen wir es auch um **Befehle** erweitern, **die nichts tun: KOMMENTARE.**

Freundlichkeit unter KollegInnen

Sie können in das Programm Kommentare einfügen, die **vom Compiler völlig ignoriert** werden. Der **Zweck** solcher Kommentare ist es zu **dokumentieren**, warum das Programm so und nicht anders geschrieben wurde. Die **Kommentare richten sich an Programmierer**. Dies können **Kollegen** sein, die Ihr **Programm korrigieren** oder **erweitern** sollen. Aber noch öfter helfen Sie sich selbst damit. In der Praxis ist es so, dass Sie vermutlich auch herangezogen werden, wenn eines Ihrer Programme nicht korrekt läuft oder ergänzt werden soll. Da Sie zwischendurch andere Programme geschrieben haben werden, vielleicht geheiratet haben, umgezogen sind und noch drei weitere Programmiersprachen gelernt haben, werden Sie sich **nicht mehr an jedes Detail erinnern**. Sie werden dankbar sein, wenn Sie in den Programmen ausführliche Hinweise finden, wie und warum es

so funktioniert oder zumindest funktionieren soll.

Es gibt **zwei Arten**, einen Text davor zu schützen, dass der Compiler versucht, ihn als Programm zu interpretieren.

Zeilenweise

Die einfachere Art der Kommentierung ist es, **zwei Schrägstriche** direkt hintereinander zu setzen. Damit gilt der **Rest der Zeile als Kommentar**.

```
int main()
{
    // Hier beginnt der Kommentar.
    // Die naechste Zeile braucht ihr
    // eigenes Kommentarzeichen.

    return 0;
}
```

Kommentarblock

Daneben gibt es die Möglichkeit, einen **größeren Text in Kommentarklammern einzuschließen** und damit dem Compiler zu entziehen. Der Anfang des Kommentars wird durch den Schrägstrich, gefolgt von einem Stern, festgelegt. Der Kommentar endet mit der umgekehrten Zeichenfolge, also mit einem Stern und dann einem Schrägstrich.

```
int main()
{
  /* Hier beginnt der Kommentar.
     Die naechste Zeile braucht kein
     eigenes Kommentarzeichen.
   */

   return 0;
}
```

!! ACHTUNG !! : Eine Verschachtelung von Kommentarblöcken ist nicht möglich!

```
int main()
{
  /* Hier beginnt der Kommentar
     /*
       Die naechste Zeile braucht kein
       eigenes Kommentarzeichen
      */
     Dies wird der Compiler wieder uebersetzen wollen.
  */
```

```
  return 0;
}
```

Anweisungen

Anweisungen sind die **Befehle, aus denen ein Programm** besteht. Eine Anweisung kann dazu dienen, etwas zu **speichern**, zu **berechnen**, zu **definieren** oder auf dem Bildschirm **auszugeben**. **Anweisungen sind die Grundeinheiten**, aus denen **Programme bestehen**.

Blöcke

Zusammenfassen von Anweisungen

Mehrere Anweisungen können **zu einem Block zusammengefasst** werden. Ein Block wird durch **geschweifte Klammern** eingefasst. Der **Compiler** wird einen **Block wie eine einzige Anweisung** behandeln. Sicher ist Ihnen schon aufgefallen, dass die Hauptfunktion main() ebenfalls solche **geschweiften Klammern** hat. In der Tat ist dies der Block, in dem die Befehle des Programms zusammengefasst werden.

Einrückungen

Um die **Übersicht zu erhöhen**, pflegt man **alle Befehle innerhalb eines Blocks einzurücken**. Achten Sie vor allem am Anfang darauf, dass die **zusammengehörigen geschweiften Klammern auf einer Ebene** stehen. Das folgende Beispiel mit Pseudobefehlen zeigt, wie das Einrücken korrekt ist.

```
int main()
{
    Hier sind Pseudo-Anweisungen;
    Diese gehört dazu;
    {
        Neuer Block, also einrücken;
        Wir bleiben auf diesem Niveau;
        Das kann ewig weitergehen;
    }
    Nun ist die Klammer geschlossen;
    Und die Einrückung ist auch zurück;

    return 0;
}
```

Hello World!

Es ist eine **alte Tradition**, eine **neue Programmiersprache** mit einem **„Hello-World“-Programm**

einzuweihen. Auch dieses Buch soll mit der Tradition nicht brechen, hier ist das „Hello-World“-Programm in C++:

```
#include <iostream>                                    // Ein- und
Ausgabebibliothek

int main(){                                            // Hauptfunktion
    std::cout << "Hallo, du schöne Welt!" << std::endl; // Ausgabe

    return 0;                                          // Optionale
Rückgabe an das Betriebssystem
}
```

Namensraum - namespace std

Die C++-Standardbibliotheken verwenden für alle Funktionen und Variablen den Namensraum std. Bei Verwendung von Typen, Funktionen oder Variablen aus den Standardbibliotheken müssten Sie jeweils std:: voranstellen. Sie können aber auch mit dem Befehl **using** erklären, dass Sie den Namensraum **std** verwenden wollen, als seien alle darin definierten Variablen und Funktionen ohne Namensraum definiert. Die Anweisung dazu lautet:

```
using namespace std;
```

6.8.2) Variablen

Die folgenden 4 Aussagen erläutern die Notwendigkeit von Variablen:

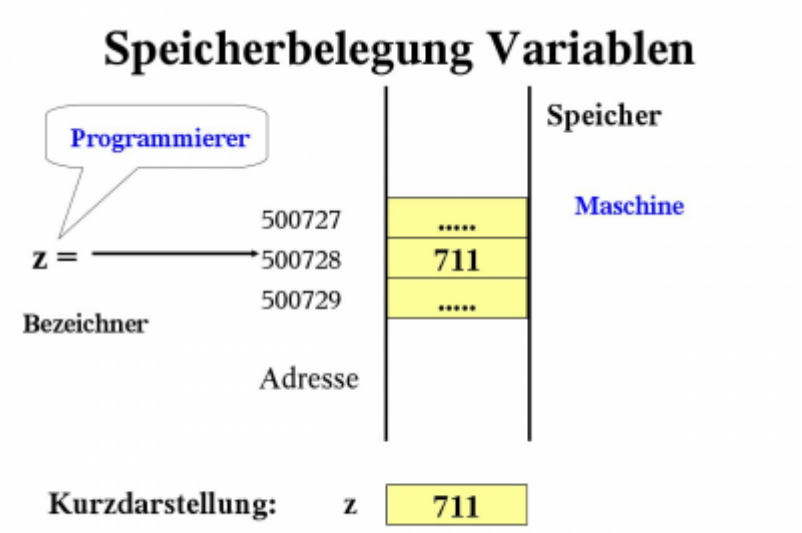
- Jedes Programm verarbeitet Informationen.
- Diese Informationen liegen im Speicher.
- Höhere Programmiersprachen greifen nicht direkt auf den Speicher zu, sondern verwenden Variablen.

