

# Grundlagen der Informatik

## Vorlesungsskript

Prof. Dr. T. Gervens, Prof.-Dr.-Ing. B. Lang, Prof. Dr.-Ing. C. Westerkamp

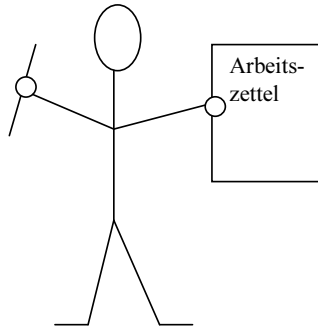
<b>1</b>	<b><u>AUFBAU UND ARBEITSWEISE EINES RECHNERS</u></b>	<b>2</b>
1.1	BEISPIEL: EIN RECHENKNECHT	2
1.2	DER DIGITALE COMPUTER	5
1.3	BEFEHLE DES DC	9
1.3.1	LADEN/SPEICHERN	10
1.3.2	ARITHMETISCHE BEFEHLE	10
1.3.3	LOGISCHE BEFEHLE	10
1.3.4	SPRUNGBEFEHLE	10
1.3.5	SHIFT-OPERATOR	10
1.3.6	EIN-/AUSGABEBEFEHLE	10
1.4	MIKROPROZESSOR 8080 (VORLÄUFER DER 80X86- UND PENTIUM-PROZESSOREN)	12
<b>2</b>	<b><u>BETRIEBSSYSTEME</u></b>	<b>14</b>
2.1	ÜBERBLICK	14
2.2	PROZESSVERWALTUNG	15
2.2.1	SCHEDULING	15
2.2.2	MÖGLICHE PROZESSZUSTÄNDE	17
2.2.3	UNIX	18
2.2.4	WINDOWS	19
2.3	SPEICHERVERWALTUNG	20
2.3.1	SPEICHERHIERARCHIE	20
2.3.2	AUSLAGERN VON PROGRAMMEN	20
2.4	DATEIVERWALTUNG	22
2.5	LISTE EINIGER UNIX-BEFEHLE	23
2.6	PROGRAMMIERSPRACHEN	24
2.6.1	KLASSEN VON PROGRAMMIERSPRACHEN	24
2.6.2	ERSTELLUNG EINES PROGRAMMS	24

# 1 Aufbau und Arbeitsweise eines Rechners

## 1.1 Beispiel: Ein Rechenknecht

Aufgabe: Berechnung einer Quadratwurzel  $\sqrt{x}$ ,  $x > 0$  (näherungsweise).

Rechenknecht:



Möglichkeiten:

- Grundrechenarten (zählen)
- eine Zahl im Kopf merken
- positiv/negativ unterscheiden
- Absolutbetrag bilden

Hilfsmittel: Papier, Bleistift, Kopf, Finger

Unsere Aufgabe: Zerlegung in einzelne Rechenschritte, die der Rechenknecht ausführen kann.

Idee: Wähle Schätzwert  $s$  für  $\sqrt{x}$  (z. B. 1)

Dann ergeben sich 3 Möglichkeiten:

i)  $s = \sqrt{x}$  (Volltreffer, unwahrscheinlich)

ii)  $s < \sqrt{x}$  ( $s > 0$ )  $\Rightarrow 1 < \frac{\sqrt{x}}{s} \Rightarrow \sqrt{x} < \frac{x}{s} \Rightarrow s < \sqrt{x} < \frac{x}{s}$

Der Zielwert liegt im Intervall zwischen  $s$  (untere Grenze) und  $\frac{x}{s}$  (obere Grenze)

Nächster Schätzwert aus Mittelwert:  $s' = \frac{s + \frac{x}{s}}{2}$ ;

Intervall verkleinert sich, bis die gewünschte Genauigkeit erreicht wird.

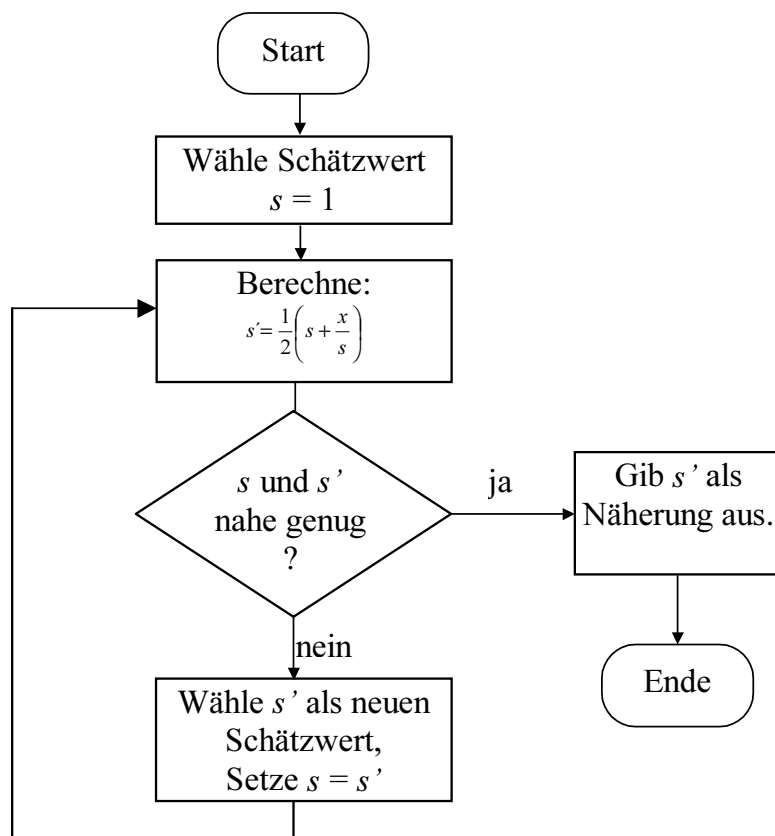
iii)  $s > \sqrt{x} \Rightarrow 1 > \frac{\sqrt{x}}{s} \Rightarrow \sqrt{x} > \frac{x}{s} \Rightarrow \frac{x}{s} < \sqrt{x} < s$

Der Zielwert liegt im Intervall zwischen  $\frac{x}{s}$  (untere Grenze) und  $s$  (obere Grenze)

Nächster Schätzwert aus Mittelwert:  $s' = \frac{s + \frac{x}{s}}{2}$ ;

Intervall verkleinert sich, bis die gewünschte Genauigkeit erreicht wird.

Vorgehensweise (Flussdiagramm)



Rechenknecht bekommt ein Blatt mit entsprechenden Anweisungen und Daten.  
(1 Zeile = 1 Rechenanweisung oder 1 Datenfeld). Dabei gilt:

- **Alles** (Daten, Felder für Zwischenergebnisse, Anweisungen) soll auf dem Datenblatt dargestellt werden.
- **A** bezeichne die Zahl, die er sich im Kopf merkt.
- Bei **Operationen** ist ein Operand immer die Zahl „A“ im Kopf. Das Ergebnis steht wieder in A.

