Wissenschaftliches Rechnen I

Vorlesung im WS 07/08

von

Prof. Dr. Nicolas Gauger

Humboldt Universität zu Berlin

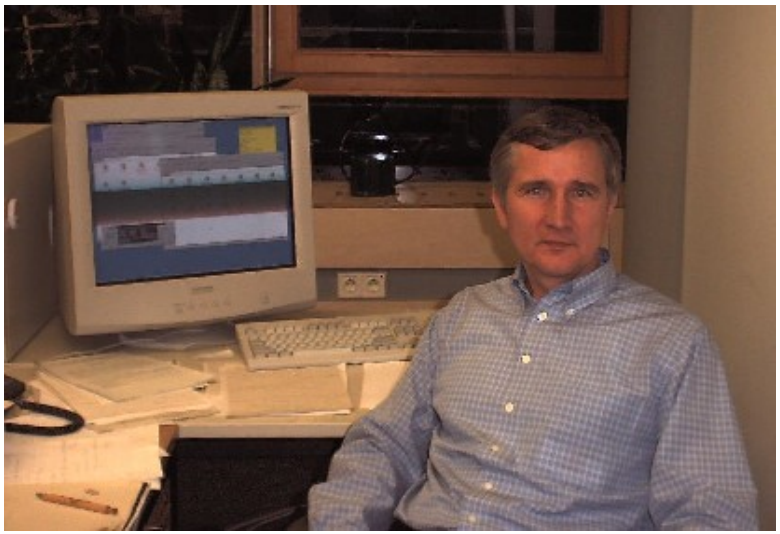
Institut für Mathematik

Zimmer 2.424

Email: nicolas.gauger@dlr.de

Tel.: 030 2093-5833 und 0531 295-3339

Das Core Team zum WR I



Dr. René Lamour
Übung und Praktikum

Dipl.-Math. Hella Rabus
Praktikum

Prof. Nicolas Gauger
Vorlesung und Übung

→ **Erklärung der Modalitäten zum Erwerb eines Scheines zum WR I**

→ **Literaturempfehlung: Java 2, Grundlagen und Einführung, 3. Auflage, RRZN, 2004.**

Alle erdenklichen Infos zum WR I

auf der Homepage:

http://www.math.hu-berlin.de/~rabus/WR_Numerik/WRI/index.shtml

Praktikum Wissenschaftliches Rechnen I

Aktuelles:

- **Übungen am 19.10.2007 in Raum 2.207**

  Die Übungen zur Einführung in JAVA finden im Computerpool 2.207 statt.

- **Aktuelle Raumverteilung**

  Alle Praktikas am Dienstag finden ab sofort in Raum 2.212 statt.

  Insbesondere zur Einführung von JAVA werden auch in die Übungen im Pool stattfinden. Auf dieser Seite finden Sie einen Hinweis darauf, wo die aktuelle Übung stattfinden wird.

Vorlesung		Praktika		Übungen	
Fr 09-11 Uhr	RUD 25, 1.013	Di 09-11 Uhr	RUD 25, 2.212	Fr 11-13 Uhr	RUD 26, 1.304 / RUD 25, 2.207
		Di 11-13 Uhr	RUD 25, 2.212	Fr 13-15 Uhr	RUD 25, 3.006 / RUD 25, 2.207
		Di 13-15 Uhr	RUD 25, 2.212	Fr 15-17 Uhr	RUD 25, 3.006 / RUD 25, 2.207
		Di 15-17 Uhr	RUD 25, 2.212	Fr 17-19 Uhr	RUD 25, 3.006 / RUD 25, 2.207

- **Beginn des Praktikums und der Übungen**

  In der ersten Vorlesungswoche finden bereits Übung und Praktika statt. Am Dienstag, den 16. Oktober beginnt die Veranstaltung mit einem Praktikum im Computerpool.

- **Einschreibung zur Übung und zum Praktikum *Wissenschaftliches Rechnen I***

  Für die Übungen und Praktika zur Vorlesung wissenschaftliches Rechnen I können Sie sich bereits vor Semesteranfang anmelden. Insgesamt stehen zwar ausreichend Plätze zur Verfügung, jedoch ist die Anzahl der Arbeitsplätze im Computerpool und die Sitzplätze in den Übungsgruppen begrenzt.