⇒ **Variablen sind also die Behälter, in denen das Programm Zahlen und Texte ablegt.**

Eigenschaften einer Variable

SPEICHER

Die **Variable benötigt** zum Ablegen der Informationen immer **Speicher**. Der Speicher wird **vom Compiler zur Verfügung gestellt**. Über die **Lage und Größe muss sich der Programmierer NICHT kümmern**. Der Compiler ermittelt die Größe vom Datentyp und reserviert den dafür notwendigen Speicher.



NAME

Die Variable wird im Programm über einen **weitgehend frei wählbaren Namen** angesprochen. Dieser **Name** identifiziert die Variable **innerhalb eines Blocks eindeutig**. C++ unterscheidet sogar zwischen **Groß- und Kleinschreibung**!

DATENTYP

Der **Datentyp** einer Variablen **bestimmt, welche Informationen abgelegt werden können**. So kann eine Variable je nach ihrem Typ beispielsweise einen Buchstaben oder eine Zahl speichern. Der **Datentyp bestimmt** natürlich auch die **Speichergröße**, die benötigt wird. Der Typ bestimmt aber auch, **welche Operationen auf die Variable angewendet werden können**. So können Sie beispielsweise Zahlen durch einander dividieren. Mit Texten geht das nicht.

Variablendefinition

Das folgende Beispiel zeigt eine Variablendefinition innerhalb der Hauptfunktion main():

```
int main()
{
   int Gehalt;
}
```

Datentyp

Der Datentyp ist in diesem Fall **int**. Dieser Datentyp steht für ganzzahlige Zahlen.

Name

Durch ein Leerzeichen abgesetzt, beginnt der Name der Variablen. Den **Namen** sollten Sie immer so wählen, dass Sie **auf den Inhalt schließen können**. Hier lässt der Name Einkommen bereits auf die Verwendung der Variablen im Programm schließen. **Verwenden Sie ruhig lange Namen**. Abkürzungen ersparen zwar Tipparbeit, Sie werden aber auf lange Sicht mehr Zeit verschwenden, wenn Sie darüber nachdenken müssen, was in welcher Variable abgelegt ist. Das gilt auch dann, wenn Sie extrem langsam tippen. Noch mehr Zeit werden Sie brauchen, wenn Sie einen Fehler suchen müssen, der entstand, weil Sie zwei Variablen verwechselt haben, nur weil der Name unklar war.

Semikolon (=Strich-Punkt)

Zu guter Letzt folgt das Semikolon. **Damit wird jede Anweisung abgeschlossen, auch eine Variablendefinition.**

Initialisierung

Sie können **Variablen gleich bei ihrer Definition mit einem Wert vorbelegen**. Dazu setzen Sie hinter den Variablennamen ein Gleichheitszeichen. Das **initialisiert die Variable mit dem nachfolgenden Wert**.

```
   int Gehalt=2000;
```

Damit wird bei der Erstellung die Variable Gehalt auf 2000 gesetzt. Normalerweise initialisiert man

einen neue Variable immer mit 0.

```
   int Gehalt=0;
```

Mehrfache Definition

Es können mehrere Variablen mit dem gleichen Datentyp hintereinander definiert werden, in dem sie durch Komma getrennt werden.

```
   int zahl, Gehalt=0, Freunde=1, Feinde=5;
```

Geltungsbereich einer Variable

Eine Variable in C++ kann je nach ihrer Definition unterschiedliche Geltungsbereiche besitzen.

Blockgrenzen

Während **in C** Variablendefinitionen **nur am Blockanfang vor der ersten Anweisung** erlaubt sind, kann in **C++ eine Variable überall definiert werden**.

WICHTIG: Sie **gilt** dann **für den gesamten restlichen Bereich des aktuellen Blocks und aller darin verschachtelten Blöcke**.

Es können sogar **in verschachtelten Blöcken Variablen mit dem gleichen Namen** verwendet werden. Dabei **überdeckt die innen liegende Definition diejenige, die außerhalb des Blocks liegt**. Folgendes Beispiel macht das deutlich:

```
int main()
{
    int a = 5;
    {
    // hier hat a den Inhalt 5
    }
    {
        int a = 3;
        // hier hat a den Inhalt 3
        {
            // a ist immer noch 3
        }
    }
    // hier ist a wieder 5

    return 0;
}
```

Lokale Variable

Die **innere Variable wird als lokale Variable** bezeichnet. Für ihren Geltungsbereich **überdeckt sie die außen liegende Variable a**. Die **außen liegende Variable existiert durchaus noch**, aber **aufgrund der Namensgleichheit** mit der lokalen Variablen **kann man bis zu deren Auflösung nicht auf sie zugreifen**. Sobald das Programm den Block verlässt, in dem die lokale Variable definiert ist, wird die äußere Variable wieder sichtbar. **Alle Operationen auf der lokalen Variablen berühren die äußere Variable nicht**.

Globale Variable

Variablen, die **außerhalb jedes Blocks definiert** werden, nennt man globale Variablen. Sie **gelten für alle Blöcke**, die nach der Definition im Quelltext auftreten, **sofern darin nicht eine Variable gleichen Namens lokal definiert wird**.

Konstante

Eine Konstante ist ein **unveränderlicher Wert**. Alle Zahlen in einem Programm sind Konstanten, da ihr Wert, im Gegensatz zu Variablen, nicht geändert werden kann. In C++ haben **auch Konstanten einen Datentyp**.