**Arbeitszettel für Rechenknecht:**

<b>Zeilen-Nr</b>	<b>Inhalt</b>	<b>Kommentar</b>
<b>0</b>	<b>Gehe zur Zeile 7</b>	
<b>1</b>	<b>1000 (Beispiel)</b>	Feld für x
<b>2</b>	<b>anfangs 1</b>	Feld für s
<b>3</b>	<b>anfangs undefiniert</b>	Feld für s'
<b>4</b>	<b>0,00001</b>	Konstante 0,00001
<b>5</b>	<b>2</b>	Konstante 2
<b>6</b>	<b>Inhalt von Zeile 1 nach A übertragen</b>	
<b>7</b>	<b>Teile A durch Zeile 2</b>	$A=A/s$
<b>8</b>	<b>Addiere Zeile 2 zu A</b>	$A=A+s$
<b>9</b>	<b>Teile A durch Zeile 5</b>	$A=A/2$
<b>10</b>	<b>Trage Inhalt von A in Zeile 3 ein</b>	s'
<b>11</b>	<b>Subtrahiere Zeile 2 von A</b>	$A=s'-s$
<b>12</b>	<b>Ersetze A durch seinen Absolutbetrag</b>	$A= A $
<b>13</b>	<b>Subtrahiere Zeile 4 von A</b>	$A=A-0,00001$
<b>14</b>	<b>Falls <math>A \geq 0</math>, gehe zu Zeile 20</b>	$ s'-s  \geq 0,00001$ ?
<b>15</b>	<b>Gebe den Inhalt von Zeile 3 aus</b>	Ergebnis: s'
<b>16</b>	<b>Halte an</b>	
<b>17</b>	<b>Inhalt von Zeile 3 nach A übertragen</b>	$A=s'$
<b>18</b>	<b>Inhalt von A nach Zeile 2 übertragen</b>	$s=A$ (Ersetzung $s=s'$ )
<b>19</b>	<b>Mache weiter mit Zeile 6</b>	

## 1.2 Der digitale Computer

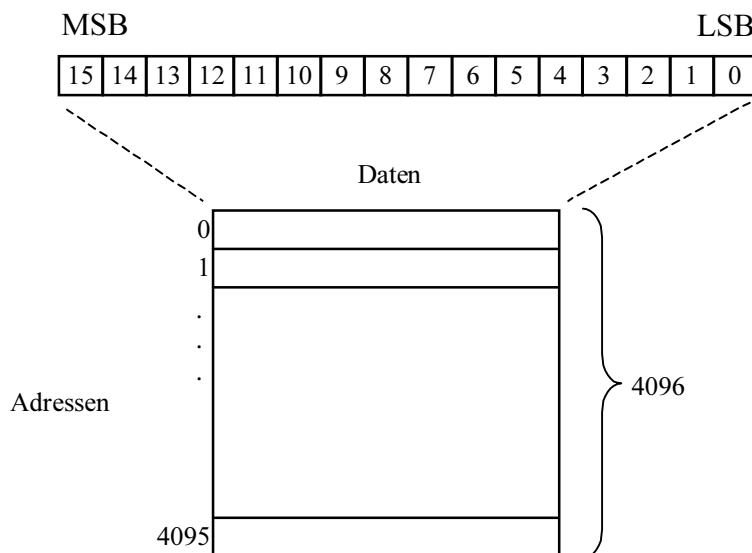
Schrittweiser Übergang: Rechenknecht → DC (Digital Computer)

### 1. Arbeitszettel → Hauptspeicher

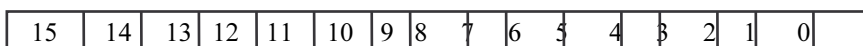
Der Hauptspeicher enthält ein Programm, dies umfasst sowohl Daten als auch Befehlsanweisungen.

Ein Programm stellt eine sinnvolle Folge von Befehlen mit zugehörigen Daten dar, die sequentiell abgearbeitet werden.

- Beispiel:** Speicher: - besteht aus  $4k = 4096 = 2^{12}$  Worte  
- jedes Speicherwort besteht aus 2 Bytes = 16 Bit  
- Zugriff auf ein Wort unter Angabe der Adresse (direkter Zugriff/direkte Adressierung)



### 2. Gemerkte Zahl A → Akkumulator: 16 Bit Wort (ACC)

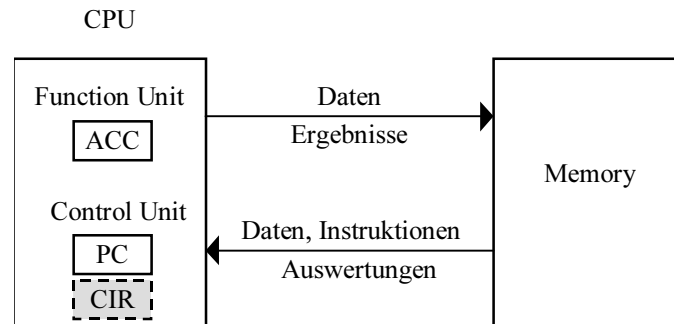


### 3. „Finger des Rechenknechts“ → Befehlszähler (Kontrolle) PC (Program Counter)

Der Befehlszähler (PC) enthält immer die Adresse des nächsten auszuführenden Befehls.

#### 4. Kopf des Mannes → CPU (Central Processing Unit)

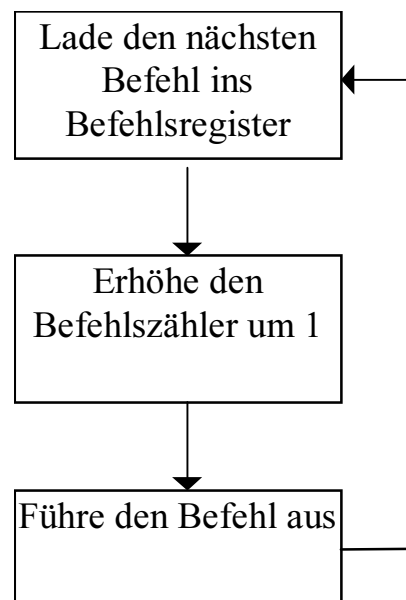
- bestehend aus:
- Steuereinheit (Control Unit);
  - Funktionseinheit (Function Unit, ALU)
  - Befehlsregister (CIR: Current Instruction Register)



Die Steuereinheit entschlüsselt (dekodiert) Befehle und steuert deren Ausführung. Der aktuell bearbeitete Befehl befindet sich im Befehlsregister. Zu jedem Zeitpunkt führt die CPU genau einen Befehl aus.