  Es stehen je Termin 20 Plätze zur Verfügung

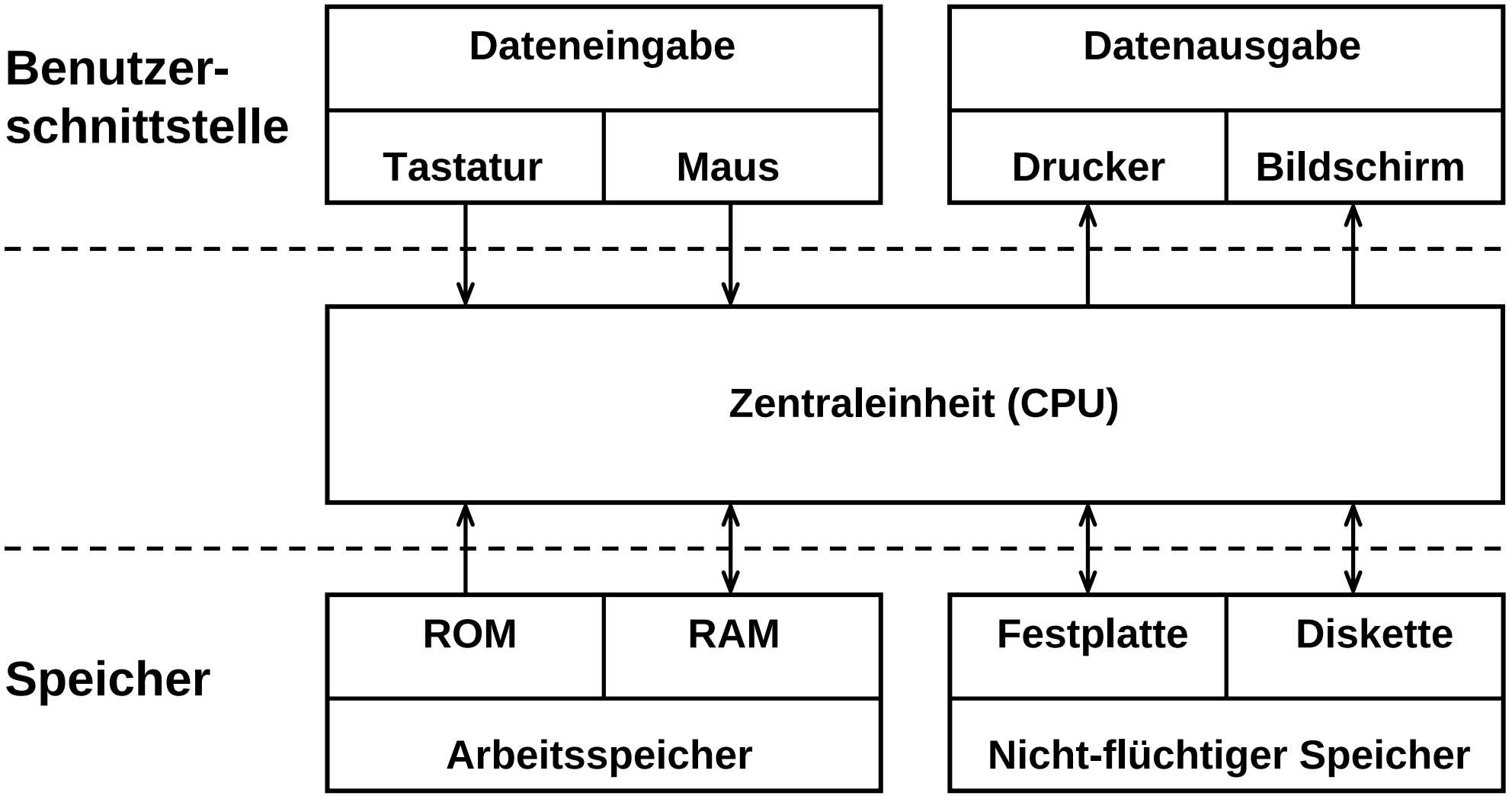
P-WRI
Aktuelles und Termine
Aufgaben
Abgabemodalitäten
Teilnehmerliste
Ansprechpartner
Skripte und Literatur
Programmieren mit Java
Java für Anfänger
Java mit Netbeans
Java von Hand
Bibliotheken für Java
Demo-Applets
Weitere Informationen
Zugriffsrechte
Latex, Gnuplot und Software für Zuhause
Aktuelle Infos und Links