Deklaration einer Konstanten

Zahlenkonstanten die ohne weitere Erklärung im Programm (=Quellcode) stehen, haben keinen Dokumentationswert und können zu Verwechslungen bzw. Verwirrungen führen. Um die Lesbarkeit des Programms zu erhöhen, könnte man anstatt der Zahl eine Konstante definieren.
Nehmen Sie an, sie möchten ein Programm für die Zeiterfassung ihrer Mitarbeiter schreiben. Jeder Mitarbeiter soll jeden Tag 8 Stunden arbeiten. Anstatt die Zahl 8 zu verwenden, könnten Sie eine Konstante namens Regelarbeitszeit verwenden.
Ist diese Konstante einmal definiert so kann sie bei Programmausführung nicht mehr verändert werden.
Notwendig dafür, ist neben dem Datentyp und dem Variablennamen noch das Schlüsselwort **const**.

```
   const int Regelarbeitszeit=8;
```

Manchmal findet man in C++ auch noch die Form aus der Programmiersprache C.

```
  #define Regelarbeitszeit 8
```

Beispiel für Variablen und ihren Geltungsbereich

```
#include <iostream>         //Bibliothek für Ein- und Ausgabe

#define Wochentage 7      //Alte Definition einer Konstanten namens
Wochentage mit dem Wert 7
```

```
int Arbeitsstunden=8;       //globale Variable

int main()       //Hauptfunktion
{
    const int Arbeitstage=5;       // Deklaration einer Konstanten namens
Arbeitstage initialisiert mit 5
    std::cout << Arbeitsstunden;      //Ausgabe der globalen Variable
Arbeitsstunden => 8
    {
        int Arbeitsstunden=5;         //Deklaration einer lokalen Variable
namens Arbeitsstunden initialisiert mit 5
        std::cout << Arbeitsstunden;        //Ausgabe der lokalen Variable
Arbeitsstunden => 5
    }

    std::cout << Arbeitsstunden;  //Ausgabe der globalen Variable
Arbeitsstunden => 8
    //Arbeitstage=4;                  --> NICHT ERLAUBT, DA Arbeitstage EINE
KONSTANTE IST
    std::cout << Arbeitstage;     //Ausgabe der Konstante Arbeitstage => 5
    std::cout << Wochentage;      //Ausgabe der Konstante Wochentage => 7

    return 0;
}
```

6.8.3) Datentypen

C++ bietet viele eingebaute Datentypen als auch Möglichkeiten Datentypen selbst zu definieren.

In C++ gibt es prinzipiell 7 grundlegende Datentypen:

- bool ⇒ true oder false
- char ⇒ Buchstabe (8 Bit)
- int ⇒ Ganze Zahl (32 Bit)
- float ⇒ Fließkommazahl (32 Bit)
- double ⇒ Fließkommazahl doppelte Genauigkeit (64 Bit)
- void ⇒ nichts
- enum ⇒ Aufzählung

In der Standard Template Library (STL) ⇒ Standardbibliothek sind dann noch weitere Datentypen enthalten, wie z.B.:

- string ⇒ Zeichenketten
- complex ⇒ Komplexe Zahlen
- vector ⇒ Ähnlich wie Arrays
- valarray ⇒ Vektorrechnung
- list ⇒ Listen
- map ⇒ Assoziative Felder (Wörterbuch)

Wahrheitswerte - bool

Der **Datentyp für Wahrheitswerte heißt in C++ bool**, was eine **Abkürzung für boolean** ist. Er kann nur **zwei Zustände** annehmen: **true (wahr)** oder **false (falsch)**. Obwohl eigentlich 1 Bit ausreichen würde, hat bool mindestens eine Größe von einem Byte (also 8 Bit), denn 1 Byte ist die kleinste adressierbare Einheit und somit die Minimalgröße für jeden Datentyp.

```
bool betrunken=false;
```

Zeichen - char

Zeichen sind eigentlich Ganzzahlen. Sie unterscheiden sich von diesen nur bezüglich der Ein- und Ausgabe. **Jeder Zahl ist ein Zeichen zugeordnet**. Mit den Zahlen lässt sich ganz normal rechnen aber bei der Ausgabe erscheint das zugeordnete Zeichen auf dem Bildschirm. Welches Zeichen welcher Zahl entspricht, wird durch den verwendeten Zeichensatz festgelegt.

Die meisten Zeichensätze beinhalten den **sogenannten ASCII-Code (American Standard Code for Information Interchange)**, welcher die Zeichen 0 – 127 belegt. Er enthält **32 Steuerzeichen (0 – 31) und 96 druckbare Zeichen (32 – 127)**.

char ist der Standard-Datentyp für Zeichen. Er ist in der Regel **1 Byte** groß und kann somit **256 verschiedene Zeichen** darstellen. Diese genügen für einen erweiterten ASCII-Code, welcher zum

Beispiel auch deutsche Umlaute definiert. Für Unicode-Zeichen gibt es die Datentypen char16_t mit einer Größe von 2 Byte und char32_t mit einer Größe von 4 Byte. Früher nutzte man für Unicode auch den Datentyp wchar_t, welcher je nach System 2 oder 4 Byte groß war. Dieser Datentyp sollte jedoch nicht mehr eingesetzt werden.

Ganzzahlen - int

Dieser Datentyp umfasst sowohl **ganzzahlige positive als auch negative Zahlen** aus dem Wertebereich von -2.147.483.648 bis 2.147.483.647. Es werden also vom Compiler für eine Variable des Datentyps int **32 Bit (4 Byte)** reserviert.

```
int x; // Variablendeklaration; die Variable x soll vom Typ int sein
x = 3; //Wertzuweisung, Initialisierung; die Variable x enthält den Wert 3
```

Fließkommazahlen

In der realen Welt sind **ganzzahlige Werte oft nicht ausreichend**.

Bei Gewichten, Geschwindigkeiten und anderen Werten aus der Physik ist immer mit Nachkommastellen zu rechnen. Dieser Tatsache kann sich auch eine Computersprache nicht entziehen. Eine solche Fließkommazahl besitzt zwei Komponenten.

Die eine ist die **Mantisse**, und die andere ist der **Exponent zur Basis 10**.