Die Aufgabe der Funktionseinheit besteht in der klassischen Verarbeitung der Daten, d.h. Ausführungen von Berechnungen, sie wird von der Steuereinheit gesteuert.

#### 5. Arbeitsweise: Fetch and Execute Cycle



## 6. Erweiterungen:

### a) Programm Status Wort:

Merk-Bits, welche Information über die letzte Operation in der Funktionseinheit geben:

Carry Flag	Meldet Überläufe bei der Berechnung.
Parity Bit	Gibt an, ob eine gerade Anzahl von Einsen im Akku steht.
Sign Flag	Speichert das Vorzeichen der Zahl im Akku.
Zero Flag	Meldet, dass der Wert im Akku gleich Null ist.

### b) **Ein- und Ausgabegeräte (input und output devices)**

- externe Speicher (Bänder, Platte, Disketten, etc.);
- Drucker
- Bildschirm
- Tastatur

### c) **BUS** (Verbindung aller Elemente)

- Steuerleitungen;
- Datenbus (n Leitungen = n Bits werden in einem Takt übertragen);
- Adressbus (n Leitungen =  $2^n$  Adressierungsmöglichkeiten)

### d) **Unterbrechungswerk**

**Zusatz:** Unterbrechung wegen bestimmter Signale/Ereignisse ermöglichen

**Ereignisse:** -

- äußere (z. B. Reset)
- innere (z. B. Fertigmeldung eine E/A-Gerätes)
- Ausnahmen (z. B. ungültiger Befehlscode, Division durch 0 etc.)

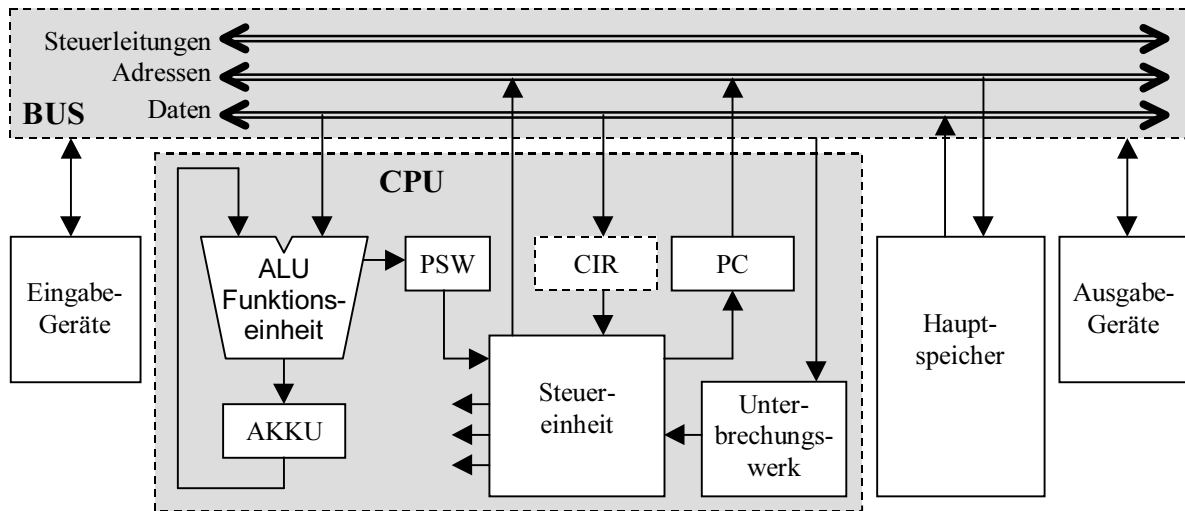
### **Reaktion des Prozessors**

- unterbricht seinen Zyklus
- nimmt eine von Signalen abhängige Maßnahme vor
- setzt seinen Zyklus fort (wenn möglich)

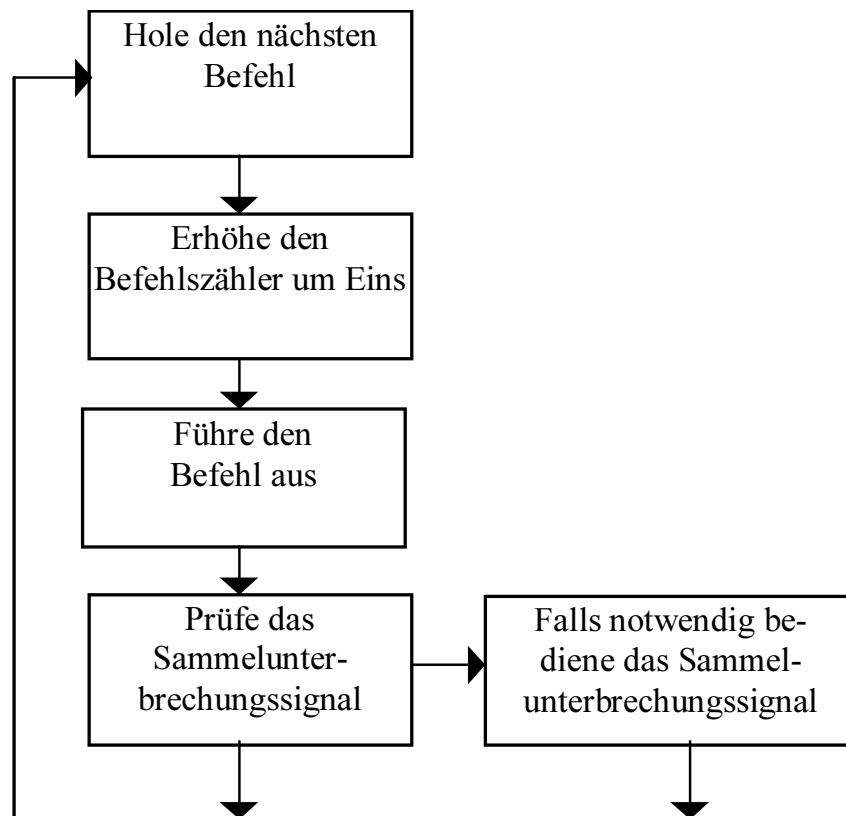
Dieser Zusatz wird vom **Unterbrechungswerk** verwaltet. Die Aufgaben des Unterbrechungswerkes sind dabei:

- Bildung des Sammelunterbrechungssignals
- Identifizierung der Startadresse einer Befehlssequenz im Hauptspeicher, die bei dieser Unterbrechung auszuführen ist
- Zurückweisen von Unterbrechungen
- Bestimmung der Rangfolge der Unterbrechungssignale

## Grobstruktur eines von Neumann-Rechners



## Der erweiterte Arbeitszyklus der CPU





## DC - Befehle:

Der Beispielrechner DC soll die folgenden Befehle kennen.