Inhalte der Vorlesung und Übung zum WR I

- Aufbau und Arbeitsweise eines Computers
- Was ist eine Programmiersprache?
- Zahlendarstellung im Computer
- Grundlagen Linux, Java, Latex, Mathematica, …
- Algorithmen und ihre Darstellung
- Komplexität von Algorithmen
- Effiziente Algorithmen
- Datenstrukturen
- Suchen und Sortieren
- Graphenalgorithmen
- Kondition von Aufgaben und Fehlerfortpflanzung
- …

1. Aufbau und Arbeitsweise eines Computers

1. Aufbau und Arbeitsweise eines Computers



1. Aufbau und Arbeitsweise eines Computers

Begriffe und Erklärungen

- **Computer:** Rechner
- **Datenverarbeitung:** Egal, was ein Computer tut, er verarbeitet immer Daten
- **Programm:** Datenverarbeitungsregeln
- **Dateneingabe:** Ermöglicht Eingabe von Daten und Datenverarbeitungsregeln
- **Dateneingabegeräte:** Tastatur, Maus, …
- **Datenausgabe:** Ermöglicht Weiterverwendung der vom Computer verarbeiteten Daten
- **Datenausgabegeräte**: Bildschirm, Drucker, Soundkarte, …

1. Aufbau und Arbeitsweise eines Computers

Begriffe und Erklärungen

- **Speicher:** Hier „merkt“ sich der Computer die zu verarbeitenden Daten
- Man unterscheidet in **flüchtigen** und **nicht-flüchtigen Speicher**, d.h. Daten gehen beim Ausschalten des Computers verloren bzw. nicht verloren
- **RAM:** Random Access Memory, Teil des so genannten **Arbeitsspeichers**, flüchtiger Speicher, in der Regel werden alle vom Computer zu verarbeitenden Daten und Programme vor der Verarbeitung hierher übertragen
- **CPU:** Central Processing Unit = **Zentraleinheit**, wird auch **Prozessor** genannt, arbeitet im Arbeitsspeicher enthaltene Programme ab

1. Aufbau und Arbeitsweise eines Computers

Begriffe und Erklärungen

- **Nicht-flüchtiger Speicher:** Festplatte, Diskette, CD-ROM, …
- Beim Start des Computers durch Einschalten muss der Computer alle angeschlossen Geräte erkennen und in Gang setzen, wie er das zu tun hat ist im **ROM** beschrieben
- **ROM:** Read-only Memory, nicht-beschreibbarer Teil des Arbeitsspeichers, nicht-flüchtiger Speicher
- Außerdem muss der Computer gleich nach dem Anschalten das **Betriebssystem** starten, auch **OS** (Operating System) oder **DOS** (Disk Operating System) genannt
- **Betriebssystem:** Ein sehr umfangreiches Programm, welches den Computer durch Menschen nutzbar macht, ermöglicht z.B. die Kommunikation Benutzer-Computer