Der Exponent wird durch ein kleines oder großes E abgetrennt. Die Zahl **1.5E2** hat den Gegenwert von **150**. Dabei ist **1.5** die **Mantisse**, **2** ist der **Exponent**.

float

Der **einfachste Typ mit Nachkommastellen** heißt **float**. Die Zahlen dieses Typs sind binär kodiert, damit sie effizient im Computer verarbeitet werden können. Die Zahl lässt **ganzzahlige, aber auch negative Exponenten** zu. Dadurch sind nicht nur extrem große Zahlen darstellbar, sondern auch sehr kleine Brüche. Die Speicheranforderung einer float-Variablen ist nicht sehr hoch, typischerweise liegt sie bei **4 Bytes**.

```
float pi=3.14159265;
```

double

Reicht die Genauigkeit nicht aus oder werden Größenordnungen benötigt, die über die Kapazität einer float-Variablen hinausgehen, steht der Typ double zur Verfügung. Der Name bedeutet »doppelt« und bezieht sich auf die **Genauigkeit**. Diese ist **doppelt so hoch wie bei float**.

```
double pi=3.14159265;
```

Nichts - void

Der Datentyp **void** bedeutet **nichts**, ist **kein echter Datentyp** und wird überall dort **verwendet, wo kein Wert benötigt wird** oder vorhanden ist. Bei **Funktionen** wird void verwendet, wenn eine Funktion keinen Wert zurückgibt oder die Funktion keinen Parameter hat.

Aufzählungen - enum

Mit **enum können Aufzählungstypen** definiert werden. Dazu gehören **Wochentage** oder **Farben**. Den **Elementen** eines Aufzählungstyps werden **Namen zugeordnet**, obwohl sie **intern natürlich als Zahlen codiert** werden

```
enum Farbe {ROT, GELB, GRUEN, BLAU};
```

Zeichenketten - string

Die C++-Standardbibliothek enthält eine Klasse namens **string**. Um diese Klasse nutzen zu können, müssen Sie die gleichnamige Headerdatei string einbinden.

```
#include <iostream>  //Bibliothek für Ein- und Ausgabe-Befehle
#include <string>  // Bibliothek für Zeichenketten

using namespace std;

int main() {
    string zeichenkette;
    zeichenkette = "Hallo Welt!";
    cout << zeichenkette << endl;
}
```

6.8.4) Verarbeitung

Zuweisung

Gleichheitszeichen

Bei der **Initialisierung** wurde **bereits das Gleichheitszeichen verwendet**, um eine **Variable mit einem Wert vorzubelegen**. Das Gleichheitszeichen wird auch ansonsten verwendet, wenn eine Variable einen **neuen Wert bekommen soll**. Dabei gibt es einen feinen Unterschied zwischen der Vorbelegung oder **Initalisierung**, die **beim Anlegen der Variablen** erfolgt, und der **Zuweisung**, wie die **Belegung** mit Werten genannt wird, **wenn die Variable bereits existiert**.

Links vom Gleichheitszeichen steht immer das Ziel der Zuweisung. Im Allgemeinen ist das eine Variable. Auf der **rechten Seite des Gleichheitszeichens steht die Datenquelle**. Das kann eine andere Variable, ein Zahlenwert oder eine Berechnung sein. Man bezeichnet die **Datenquelle auf der rechten Seite** einer Zuweisung **allgemein als Ausdruck**. Die englische Bezeichnung dafür ist **»expression«**. Ein Ausdruck ist Konstrukt, das einen Wert liefert und so als Datenquelle dienen kann. Das folgende Beispiel zeigt mehrere Zuweisungen. Dabei wird auch schon den Rechenoperationen ein wenig vorgegriffen.

```
int main()
{

    //Initialisierungen
    int MWStSatz=0, Netto=0;
    double MWStBetrag=0.0, Brutto=0.0;

    //Zuweisungen
    MWStSatz = 16;
    Netto = 200;
    MWStBetrag = Netto * MWStSatz / 100;
    Brutto = Netto + MWStBetrag;

    return 0;
}
```

Kaskadierende Zuweisung

C++ hat die Besonderheit, dass Sie in einer Anweisung mehreren Variablen den gleichen Wert zuweisen können. Sie müssen sich das so vorstellen, dass eine Zuweisung ihren Wert nach links durchreicht. Diese Fähigkeit ermöglicht eine Zeile wie die folgende:

```
int main()
{
    int a,b,c,d;
```

```
    a = b = c = d = 5 + 2;

    return 0;
}
```

Grundrechenarten

Letztlich sieht es nicht sehr viel anders aus, als wenn Sie sich eine Rechenaufgabe auf einen Zettel schreiben. Etwas ungewohnt ist lediglich, dass auf der linken Seite das Zuweisungsziel und ein Gleichheitszeichen steht. Das **Multiplikationszeichen ist der Stern**, und das **Divisionszeichen der Schrägstrich**. **Plus- und Minuszeichen** sehen so aus, wie man es erwartet. Sie können sogar das **Minuszeichen wie gewohnt als Vorzeichen** verwenden.

Restrechnung bzw. Modulo

Eine besondere Rechenart ist die **Modulo-Rechnung**. Sie liefert den **Rest einer ganzzahligen Division**. Wenn Sie sich daran erinnern, wie Sie in den ersten Schulklassen dividiert haben, dann fallen Ihnen vielleicht noch Sätze ein wie: »25 geteilt durch 7 sind 3, Rest 4«. Diese Restberechnung gibt es auch unter C++. Man bezeichnet sie als die Modulo-Rechnung. Als **Operatorzeichen wird das Prozentzeichen** verwendet.

```
Rest = 25 % 7; // Rest ist also 4
```

Punkt vor Strich

Auch in C++ werden die Gesetze der Mathematik geachtet. So wird die alte Regel **»Punktrechnung geht vor Strichrechnung«** auch hier eingehalten. Diese Regel sagt aus, dass die Multiplikation und die Division vor einer Addition oder Subtraktion ausgeführt werden, wenn beide gleichwertig nebeneinander stehen.

Links nach rechts

Binäre Operatoren, also Rechensymbole, die zwei Ausdrücke miteinander verknüpfen, sind im Allgemeinen **linksbindend**. Das bedeutet, dass sie von links nach rechts ausgeführt werden, wenn die Priorität gleich ist. Das heißt, dass **a*b/c als (a*b)/c** ausgewertet wird. Eine **Ausnahme ist die Zuweisung**. Hier wird **a=b=c als a=(b=c)** ausgewertet. Der **Zuweisungsoperator ist also rechtsbindend**. Auch einstellige Operatoren sind rechtsbindend. Dazu gehört auch der Operator ++, den Sie im Laufe des Abschnitts noch kennen lernen werden.

Abkürzungen

Der Mensch neigt zur Bequemlichkeit. Nicht anders geht es Programmierern. Sie handeln nach dem Grundgedanken, niemals etwas zu tun, was ein Computer für sie tun kann, und so ist es naheliegend,

dass es Wege gibt, wiederkehrende Aufgaben möglichst kurz zu formulieren.