### 1.3.1 Laden/Speichern

LDA X	Lade den Inhalt der Adresse X in den Akku
STA X	Speicher Akkuinhalt an der Adresse X

### 1.3.2 Arithmetische Befehle

ADD X	Addiere den Inhalt der Adresse X zum Akkuinhalt
SUB X	Subtrahiere den Inhalt der Adresse X vom Akku
MPY X	Multipliziere den Inhalt der Adresse X mit dem Akku
DIV X	Dividiere den Akkuinhalt durch die Adresse X

### 1.3.3 Logische Befehle

AND X	Bitweises UND des Akku mit der Adresse X
OR X	Bitweises ODER des Akku mit der Adresse X

### 1.3.4 Sprungbefehle

JMP X	Setze Verarbeitung mit der Anweisung bei X fort
JLE X	Sprung zur Adresse X, falls Akkuinhalt $\leq 0$
JEQ X	Sprung zur Adresse X, falls Akkuinhalt = 0
JOV X	Sprung zur Adresse X, wenn ein „Overflow“ auftritt

### 1.3.5 Shift-Operator

SHR	zirkuläres Verschieben der Bits um eine Stelle nach „rechts“, Rotation, Bit ganz rechts wandert an die erste Stelle (ganz links)
-----	--

### 1.3.6 Ein-/Ausgabebefehle

WTA id	Akkuinhalt auf das Gerät id ausgeben
RDA id	Wert des Gerätes id in den Akku einlesen
SKP id	Überspringen der nächsten Anweisung, falls Gerät id nicht „busy“ z. B.: SKP id JMP a RDA id

## Beispiele: Kurze DC-Programme

a) Berechnung von:  $y = ax^2 + bx + c = (ax + b)x + c$ .

Start des Programmes bei Adresse 3000.

A = 2250	a
B = 2251	b
C = 2252	c
X = 2253	x
Y = 2254	y
3000	LDA A
3001	MPY X
3002	ADD B
3003	MPY X
3004	ADD C
3005	STA Y

b) Maximum-Bestimmung von a und b

Speicherzelle mit symbolischer Adresse A enthält den Wert  $a = 3$ .

Speicherzelle mit symbolischer Adresse B enthält den Wert  $b = 5$ .

Speicherzelle mit symbolischer Adresse Max nimmt das Ergebnis auf.

Start des Programmes bei Zelle 100

Adresse	Mnemocode	Akku-Inhalt	Befehlszähler(PC)
100	LDA A	a	3
101	SUB B	a-b	3-5=-2
102	JLE 105	a-b	-2
103	LDA A	a	
104	JMP 106	a	
105	LDA B	b	5
106	STA Max	b	5
107	... z.B. WTA 1		

c) Sprung, wenn Akku > 0: JGT a

100 JLE 102

101 JMP a

102 nächster Befehl...

## 1.4 Mikroprozessor 8080 (Vorläufer der 80x86- und Pentium-Prozessoren)

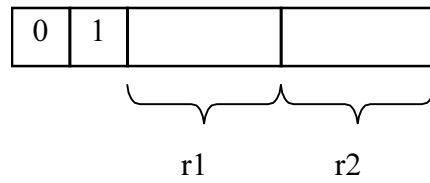
- Adressbus: 16 Bit =  $2^{16}$  Adressen = 64 k, 1 Speicherwort: 8 Bit
- Datenbus: 8 Leitungen
- Steuerbus: 4 Leitungen

- Speicher:



- Befehlsablauf: sequentiell (Fetch and Execute)  
Abweichungen: - Pausen;  
- Programmsprünge;  
- Unterprogramme.
- Befehlsregister: 8 Bit plus 2 Hilfsregister für 1-3 Byte-Befehle
- 16 Bit Stapelanzeiger (Stackpointer SP)
- Programmstatusregister F (Flags)
  - S: Sign-Flag
  - Z: Zero-Flag
  - P: Parity-Flag
  - C: Carry-Flag
  - AC: Auxiliary Carry Flag (Überlauf beim ersten Halbbyte beim Befehl DAA)
- Befehle (244):
  - 96 Befehle für Datenverschiebung
  - 65 Befehle für Arithmetik
  - 43 logische Befehle
  - 36 Befehle für Programmsprünge
  - 4 sonstige Steuerbefehle

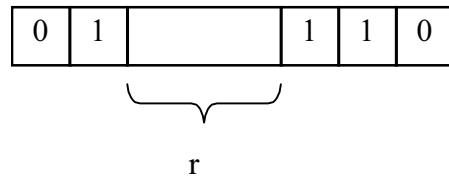
z. B. **MOV r1,r2** Inhalt Reg r2 → Reg r1



111 - Reg A    001 - Reg C  
 000 - Reg B    010 - Reg D

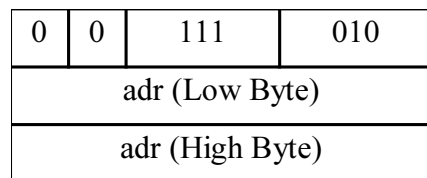
(3 Bits = 1 Register)

**MOV r, M** Inhalt Memory M → Reg r



M: H, L Register  
 (Die Speicheradresse wird durch H, L angegeben)

**LDA adr**                      Inhalt der Adresse adr in den Akku laden



## 2 Betriebssysteme

### 2.1 Überblick

Das Betriebssystem stellt ein Interface (Schnittstelle) zwischen dem Benutzer und der Rechnerhardware dar. Es besteht aus einer Vielzahl von Programmen

- zur **Überwachung und Verwaltung der Hardware** (CPU, Hauptspeicher, Sekundärspeicher, Peripheriegeräte, Drucker, Terminals, etc.);
- zur **Steuerung des Prozessablaufs**, inklusive Prozesserschöpfung, Prozessverdrängung und Prozesskommunikation;
- zur **Behandlung von Hard- und Softwarefehlern**, zu internen Diagnoseabläufen, zum Datenschutz und zur Datensicherheit;
- zur **Programmentwicklung** (Sprachübersetzer, Compiler, Editieren, Debugger, etc).

Bekannte Betriebssysteme:

- UNIX (Varianten: HP-UX, AIX, , XENIX, LINUX, SOLARIS, POSIX)
- MS-DOS
- VMS (Fa. DEC)
- OS/2
- Windows9x
- Windows NT; Windows 2000, Windows XP
- Mac OS
- Echtzeitbetriebssysteme: VxWorks, OSE, pSOS, Windows NT Embedded
- Betriebssysteme für kleine und mobile Plattformen: EPOC, Windows CE

Das Betriebssystem besteht aus mehreren Schichten:

Prozess	Prozess	Prozess	Benutzer- schnittstelle (Shell)
Prozessver- waltung		Datenver- waltung	
Ressourcenverwaltung (für Speicher, Festplatte,...)			
Betriebssystem-Kern (Prozessumschalter (Scheduling), Gerätetreiber, Unterbrechungsmodule,...)			
Hardware			

Die unteren Schichten sprechen die Hardware an (Gerätetreiber), bedienen Unterbrechungen (Unterbrechungsmodule) und ermöglichen das Umschalten (Scheduling) zwischen Prozessen. Weiterhin werden die Ressourcen des Rechners verwaltet. Diese sind z.B. der Hauptspeicher und der Festplattenspeicher. Weitere Module des Betriebssystems verwalten die laufenden Prozesse und die Daten des Rechners (Dateisystem).