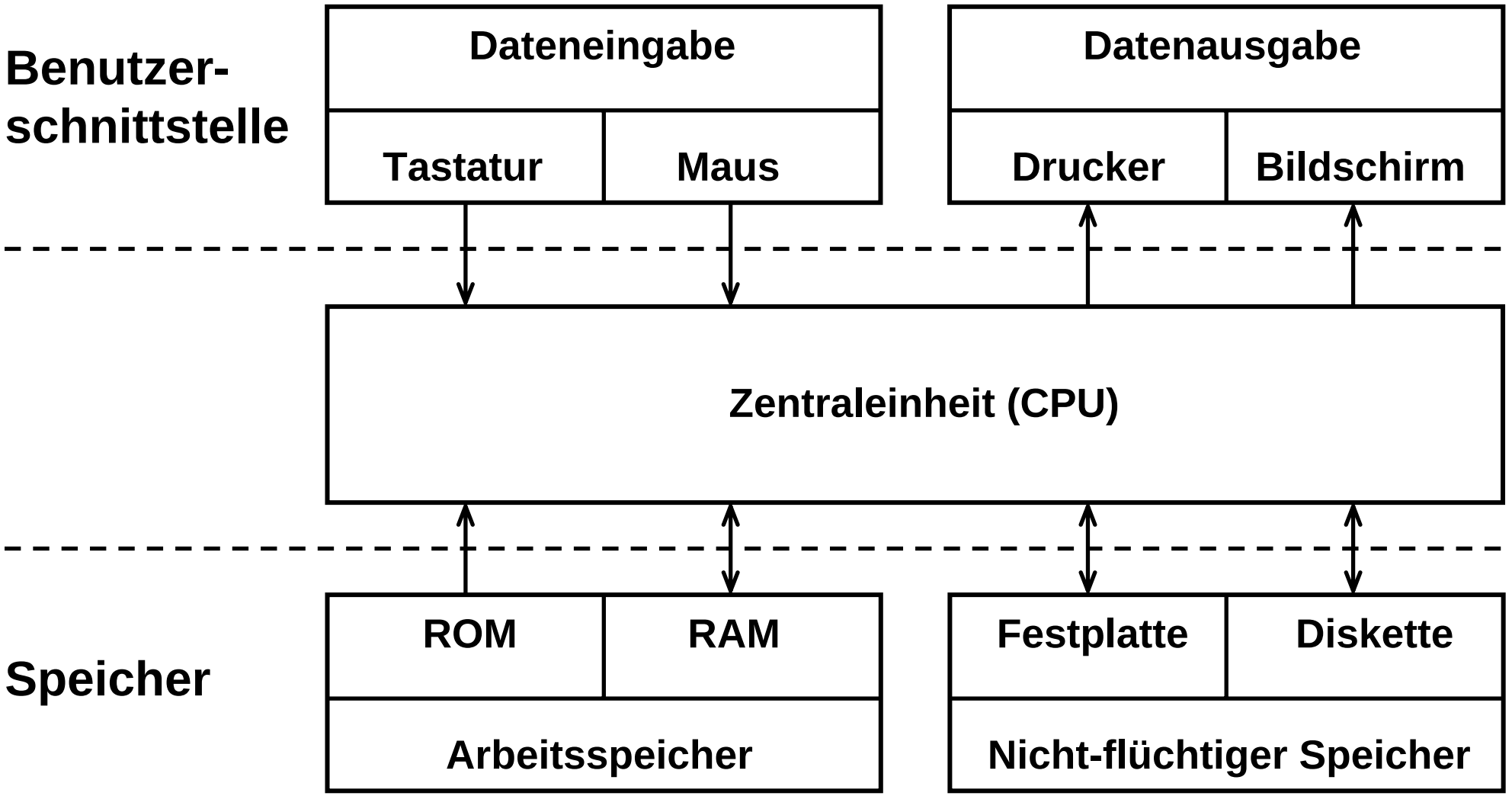
1. Aufbau und Arbeitsweise eines Computers

Beispiele von Betriebssystemen:

- **MS DOS**
- **MS Windows**
- **Mac-OS**
- **Unix-Betriebssysteme, wie z.B. Linux oder Solaris**
- **…**

Die Rechner, an denen Sie im Praktikum arbeiten, laufen z.B. unter dem Betriebssystem Linux.

1. Aufbau und Arbeitsweise eines Computers



1.1 Organisation des Arbeitsspeichers – Bits und Bytes

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Alle Daten und Programme müssen vor ihrer Verarbeitung im RAM abgelegt werden.

Auf der physikalischen Ebene ist das RAM aus unzähligen elektrischen Kondensatoren und Transistoren aufgebaut.

Ein Kondensator kann dabei in einem von zwei Zuständen sein: entweder er trägt elektrische Ladung oder er ist entladen.

Eine Speichereinheit, die nur zwei Zustände annehmen kann, nennen wir Bit, und die beiden Zustände eines Bits beziffern wir mit 0 und 1.

Zum Beispiel könnte dem durch einen ungeladenen bzw. geladenen Kondensator repräsentierten Bit der Wert 0 bzw. 1 zugeordnet werden.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Da ein Bit nur sehr kleine Informationsmengen speichern kann, werden mehrere Bits zu größeren Einheiten gruppiert.

Beispiele:

Sicher haben Sie auch schon Werbeslogans gehört, in welchen für 32-Bit-Betriebssysteme oder 32-Bit-Prozessoren geworben wird.

Ein 32-Bit-Prozessor ist in der Lage immer 32 Bit auf einmal aus dem Speicher zu lesen und als Einheit zu verarbeiten.