Inkrementieren

Um den Wert einer Variablen um 1 zu erhöhen, können Sie die folgende Zeile schreiben:

```
int Zaehler=0;
Zaehler=Zaehler+1;
```

Sind sie schreibfaul, so funktioniert auch folgende Variante:

```
int Zaehler=0;
Zaehler+=1;
```

Hier wird das Pluszeichen mit dem Gleichheitszeichen kombiniert. Damit das Plus nicht fälschlicherweise als Vorzeichen der 1 interpretiert wird, muss es vor dem Gleichheitszeichen stehen. Zwischen Plus- und Gleichheitszeichen darf kein Leerzeichen stehen. Die Bedeutung der Zeile ist also: **»Addiere der Variablen Zaehler den Wert 1 hinzu.«**

Das geht mit allen Rechenarten:

kurze Schreibweise	**lange Schreibweise**
a+=b	a=a+b
a-=b	a=a-b
a/=b	a=a/b
a*=b	a=a*b
a%=b	a=a%b

6.8.5) Ein- und Ausgabe

Ausgabestrom nach cout

C++ verwendet für die **Ein- und Ausgabe** das **Datenstrom-Modell**. Der Datenstrom zur Ein- und Ausgabe wird in einer **eigenen Bibliothek (iostream)** behandelt. Um sie zu verwenden, muss vor der ersten Verwendung die folgende Zeile eingegeben werden:

```
#include <iostream.h>
```

In einigen Listings werden Sie auch sehen, dass iostream ohne die Erweiterung .h verwendet wird. Wenn Sie einen halbwegs aktuellen Compiler verwenden, ist beides möglich. Der Unterschied hat mit der Verwendung von Namensräumen zu tun.

Um Daten anzuzeigen, werden sie auf die **Datenausgabe cout** (Das »out« in cout ist englisch und bedeutet »aus«). Zunächst wird das Datenziel genannt. Dann werden zwei **Kleiner-Zeichen** verwendet, um die **Richtung der Daten** in Richtung der Ausgabe anzuzeigen.

```
cout << MeinZahlenWert;
cout << endl;
```

Im Beispiel wird die **Variable MeinZahlenWert ausgegeben**. In der Zeile darunter wird **endl** auf den Datenstrom gesendet. Dies ist keine selbst definierte Variable, sondern eine **vordefinierte Konstante für das Zeilenende**. Die Verwendung von endl sorgt auch dafür, dass alle anzuzeigenden Daten sofort auf dem Bildschirm erscheinen.

Kaskadierung

Der Übersicht halber oder um sich Tipparbeit zu sparen, können mehrere Ausgabeobjekte direkt hintereinander, durch die doppelten Kleiner-Zeichen getrennt, aufgeführt werden.

```
cout << MeinZahlenWert << endl;
```

Textausgabe

Sie können nicht nur Variablen, sondern auch **Konstanten (Text)** auf diesem Weg ausgeben. Das ist besonders interessant bei Zeichenketten, mit denen Sie Ihren Programmausgaben ein paar **erläuternde Texte beifügen** können.

```
cout << "Ergebnis: " << MeinZahlenWert << endl;
```

Im Sinne der Benutzerfreundlichkeit eines Programms, sagt der Programmierer dem Benutzer, dass der Wert, der jetzt auf dem Bildschirm erscheint, das Ergebnis des Programms ist.

Eingabestrom aus cin

Um **Eingaben von der Tastatur zu lesen**, wird die Datenquelle **cin** (Das »in« in cin steht für »ein«. Es ist also das Eingabe-Objekt. Der Eingabeoperator besteht genau **spiegelverkehrt zum Ausgabeoperator** aus **zwei Größer-Zeichen**. Sie weisen quasi vom cin auf die Variable, in der die Eingabe abgelegt werden soll.

```
#include <iostream.h>

int main(void)
{
    int Zahleingabe;
    int Doppel;

    cout << "Bitte geben Sie eine Zahl ein!" << endl;
    cin >> Zahleingabe;
    Doppel = Zahleingabe * 2;
    cout << "Das Doppelte dieser Zahl ist "
        << Doppel << "." << endl;

    return 0;
}
```

6.8.6) Ablaufsteuerung

Bisher bestanden die Programme nur aus aneinander gehängten Anweisungen. Jedoch ist es oft notwendig, den Ablauf eines Programms aufgrund einer Bedingung bzw. von Abhängigkeit von Zuständen zu verändern. Hier ein Beispielfür ein besseres Verständnis zum Thema Kaffeekochen.

Ablaufsteuerung - Beispiel "Kaffee kochen"

Folgende Anweisungen werden beim Kaffeekochen nacheinander ausgeführt:

```
Filtertüte in den Filter stecken
6 Löffel Kaffeepulver in die Filtertüte einfüllen
1 Liter Wasser in den Wasserbehälter einfüllen
Maschine einschalten
Warten
Maschine ausschalten
Kaffee entnehmen
```

Das ist allerdings eine sehr unvollständige Beschreibung des Ablaufs. In der Praxis werden Abläufe in Abhängigkeit von Zuständen erfolgen und Vorgänge wiederholt. Eine detailliertere Beschreibung des Kaffeekochens sieht so aus:

```
Wenn noch eine benutzte Filtertüte in der Maschine steckt
{
    Entnimm die Filtertüte
    Wirf sie in den Müll
}

Neue Filtertüte aus der Packung nehmen
Filtertüte in den Filter stecken

Wiederhole 6-mal:
{
    fülle einen Löffel Kaffeepulver in die Filtertüte
}

1 Liter Wasser in den Wasserbehälter einfüllen
Maschine einschalten

Wiederhole:
{
    Warte eine Minute
} bis kein Kaffee mehr aus dem Filter tropft

Maschine ausschalten
Kaffee entnehmen
```