Die oberste Schicht ist die Prozessschicht. Hier laufen die Anwenderprozesse. Ein hervorgehobener Prozess ist die Benutzerschnittstelle. Diese ermöglicht das Bedienen des Rechners und das Starten von neuen Prozessen.

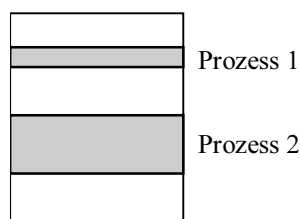
Unter UNIX erfüllt die Shell diese Aufgabe. Sie ermöglicht dem Benutzer die Eingabe von Kommandos als Text. Die Shell beinhaltet einen Kommandointerpreter, der die Kommandos auswertet und zu deren Ausführung zugehörige Prozesse startet. Alternativ kann eine graphische Oberfläche als Benutzerschnittstelle dienen. Unter Windows dient dazu der Explorer, der in verschiedenen Ausprägungen (Desktop, Windows-Explorer, Internet-Explorer) dem Benutzer die Bedienung des Rechners ermöglicht. Auch bei UNIX kann eine grafische Bedienung des Rechners erfolgen. Basierend auf der Grafiksoftware X-Windows und Aufsätzen wie KDE sind Dateimanager-Programme ähnlich dem Windows-Explorer verfügbar, welche ein interaktives Bedienen des Rechners ermöglichen.

Bei eingebetteten Geräten (Steuerungen, Telekommunikationsgeräten) geht der Trend zu browser-basierten Benutzerschnittstellen (Web-Server im Gerät).

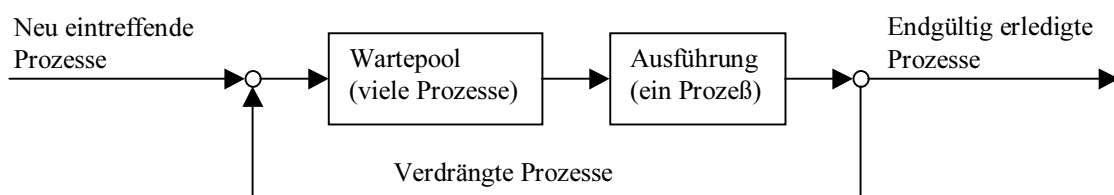
## 2.2 Prozessverwaltung

### 2.2.1 Scheduling

Ein Prozess (Job, Task) ist die Instanz eines ausführbaren Programmes im Speicher, die ausgeführt wird, oder zur Ausführung ansteht. In einem System mit einem Prozessor kann immer genau ein Prozess ausgeführt werden.

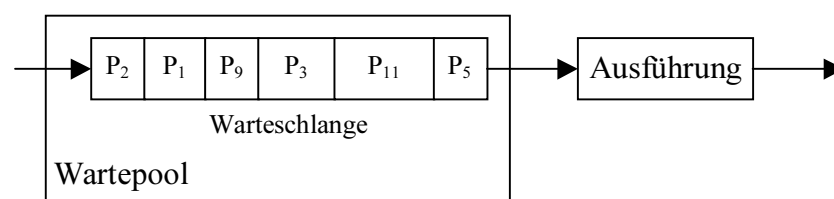


Das Betriebssystem hat die Aufgabe des **Scheduling's**: Es wählt den Prozess aus, der zur Ausführung kommt und bestimmt wann ein Prozess verdrängt wird.



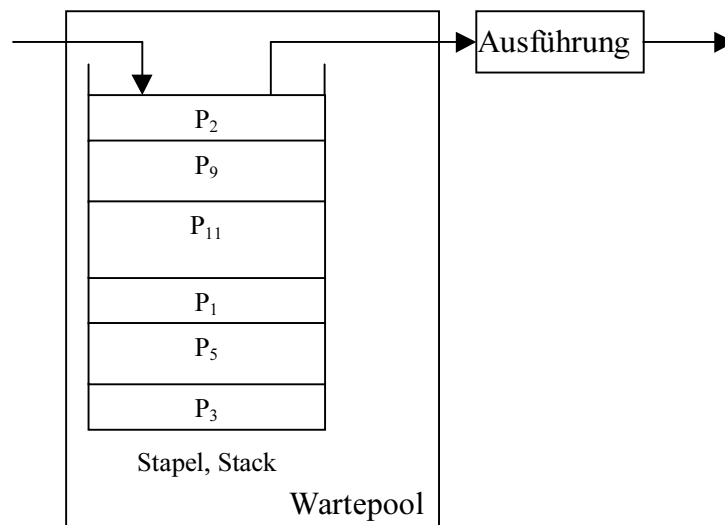
### Scheduling Strategien

#### 1. FIFO (First In First Out)



Bei Verwendung eines Wartepools mit FIFO-Strategie werden die Prozesse in der Reihenfolge, mit der sie in den Wartepool eingestellt wurden, zur Ausführung gebracht.

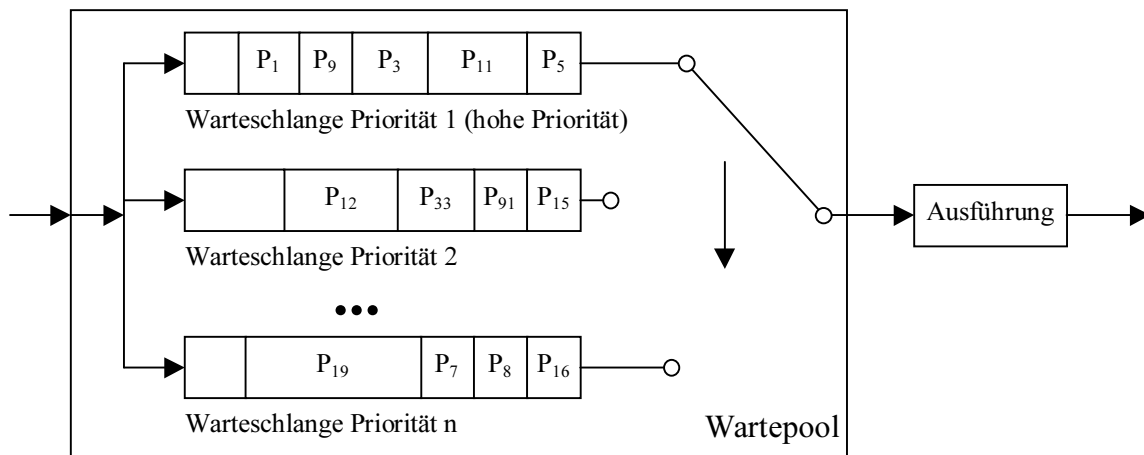
## 2. LIFO (Last In First Out)