Ein älterer 16-Bit Prozessor hingegen verarbeitet Daten immer in 16-Bit-Portionen.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Da ein 16-Bit-Prozessor nur 16 Bit auf einmal verarbeiten kann, müsste er zweimal auf den Speicher zugreifen, um 32 Bit zu lesen, und er müsste zwei Operationen ausführen, um 32 Bit zu verarbeiten.

Die Anzahl von Bits, die ein Prozessor auf einmal verarbeiten kann, nennt man die Bus-Breite des Prozessors.

Ein Bus ist ein Bündel von Leitungen, über das Daten transportiert werden.

Wenn die zu verarbeitenden Daten in größeren Gruppen als 16 Bit vorliegen, kann ein 32-Bit-Prozessor also tatsächlich schneller sein als ein 16-Bit Prozessor.

Allerdings müssen die benutzten Programme dazu auch wirklich eine Verarbeitung in 32-Bit-Portionen vorsehen.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Bevor wir darauf eingehen, zu was für Gruppen man Bits zusammenfasst, wollen wir uns überlegen, wie viele Zustände man mit einer vorgegebenen Anzahl von Bits darstellen kann.

Beispiele:

Wenn wir ein Bit verwenden, können wir nur die zwei Zustände 0 und 1 darstellen.

Wenn wir zwei Bits verwenden, können wir die vier Zustände 00, 01, 10, 11 darstellen.

Drei Bits können die acht Zustände 000, 001, 010, 011, 100, 101, 110, 111 darstellen.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Es ist leicht zu sehen, dass n Bits genau 2^n verschiedene Zustände annehmen können.

Will man also N verschiedene Zustände beschreiben, benötigt man eine Anzahl von Bits, die sich durch Aufrunden von $\log_2 N$ ergibt.

Beispiele:

Um jede Zahl von 0 bis 15 darstellen zu können, benötigt man $\log_2 16 = 4$ Bits.

Zur Darstellung der Zahlen 0 … 100 sind 7 Bits nötig ($\log_2 101 = 6{,}66$).

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Die wichtigsten Einheiten, zu denen Bits zusammengefasst werden, sind:

Byte:

Ein Byte besteht aus 8 Bit und kann folglich $2^8 = 256$ Zustände annehmen.

Word:

Die Größe eines Word hängt ursprünglich vom verwendeten Computer ab. Es besteht in der Regel aus so vielen Bits, wie der Computer zugleich verarbeiten kann.

Heutzutage versteht man unter einem Word meist 8 Byte oder 64 Bit.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Meist verwendet man die Begriffe Byte oder Word, um einen einzelnen Zahlenwert zu beschreiben.

Jedem Zustand eines Bytes oder Words kann man eine Dezimalzahl zuweisen. Dazu fasst man die Zustände der einzelnen Bits als Binärzahl auf.

Der Dezimalwert einer Binärzahl berechnet sich wie folgt:

Die Bits der Zahl werden von rechts beginnend und mit der Zahl 0 startend durchnummeriert.

Anschließend bildet man eine Summe, in welcher man für jedes an der Position i gesetzte Bit die Zahl 2^i einsetzt.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Beispiele:

Zum Beispiel hat die Binärzahl 010 die Dezimaldarstellung $2^1 = 2$.

Die Zahl 010100100 hat die Dezimaldarstellung $2^2+2^5+2^7 = 164$.

Computer verwenden meist Byte-Werte, um Buchstaben und andere Zeichen zu beschreiben.

Dabei wird jedem Zeichen ein anderer Zahlenwert zugewiesen.

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Auf meinem Rechner sehen einige Beispiele für diese Zuordnung so aus:

Zeichen	Zahlenwert	Darstellung in Bits
a	97	01100001
b	98	01100010
c	99	01100011
A	65	01000001
#	35	00100011
+	43	00101011
1	49	00110001

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Wir haben nun die Grundeinheiten kennen gelernt, mit denen ein Computer operiert.

Diese Einheiten werden Ihnen auch bei der Programmierung mit Java immer wieder begegnen, denn auch wenn Sie in Java mit Zahlen rechnen, müssen Sie manchmal angeben, in welcher Grundeinheit die Zahlen gespeichert werden sollen.