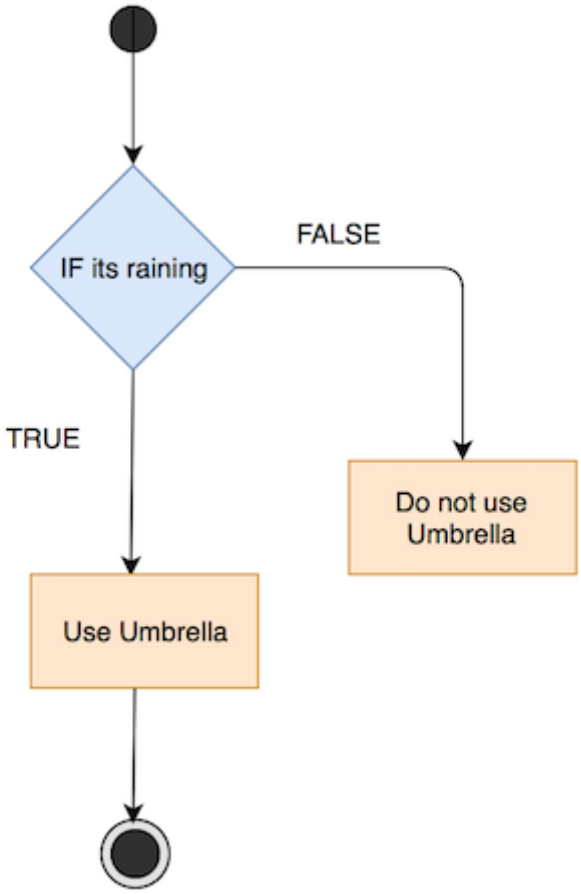
- Verzweigungen
- Schleifen

Verzweigungen

Es gibt diverse Gründe, warum ein Programm in Abhängigkeit von Variablen bestimmte Operationen nicht ausführen soll. Hier sind ein paar typische Beispiele zusammengestellt:

- Ganz offensichtlich darf ein Programm nicht dividieren, wenn der Nenner 0 ist.
- Vor dem Ziehen einer Wurzel ist es sinnvoll zu prüfen, ob der Operand negativ ist.
- Das Programm darf nicht mehr auf Aliens schießen, wenn die letzte Rakete des eigenen Raumschiffs verschossen ist.

Das Programm soll in all diesen **Fällen** in **Abhängigkeit von einem Variableninhalt** seinen **Ablauf ändern**. Dazu brauchen wir einen **Befehl, der unter einer Bedingung eine Anweisung oder einen Anweisungsblock** ausführt. Um die Bedingung zu formulieren brauchen wir **Operatoren**, mit denen wir **Werte vergleichen** können.



Verzweigung - IF (Wenn-Dann)

Folgende Sätze wären Bedingungen, die man

```
Wenn eine Frau ihren Mann aus dem Fenster wirft, liest man das in der
Kronen-Zeitung.
Wenn ein Mann seine Frau aus dem Fenster wirft, liest man das in Schöner
Wohnen.
```

Zitat unbekannter Herkunft.

Mit dem Befehl **»if«** ist es möglich, eine **Befehlssequenz unter einer Bedingung** auszuführen. Das Schlüsselwort if kommt aus dem Englischen und bedeutet **»falls«** oder **»wenn«**. Der Syntax einer einfach if-Verzweigung sieht folgendermaßen aus:

```
if( «Bedingung» ){
    «Anweisungen»;
}
```

Dem **Schlüsselwort if** folgt eine Klammer, in der die Bedingung formuliert wird, unter der der nachfolgende **Anweisungsblock** ausgeführt wird. Ein **Anweisungsblock** ist entweder **eine einzelne Anweisung oder mehrere Anweisungen**, die durch **geschweifte Klammern zusammengefasst** werden.

```
int Gemeindeeinwohner=12000;

if (Gemeindeeinwohner > 10000)
{
    cout << "Diese Gemeinde ist eine Gemeinde mit Stadtrecht!" << endl;
}
```

Einrücken

Es hat sich bewährt, **Anweisungen**, die unter Bedingungen oder in Schleifen ausgeführt werden, **einzurücken**. Einige Programmeditoren führen dies automatisch durch. Für das Einrücken können die **Tabulatortaste oder Leerzeichen** verwendet werden.

```
if (Divisor != 0)
{
    Ergebnis = Dividend / Divisor;
}
```

Die Division wird nur durchgeführt, wenn die Bedingung zutrifft, also der Divisor ungleich 0 ist.

Verschachtelte IF-Verzweigungen

Die Anweisung, die hinter der if-Bedingung steht, kann selbst wieder eine if-Anweisung sein. Sie wird nur ausgeführt, wenn die erste if-Bedingung eintritt. Eine solche Folge von Abfragen nennt man Verschachtelung.

```
int a=0, b=0;
int c=2;
if (a==0)
    if (b==0)
        c=0;
cout << c << endl;
```

Die Abfrage, ob die Variable b 0 enthält, wird nur gestellt, wenn die Variable a den Inhalt 0 hat. Nur wenn beide Variablen 0 sind, wird die Variable c auf 0 gesetzt. Ansonsten enthält sie weiterhin 2.

Andernfalls: ELSE (Wenn-Dann-Sonst)

Kehren wir zum Divisionsbeispiel zurück. Es wäre schön, wenn das Programm nicht nur prüft, ob der Divisor ungleich 0 ist, sondern den Anwender darüber informieren würde, dass es die Division gar nicht durchführt. Diese Meldung sollte genau dann erscheinen, wenn die if-Bedingung nicht zutrifft. Das könnten Sie erreichen, indem Sie zwei gegensätzliche if-Konstruktionen hintereinander setzen:

```
if (Divisor != 0)
    Ergebnis = Dividend / Divisor;
if (Divisor == 0)
    cout << "Divisor ist 0! Keine Berechnung!";
```

Da solche Konstruktionen immer wieder gebraucht werden, bietet C++ einen eigenen Befehl an, um den gegenteiligen Fall der Bedingung abzudecken. Das Schlüsselwort dazu heißt **else**.

```
if (Divisor != 0)
    Ergebnis = Dividend / Divisor;
else
    cout << "Divisor ist 0! Keine Berechnung!";
```