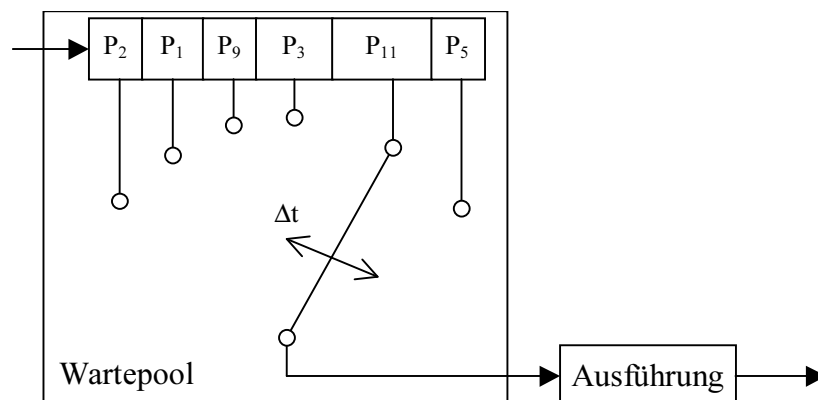
Bei Verwendung eines Wartepools mit LIFO-Strategie (Stapel, Stack) wird der Prozess, der zuletzt in der Wartepool eingestellt wurde, als erster zur Ausführung gebracht.

## 3. Prioritätensteuerung

Jeder Benutzer erhält eine bestimmte Priorität und für jede Priorität gibt es eine Warteschlange. Prozesse mit hoher Priorität werden vorgezogen.



## 4. Time Sharing-System



Das Time Sharing System bringt die Prozesse nach einem bestimmten Zyklus für einen Zeitabschnitt  $\Delta t$  (Zeitscheibe) zur Ausführung, dann schaltet es zu einem anderen Prozess um. Damit wird ein Prozess schrittweise ausgeführt bis er abgearbeitet ist.

Im praktischen Einsatz werden meist die Scheduling-Strategien kombiniert. In UNIX findet man beispielsweise eine Kombination des Time Sharing mit der Prioritätenvergabe.

Andere weitergehende Scheduling-Strategien (inkl. Optimierungsstrategien) werden benötigt im:

- Mehrprozessorbetrieb
- Parallel Processing
- Massive Parallel Processing

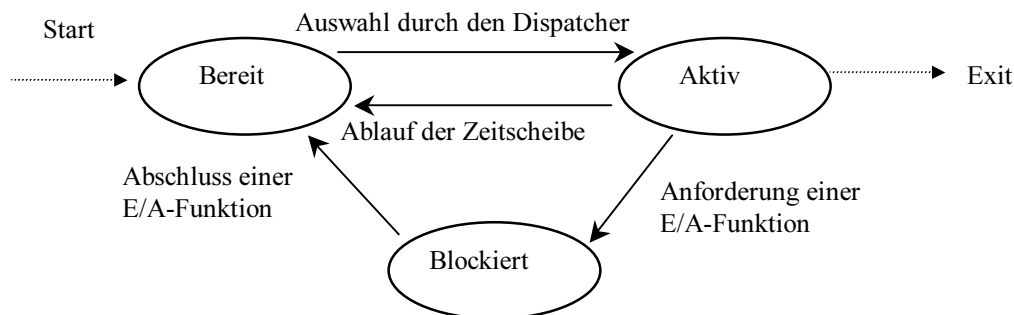
## 2.2.2 Mögliche Prozesszustände

1. Prozess ist aktiv (Unterschiede: Benutzermodus/Systemmodus).
2. Prozess ist nicht aktiv, aber bereit, d. h. wartet auf Zuweisung durch den Scheduler.
3. Prozess ist blockiert, d. h. wartet auf eine E/A-Operation.

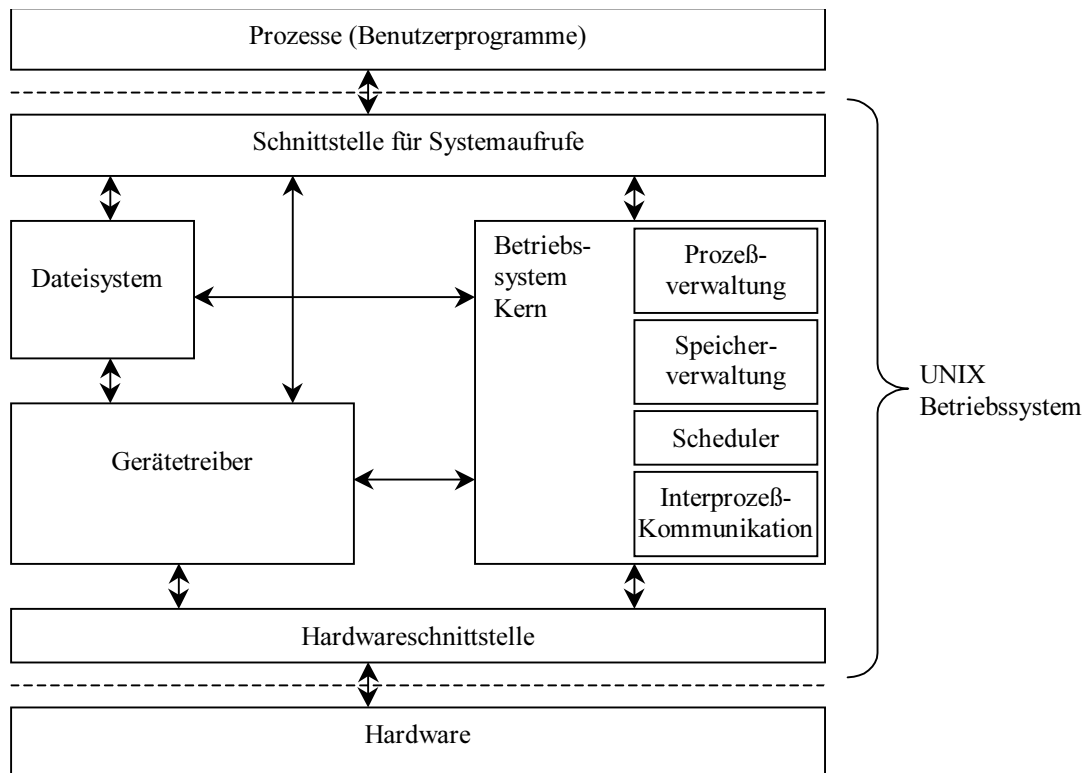
Für die Umschaltung auf einen neuen Prozess ist der Dispatcher zuständig. Er wird angestoßen, falls:

- das Zeitquantum für einen Prozess abgelaufen ist (Time Sharing)
- ein aktiver Prozess eine E/A-Anforderung stellt
- ein Prozess beendet ist

Nachfolgendes Diagramm zeigt ein einfaches Modell der Prozesszustände und beschreibt die Übergänge zwischen den Zuständen für ein Time Sharing System:



## 2.2.3 UNIX



Das UNIX Betriebssystem erlaubt das Ausführen mehrerer Prozesse nach Prioritätensteuerung und Time Sharing. Die Prozesse kommunizieren mit dem Betriebssystem über Systemaufrufe, für welche eine eindeutige Schnittstelle existiert.