Will man beschreiben, wie viele Daten ein Computer speichern kann, benützt man noch größere Einheiten:

1.1 Organisation des Arbeitsspeichers – Bits und Bytes

Größere Einheiten sind:

KB, Kilo-Byte: 1 KB sind 2^{10} = 1024 Byte

MB, Mega-Byte: 1 MB sind 2^{10} KB = 2^{20} = 1.048.576 Byte

GB, Giga-Byte: 1 GB sind 2^{10} MB = 2^{30} = 1.063.641.824 Byte

TB, Tera-Byte: 1 TB sind 2^{10} GB bzw. 2^{40} Byte

Heutige Rechner für den Heim- oder Bürogebrauch haben zumeist schon mehr als 128 MB Arbeitsspeicher und einige GB Festplattenspeicher.

Hochleistungsrechner haben heute mehrere GB Arbeitsspeicher und mehrere hundert GB Festplattenspeicher.

1.2 Organisation des Arbeitsspeichers – Adressen

1.2 Organisation des Arbeitsspeichers – Adressen

Sie haben soeben erfahren, dass der Arbeitsspeicher eines Rechners aus vielen Bits besteht, die zu Bytes oder größeren Gruppierungen zusammengefasst werden.

Obwohl Sie dies zur Programmierung mit Java nicht unbedingt wissen müssen, möchte ich dennoch kurz erklären, wie der Prozessor Daten im Arbeitsspeicher verwendet.

Jedes Byte des Arbeitsspeichers hat eine eindeutige Nummer – die Nummer eines Bytes nennt man seine Adresse.

Der Prozessor kann über die Adresse Daten im Arbeitsspeicher auslesen und Daten in den Arbeitsspeicher schreiben.

1.2 Organisation des Arbeitsspeichers – Adressen

Beispiel:

Wir können den Prozessor anweisen, eine Zahl in dem Byte an der Adresse 100 und eine weitere Zahl in dem an der Adresse 200 beginnenden Word zu speichern.

Anschließend können wir anweisen, die beiden Zahlen zu addieren und das Ergebnis an der Adresse 300 abzulegen.

Da hierbei ein Byte und ein Word addiert wird, sollte das Ergebnis mindestens Word-Größe haben.

2. Was ist eine Programmiersprache?

2. Was ist eine Programmiersprache?

Eine Programmiersprache ist eine formale Sprache, die zur Erstellung von Datenverarbeitungsanweisungen für Rechnersysteme verwendet wird.

Als formale Sprachen bezeichnet man Sprachen, die durch formale Grammatiken erzeugt werden können.

Als Grammatik bezeichnet man jede Form einer systematischen Sprachbeschreibung.

Formale Grammatiken sind mathematische Modelle von Grammatiken.

Die Theorie der formalen Sprachen ist ein eigenständiges Forschungsgebiet in der Theoretischen Informatik.

Im Gegensatz zur Mathematik liegen die Objekte in den formalen Sprachen immer in kodierter Form vor.

2. Was ist eine Programmiersprache?

Beispiele:

Die natürlichen Zahlen IN = {0,1,2,…} sind zunächst ein gewissermaßen gedankliches Gebilde.

In der Theoretischen Informatik wird aus ihnen durch die Kodierung in das Dezimalsystem, bestehend aus den 10 Ziffern 0 bis 9, eine formale Sprache.

Wenn wir die Wörter einer natürlichen Sprache als Alphabetzeichen ansehen, dann bilden die Sätze der natürlichen Sprache eine formale Sprache über dem Alphabet der natürlichsprachlichen Wörter.

Allerdings entzieht sich die natürliche Sprache einer vollständigen Definition, die festlegt, welche Sätze zu der natürlichen Sprache hinzugehören und welche nicht.

2. Was ist eine Programmiersprache?

Programmiersprachen dienen der Informationsverarbeitung.

Informationsverarbeitung bedeutet in der Informatik alle Arten von Datenverarbeitung.

Die äußere Form, in der sich eine Programmiersprache dem Programmierer repräsentiert, bezeichnet man als Syntax.

Der Quelltext besteht aus Wörtern und Trennzeichen, ganz ähnlich zu einer natürlichen Sprache.

Die Bedeutung eines speziellen Symbols in einer Programmiersprache nennt man dessen Semantik.

Syntax und Semantik kann man der Spezifikation, teilweise auch der Dokumentation der Programmiersprache entnehmen.