Mehrfachverzweigung: switch case

Die Abfrage mit if ermöglicht die Unterscheidung zweier Fälle. Wenn auf Grund verschiedener Werte unterschiedliche Anweisungen ausgeführt werden sollen, können **mehrere verschachtelte if-Anweisungen** hintereinander gesetzt werden oder es kann eine **spezielle Fallunterscheidung** verwendet werden. Ein schönes Beispiel ist ein Fahrstuhl in einem Kaufhaus, in dem eine Vielzahl von Stockwerken zur Auswahl stehen und in jedem Stockwerk andere Waren angeboten werden. Zunächst wird die Aufgabe durch **kaskadierende if-Anweisungen** gelöst:

```
if (Stockwerk == 1)
{
     cout << "Süssigkeiten, Bücher" << endl;
}
else if (Stockwerk == 2)
{
    cout << "Bekleidung" << endl;
}
else if (Stockwerk == 3)
{
    cout << "Bekleidung" << endl;
}
else if (Stockwerk == 4)
{
    cout << "Spielzeug" << endl;
}
```

```
else if (Stockwerk == 5)
{
    cout << "Unterhaltungselektronik" << endl;
}
else
{
    cout << "Garage" << endl;
}
```

Diese Konstruktion lässt sich durch die Mehrfachverzweigung vereinfachen.

```
int Stockwerk;

cin >> Stockwerk;

switch (Stockwerk)
{
    case 1:
        cout << "Süssigkeiten, Bücher" << endl;
        break;
    case 2:
    case 3:
        cout << "Bekleidung" << endl;
        break;
    case 4:
        cout << "Spielzeug" << endl;
        break;
    case 5:
        cout << "Unterhaltungselektronik" << endl;
        break;
    default:
        cout << "Garage" << endl;
        break;
}
```

Die Fallunterscheidung beginnt mit dem **Schlüsselwort switch**. In der darauf folgenden Klammer steht der **ganzzahlige Ausdruck**, dessen **Ergebnis die Verzweigung steuert**. Ein Ausdruck kann eine Variable sein, wie im Beispiel die Variable Stockwerk. Hier könnte aber auch eine Berechnung stehen, die zu einem ganzzahligen Ergebnis führt. Es können **auch Buchstaben** verwendet werden, da Buchstaben aus Sicht von C++ letztlich nichts anderes als getarnte Zahlen sind. Im Beispiel wird die **Variable Stockwerk ausgewertet**, die offensichtlich eine ganze Zahl aufnehmen kann.

Schleifen

Es gibt viele Aufgaben, in denen Befehle wiederholt werden sollen, bis ein bestimmtes Ergebnis erzielt worden ist. Eine solche Wiederholung nennt man in Programmen eine Schleife. Jede Form des Zählens wird im Programm in einer Schleife realisiert. In Computerspielen dürfen Sie so lange ballern, bis Ihr letztes Raumschiff abgeschossen wurde. Eine Schleife hat den Wiederholungscharakter, enthält aber auch immer eine Bedingung, unter der die Schleife verlassen wird, oder positiv formuliert: Die Schleife wird so lange durchlaufen, wie die Schleifenbedingung gilt.

Kopfgesteuerte Schleife : for-loop

Die beiden bisher gezeigten Schleifen benötigten drei Elemente:

- Eine Initialisierung vor Beginn der Schleife
- Eine Bedingung, unter der die Schleife durchlaufen wird
- Eine Aktion, die die Bedingung verändern kann

Fehlt eines dieser Elemente oder ist es nicht korrekt, dann ist das Risiko einer Endlosschleife groß. Die for-Schleife enthält alle drei Elemente in ihrem Kopf. So sind sie an einer Stelle zusammengefasst und können leichter unter Kontrolle gehalten werden.

Eine sehr typische Anwendung für Schleifen ist das Zählen; und gerade hier zeigt sich besonders die Übersichtlichkeit der for-Schleife. Das folgende Beispiel zählt von 0 bis 9.

```
for (i=0; i<10; i++)
{
    cout << i << endl;
}
```

Zusammengefasst

In der Klammer, die dem Schlüsselwort for folgt, findet sich als Erstes eine Startanweisung, die die Variable i auf 0 setzt. Bei den while-Schleifen wurde diese Initialisierung vor der Schleife durchgeführt. Als Zweites ist die Schleifenbedingung zu erkennen. Die dritte Anweisung ist das Inkrementieren der Zählervariablen, die üblicherweise am Ende der Schleife steht. Diese drei Elemente werden je durch ein Semikolon getrennt.

Die for-Schleife ist der while-Schleife sehr ähnlich. Sie verpackt ihre Fähigkeiten nur eleganter und bewirkt, dass der Programmierer nicht so leicht wichtige Elemente vergisst. Sie könnten das obige Schema leicht auch als while-Schleife darstellen.

```
Startanweisung;
while ( Bedingung)
{
    Schleifenkörper;
    Schlussanweisung;
}
```

Entsprechend unterscheidet sich das Struktogramm nicht von dem der while-Schleife. Es wird lediglich inhaltlich der Kopf der Schleife in den Abfragebereich geschrieben.

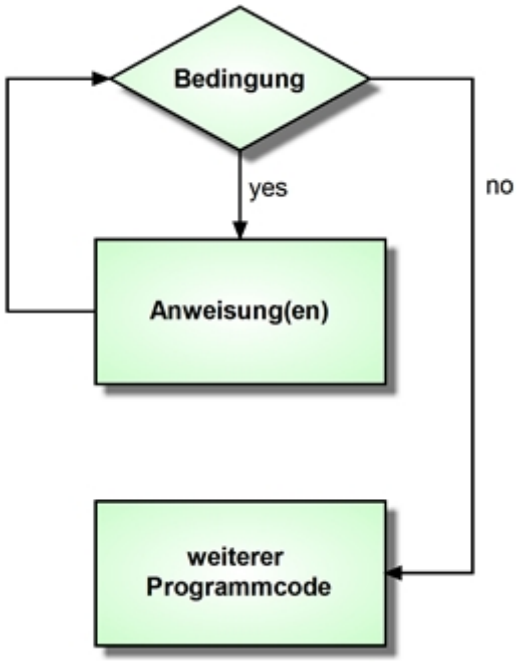
Kopfgesteuerte Schleife : while-loop

Solange

Die **einfachste Schleife** wird durch das **Schlüsselwort while** eingeleitet. Die deutsche Übersetzung lautet **»solange«**.

Sie **wiederholt Vorgänge, solange eine bestimmte Bedingung erfüllt ist**.

Auch im Alltag verwenden wir solche Schleifen. Solange die Suppe noch nicht salzig genug ist, schüttet man noch einen Teelöffel Salz hinein. Und wie in diesem Beispiel zuerst der Salzgehalt geprüft wird, so prüft auch die while-Schleife zuerst die Bedingung und führt dann erst die Anweisung aus.