Das Betriebssystem UNIX ist unterteilt in mehrere Komponenten. Der Betriebssystem Kern verwaltet den Speicher und die laufenden Prozesse. Der Scheduler ist für das Umschalten zwischen den Prozessen zuständig. Weiterhin ermöglicht der Kern eine Kommunikation zwischen Prozessen (Interprozesskommunikation).

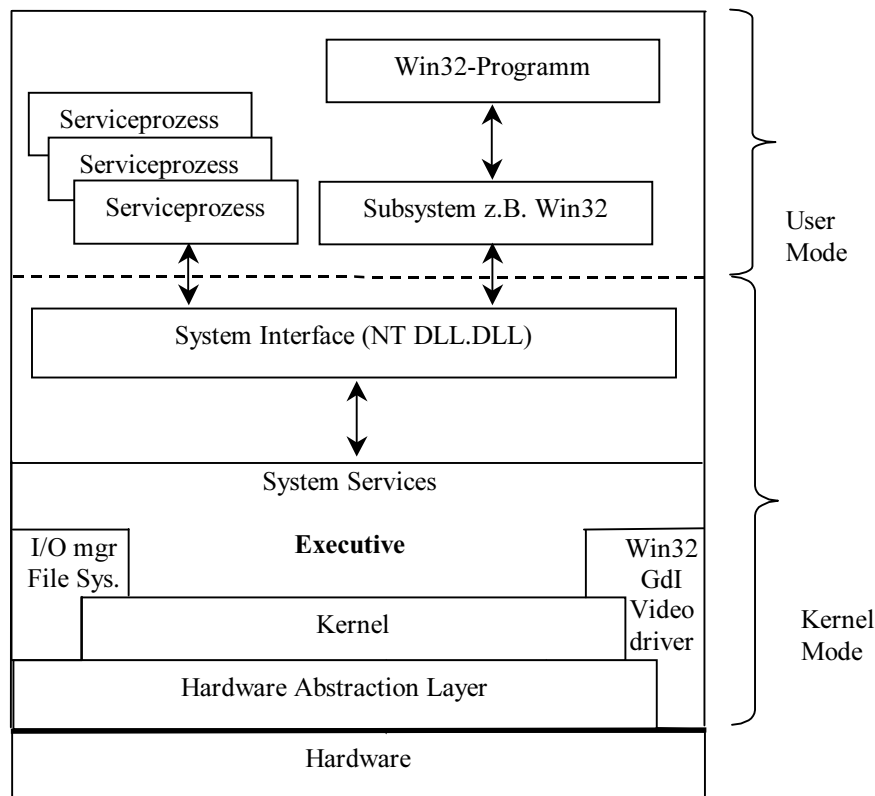
Die Gerätetreiber kapseln Hardwareeinheiten und geben unterschiedlicher Hardware die gleiche Software-Schnittstelle. Beispielsweise werden unterschiedliche Festplatten oder Grafikkarten durch Treiber gekapselt und können einheitlich durch die Software angesprochen werden.

Das Dateisystem organisiert die Plattenspeicher (Festplatte, Diskette, CD-ROM ...) und ermöglicht den Zugriff über eine logische Dateistruktur. Diese Dateistruktur bildet das Dateisystem ab auf die physikalischen Spuren und Sektoren der Platten.

UNIX ist unabhängig vom Prozessor (Pentium, PowerPC, StrongARM, ...). Ein kleiner Teil wird an den Prozessor angepasst. Dieser Teil ist in einer Hardwareschnittstelle zusammengefasst.

UNIX ist hauptsächlich in der Sprache C programmiert. Teile der Hardwareschnittstelle müssen in der Assemblersprache des Prozessors geschrieben werden.

## 2.2.4 Windows

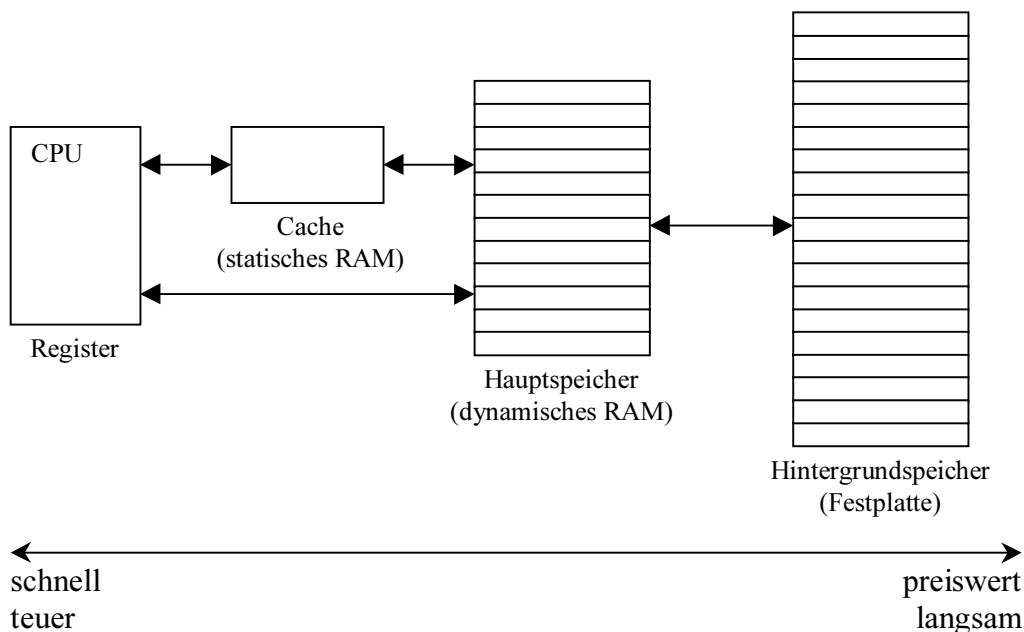


Windows entstand als grafische Benutzeroberfläche für das einfache PC-Betriebssystem DOS. Dieses war für genau einen Prozessortyp (X86) in großen Teilen in Assembler programmiert. Seit Windows NT ist gibt es Versionen für mehrere Prozessorplattformen, auch wenn weiterhin der Schwerpunkt auf Pentium-Prozessoren (Windows 9x, NT, 2000, XP) bzw. StrongARM (Windows CE, Pocket PC 2002) liegt.