2. Was ist eine Programmiersprache?

Der Quelltext (oder auch Quellcode oder Programmcode) der Programmiersprache wird anschließend in eine Anweisungsfolge für den Computer übersetzt, der Maschinensprache.

Unter Maschinensprache versteht man ein System von Instruktionen und Daten, die der Prozessor eines Computers direkt ausführen kann.

Maschinensprache ist nur für Experten zu lesen und von diesen mittels so genannter Maschinensprachmonitore zu bearbeiten.

Das Programm wird in eine für den Computer verständliche Folge von Bits umgesetzt.

Dies kann entweder offline durch einen Compiler oder zur Laufzeit durch einen Interpreter geschehen.

2. Was ist eine Programmiersprache?

In vielen Fällen wird mittlerweile eine Kombination aus Compiler und Interpreter gewählt, bei der der Quellcode in einen abstrakten Zwischencode übersetz wird.

Der Zwischencode wird dann zur Laufzeit in einer so genannten Laufzeit-umgebung durch einen Interpreter in den eigentlichen Maschinencode überführt.

Das hat den Vorteil, dass ein und der selbe Zwischencode auf sehr vielen verschiedenen Plattformen ausführbar ist.

Beispiele für solche Zwischencodes sind der Java-Bytecode sowie die Common Intermediate Language.

2. Was ist eine Programmiersprache?

Eine Assemblersprache ist eine spezielle Programmiersprache, welche die Maschinensprache einer speziellen Prozessorarchitektur in einer für den Menschen leichter lesbaren Form repräsentiert.

Jede Computerarchitektur hat folglich ihre eigene Assemblersprache.

Eine Hochsprache oder höhere Programmiersprache ist eine für den Menschen geeignete Programmiersprache, die zwar maschinen-unabhängig aber nicht unmittelbar für den Computer verständlich ist.

Als weltweit erste höhere Programmiersprache gilt das Plankalkül von Konrad Zuse.

Andere Beispiele sind Fortran (FORmula TRANslation), ALGOL (ALGOrithmic Language), LISP (LISt Processing), C (Nachfolger von B, benannt nach den Bell Labs), C++, Java, Perl, …

2.1 Java

Java sollte ursprünglich Oak (= Eiche) heißen, nach den Bäumen, die vor dem Büro des wichtigsten Entwicklers von Java, James Gosling, standen.

Er fand jedoch heraus, dass es bereits eine Programmiersprache mit dem Namen Oak gab.

So musste sich das Team etwas anderes einfallen lassen. Eine der Kaffeesorten des Ladens, in dem das Team seinen Kaffee kaufte, hieß Java. So kam Java zu seinem Namen.

Java wurde von einem Team bei Sun Microsystems, Kalifornien, entwickelt.

2.1 Java

Java entstand aus der Notwendigkeit heraus, Software für elektronische Geräte wie VCRs, Fernseh-, Telefon- und Funkgeräte zu entwickeln, und vielleicht auch irgendwann einmal für einen Toaster ;-)

Die sich zunächst scheinbar hierfür anbietenden Sprachen C und C++ hatten Nachteile – es war nicht einfach, Software auf einen neuen Prozessor-Chip zu portieren.

So entstand Java, mit dem Ziel, sehr kleine, schnelle, zuverlässige und transportable Programme zu erzeugen.

2.1 Java

Java ist eine objektorientierte höhere Programmiersprache.

Objektorientiertes Programmieren ist die am häufigsten verwendete Programmiermethode seit der späten 90er Jahre.

Java ist von Anfang an vollständig objektorientiert. Es ist keine Sprache, in die Objektorientiertheit als nachträgliche Idee implementiert wurde (vergleiche C $\rightarrow$ C++).

Java unterstützt das Internet.

Java ermöglicht Ihnen das Schreiben von Programmen, die das Internet (Abkürzung für Interconnected Networks) und das WWW nutzen.

2.1 Java

Das World Wide Web (kurz Web oder WWW) ist ein über das Internet abrufbares Hypertext-System.

Java ist architekturunabhängig, d.h. es läuft (genauer gesagt der Java-Zwischencode) auf jedem Computer unverändert.

Die Grundlagen der Programmierung lernen Sie von dieser Woche an in den Übungen!

3. Zahlendarstellung im Computer

3. Zahlendarstellung im Computer

Jedem Zustand eines Bytes oder Words kann man eine Dezimalzahl zuweisen. Dazu fasst man die Zustände der einzelnen Bits als Binärzahl auf.