Analogie zu if

Der Syntax ähnelt sehr einer if-Anweisung. Und tatsächlich bestehen die Unterschiede nur darin, dass die **Schleife die Anweisung wiederholt**. Ist die **Bedingung von vornherein nicht erfüllt**, verhält sich die Schleife exakt wie die if-Anweisung - **Sie führt die Anweisung nicht aus**.

Die Schleife beginnt mit dem **Schlüsselwort while**. In Klammern folgt die Bedingung, unter der die direkt anschließende Anweisung ausgeführt wird. Auch hier können mehrere Anweisungen zu einem Anweisungsblock zusammengefasst werden, wenn sie in geschweiften Klammern stehen.

Zählen für Anfänger

Als erstes Beispiel bringen wir dem Computer das Zählen bei. Denken Sie darüber nach, was Sie tun, wenn Sie zählen. Zunächst beginnen Sie bei einer bestimmten Zahl, typischerweise bei eins. Dann folgt die Wiederholung, in der Sie immer den bisherigen Wert um eins erhöhen. Da Sie auch irgendwann einmal etwas anderes tun wollen als zählen, werden Sie bei einem bestimmten Wert die Wiederholung abbrechen.

Zählen für Computer

Im Programm verwenden Sie eine Variable, die Sie mit dem Startwert vorbesetzen. Dieser Startwert muss vor der Schleife gesetzt werden, sonst würde die Variable in jeder Runde auf ihren Ausgangspunkt zurück gesetzt, und die Schleife würde ewig rotieren. Danach soll das Programm in die Schleife eintreten, in der die Variable immer um eins erhöht wird. Die Schleife beginnt mit dem Schlüsselwort while. Direkt dahinter wird in Klammern die Bedingung formuliert. Solange diese Bedingung gilt, bleibt das Programm innerhalb der Schleife. Wenn Sie bis zehn zählen wollen, müssen Sie hier prüfen, ob die Variable kleiner oder gleich 10 ist. Direkt anschließend an die Bedingung können Sie eine Anweisung angeben, die in der Schleife wiederholt werden soll. Da Sie ja einerseits die Variable um eins erhöhen wollen und andererseits den aktuellen Stand auf dem Bildschirm ausgeben wollen, brauchen Sie aber zwei Anweisungen, die in der Schleife ausgeführt werden sollen. Das kann aber leicht mit einem Block, also geschweiften Klammern gelöst werden. In den Block wird also die Ausgabe der Variablen und die anschließende Erhöhung der Variablen gesetzt.

```
#include <iostream>
using namespace std;
int main()
{
    i = 1;
    while (i <= 10) // Schleifenbedingung
    {
        cout << i << endl;  // Aktion
        i++; // Ohne diese Erhöhung wird es eine Endlosschleife
    }

    return 0;
}
```

Grenzfragen

Leicht entstehen Fehler bei der Festlegung der Schleifengrenzen. Würden Sie statt ⇐ nur das Kleinerzeichen verwenden, würde das Programm nur bis neun zählen. Der Grund ist, dass sobald der Inhalt von i den Wert 10 erreicht, die Bedingung nicht mehr gilt und damit das Schleifeninnere nicht mehr betreten wird. Sie können auch eine Schleife bilden, die zehnmal durchläuft, indem Sie bei 0 beginnen und abfragen, ob die Variable kleiner als zehn ist. Dann würde die Variable i von 0 bis 9 laufen. Wenn Sie i allerdings vor der Ausgabe erhöhen, erhalten Sie wieder die Zahlen von 1 bis 10.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int i = 0;
    while (i < 10)
    {
        i++;
        cout << i << endl;
    }
    return 0;
}
```

Test am Kopf

Die **while-Schleife** prüft die Bedingung am Schleifenkopf. Das bedeutet, dass die Anweisungen in der Schleife gar nicht ausgeführt werden, wenn die Bedingung vor dem Betreten der Schleife unwahr ist.

Endlosschleife

Achten Sie besonders auf die Formulierung der Schleifenbedingung. Nicht nur Anfängern passiert es, dass die Schleife niemals endet, weil die Schleifenbedingung niemals unwahr wird. Eine solche Schleife wird Endlosschleife genannt. Eine andere Ursache für eine Endlosschleife kann darin bestehen, dass die Bedingung zwar richtig formuliert ist, aber dass der Programmierer vergisst, die Daten, die in der Bedingung abgefragt werden, innerhalb der Schleife zu ändern.

```
while (true)
{

}
```

Fußgesteuerte Schleife : do....while-loop

In manchen Situationen ist die Prüfung am Kopf der Schleife ungünstig. Manchmal wollen Sie eine Aktion durchführen und werden erst an ihrem Ende erkennen können, ob die Aktion wiederholt werden soll. Ein Kind wird beispielsweise seinen Opa so lange um Süßigkeiten anbetteln, bis dieser keine mehr hat. (Erfahrene Eltern werden zu Recht einwerfen, dass die Abschlussbedingung von dem System Opa nicht eingehalten wird.) Das Kind wird in jedem Fall erst einmal betteln, ansonsten kann es die Bedingung gar nicht erst prüfen. In diesem Fall findet also die Prüfung erst nach dem Durchlauf des Schleifenkörpers statt.

Eingabeprüfung

In Programmen wird eine fußgesteuerte Schleife besonders bei Eingabeprüfungen eine typische Anwendung. Innerhalb der Schleife fordert das Programm vom Anwender eine Eingabe an. Erst dann wird geprüft, ob die Eingabe korrekt ist. Ist die Eingabe nicht zufriedenstellend, wird die Schleife wiederholt. Ein weiteres Beispiel ist das Spiel Minesweeper. Der Benutzer soll auf ein Feld klicken. Solange unter dem Feld keine Mine liegt, darf er weiterspielen. Sie können erst am Ende der Schleife

feststellen, wohin der Benutzer geklickt hat.

Auch hier werden Sie in den meisten Fällen einen Block aus geschweiften Klammern verwenden, da eine Schleife meist mehrere Anweisungen umfasst.

Im folgenden Beispiel wird rückwärts gezählt. Betrachten Sie das Listing, und überlegen Sie, ob die Zahlen 10, 1 und 0 ausgegeben werden! Prüfen Sie, ob Ihre Vermutungen stimmen!

```
#include <iostream>
using namespace std;

int main()
{
    int i = 10;
    do
    {
        cout << i << endl;
        i--;
    }
    while(i>0);

    return 0;
}
```