Der Dezimalwert einer Binärzahl berechnet sich wie folgt:

Die Bits der Zahl werden von rechts beginnend und mit der Zahl 0 startend durchnummeriert.

Anschließend bildet man eine Summe, in welcher man für jedes an der Position i gesetzte Bit die Zahl 2^i einsetzt.

3. Zahlendarstellung im Computer

Beispiele:

Zum Beispiel hat die Binärzahl 010 die Dezimaldarstellung $2^1 = 2$.

Die Zahl 010100100 hat die Dezimaldarstellung $2^2+2^5+2^7 = 164$.

Das höchstwertige Bit gibt in der Regel das Vorzeichen der Zahl an:

Beispiele:

2 = 010	**164 = 010100100**
-2 = 110	**-164 = 110100100**

Merke: **Der darstellbare Zahlenbereich wird von 2^{64} auf $+/-2^{63}$ eingeschränkt!**

3. Zahlendarstellung im Computer

Eine Alternative bietet das so genannte Einerkomplement.

Das Einerkomplement ist eine arithmetische Operation auf Binärzahlen.

Dabei werden alle Ziffern bzw. Bits gekippt, d.h. aus 0 wird 1 und umgekehrt.

Beispiel:

Zu $x = 01101100 (= 108)$ ergibt sich das Einerkomplement x_1 wie folgt:
$x_1 = 10010011$ **In der Summe ergibt sich dann:**
$x + x_1 = 11111111 = 2^8 - 1$.

3. Zahlendarstellung im Computer

Wenn man zum Einerkomplement noch 1 addiert, so erhält man das so genannte Zweierkomplement x_2:

$x_2 = x_1 + 1 = 10010100$.

Und schließlich:

$x + x_2 = 2^8 = 100000000 = 0$,

wenn man das neunte Bit einfach überlaufen lässt.

3. Zahlendarstellung im Computer

Merke:

Auch hier gibt das höchste Bit das Vorzeichen an.

Ferner gilt:

$x - y = x + y_2$ **,**

d.h. wir haben die Subtraktion auf die Addition des Zweierkomplements zurückgeführt.

3. Zahlendarstellung im Computer

Als nutzbarer Zahlenbereich stehen also für 2-Byte-Zahlen nur die Zahlen zwischen 0 und 2^{16} = 65355 oder, wenn man das höchste Bit als Vorzeichenbit interpretiert, zwischen -32768 und 32767 zur Verfügung.

Das sind die Zahlen, deren Zahlentyp man als INTEGER bezeichnet.

Da dieser Zahlenbereich in der Regel nicht ausreicht, stellen höhere Programmiersprachen zusätzlich so genannte Gleitkommazahlen zur Verfügung.

3. Zahlendarstellung im Computer

Gleitkommazahlen (Typ-Name float, double):

Die Gleitzahldarstellung kennt man z.B. vom Taschenrechner:

z.B. 1.234567E-10 für $1.234567 \cdot 10^{-10}$.

7-stellige Mantisse

Die Bilder der vorhandenen Gleitkommazahlen sind auf der Zahlengeraden unterschiedlich dicht.

3. Zahlendarstellung im Computer

Die Bilder der vorhandenen Gleitkommazahlen sind auf der Zahlengeraden unterschiedlich dicht.

Beispiel:

Es wird mit 7-stelliger Mantisse gerechnet.

Dann ist die Nachbarzahl von $1.0 \cdot 10^{-10}$ die Zahl $1.000001 \cdot 10^{-10}$, der Unterschied ist 10^{-16}.

Die Nachbarzahl von $1.0 \cdot 10^{10}$ ist $1.000001 \cdot 10^{10}$, der Unterschied ist 10^{4}.

3. Zahlendarstellung im Computer

Eine Folge daraus ist:

Das Addieren einer kleinen Zahl zu einer großen ändert diese evtl. nicht.

Phänomen der Auslöschung:

Die Subtraktion fast gleichgroßer Zahlen kann zu erheblichen Fehlern führen, da sich die führenden Ziffern gegenseitig auslöschen.

Fazit:

Vermeidbare Subtraktionen fast gleichgroßer Zahlen sollten vermieden werden.

→ Weitere Ausführungen an der Tafel …