Die unterliegende Hardware wird (bis auf direkten Video-Zugriff) durch einen Hardware Abstraction Layer (HAL) gekapselt. Dieser überträgt alle Hardware-Zugriffe des Kernels und des Dateisystems an die Hardware. Der Executive enthält mehrere Management-Dienste, die Steuerung von Objekten, Cache, Prozessen, Konfiguration etc. übernehmen. Alle Operationen unterhalb der System-Services-Ebene finden im Kernel-Mode statt und sind damit dem direkten Zugriff von Benutzerprogrammen entzogen. Bei der Erzeugung von Kernel-Objekten (Dateien, Prozesse, Threads, Pipes etc.) erhält der aufrufende Prozess ein „Handle“, über das die weitere Nutzung des Objektes geschieht. Benutzerprogramme nutzen die sehr umfangreiche Win32-API (Application Programming Interface). So gibt es über 60 Funktionen für Dateizugriff und mehr als 1000 für die grafische Benutzeroberfläche. Viele Funktionen lassen sich auf mehrere Arten realisieren. Der Zugriff geschieht über das Win32-Subsystem auf die System Services. Weitere Programmiermodelle und Subsysteme sind in einigen Versionen realisiert (z.B. OS/2 bis Windows NT). Die Verwaltung der Konfiguration und der Eigenschaften von Benutzerprogrammen geschieht über eine große Datenbank (Registry).

## 2.3 Speicherverwaltung

### 2.3.1 Speicherhierarchie



In Rechnern besteht eine Speicherhierarchie mit den Registern in der CPU an der Spitze und dem Hintergrundspeicher an der Basis. Der Speicher an der Spitze der Hierarchie ist sehr schnell im Zugriff aber auch sehr teuer. Der Hintergrundspeicher an der Basis ist preiswert, aber der Zugriff auf diesen Speicher ist langsam.

Cache-Speicher stellt ein Bindeglied zwischen CPU und Hauptspeicher dar, in dem die als nächstes benötigten Daten zwischengespeichert sind. (90:10-Regel: Bei 90% aller Zugriffe werden nur 10% der Daten verwendet). Speichert man diese 10% der Daten in einem schnellen Cache-Speicher, können somit 90% aller Zugriffe beschleunigt durchgeführt werden.

Für die Speicherverwaltung wird zwischen physikalischem und logischem Adressraum unterschieden:

- physikalischer Adressraum: reale Hauptspeicher
- logischer Adressraum: der durch ein Programm ansprechbare Adressraum.  
(virtuell)

Die Umsetzung zwischen logischen und physikalischen Adressen erfolgt mit Hardware und wird durch das Betriebssystem gesteuert. Der Programmierer von Benutzerprogrammen braucht sich darum nicht kümmern.

### 2.3.2 Auslagern von Programmen

Bei gleichzeitiger Ausführung mehrerer Prozesse reicht der Hauptspeicher als Speicherraum nicht aus. Daher werden dann Teile der Programme in den Hintergrundspeicher ausgelagert. Das Betriebssystem führt nach bestimmten Strategien (Paging, Swapping) dieses Auslagern durch.

Hierzu werden Hauptspeicher und Hintergrundspeicher in Seiten (Pages) bzw. Kacheln (Page-Frames) gleicher Größe (z. B. 4K) unterteilt. Die im Programm angesprochene Adresse steht nun entweder im Haupt- oder im Hintergrundspeicher. Steht sie nicht im Hauptspeicher (Page-Fault), so wird sie aus dem Hintergrundspeicher nachgeladen und

für die Verarbeitung verfügbar gemacht. Ist im Hauptspeicher kein Platz, so muss eine andere Seite des Hauptspeichers in den Hintergrundspeicher ausgelagert werden.

### **Auslagerungsstrategien**

- FIFO (First In First Out)  
Die am längsten im Hauptspeicher liegende Seite wird ausgelagert.
- LRU (Least Recently Used)  
Die am längsten nicht benutzte Seite wird ausgelagert.
- LFU (Least Frequently Used)  
Die bislang am wenigsten benutzte Seite wird ausgelagert.

Bei jeder Strategie sind entsprechende Listen von Zugriffen oder Zeiten zu führen.

In der Praxis: LRU

### **Einlagerungsstrategien**

- demand paging Die Seite wird bei aktuellem Bedarf angefordert.
- preplanned paging Der Seitentransport wird vorgeplant, so dass die Seite bei Bedarf schon zur Verfügung steht. Hierzu sind Kenntnisse über den Programmablauf erforderlich.

Übliche Strategie: demand paging

## 2.4 Dateiverwaltung

Die Dateiverwaltung gibt dem Anwender eine logische Sicht auf die Daten der Massenspeicher (Festplatten, Disketten, ...) des Rechners. Diese logische Sicht basiert auf Dateien und Verzeichnissen.

**Datei:** Einheit zusammenhängender Zeichen.

In der Regel werden diese Zeichen dem Programm in sequentieller Reihenfolge zur Verfügung gestellt.

Eine Datei besitzt einen Dateinamen und ist einem Verzeichnis zugeordnet.

**Verzeichnis:** Zugriffspfad auf Dateien innerhalb einer Verzeichnisstruktur.

Verzeichnisstrukturen sind üblicherweise in einer Baumstruktur angelegt.

Die Dateiverwaltung muss somit die physikalische Struktur der Massenspeicher umsetzen auf die gewünschte logische Sicht. Zu berücksichtigen sind:

- die physikalische Sicht (Zugriff, Struktur, Aufbau, Ablage, Einteilung in Blöcke)
- die logische Sicht (Dateiname, Dateiverzeichnisse, Dateiorganisation, Dateiformate, Datensicherheit, Datenschutz)

**Dateiname:** In vielen Betriebssystemen besteht der Name aus zwei Feldern, dem Namensfeld und dem Typfeld (DOS, VMS, OS/2, nicht UNIX)

<b>Bsp.:</b>	Schach.c	(C-Programm)
	Schach.obj	(übersetztes Programm)
	Schach.exe	(ausführbares Programm)
	Readme.txt	(zu lesender Text)
	DV.doc	(z. B. Word Dokument)

In UNIX sind Dateinamen nicht strukturiert. Der Punkt kann Bestandteil des Namens sein. Steht der Punkt an erster Stelle, so wird die Datei beim Auflisten der Dateien nicht mit angezeigt.

Durch Verwendung von Kürzelzeichen (Wildcard) können mehrere Dateien gleichzeitig angesprochen werden.

z.B. UNIX: \* null oder mehrere Zeichen  
? genau ein beliebiges Zeichen (kein Leerzeichen!)  
[..]genau eins der Zeichen in der Klammer, wobei  
Abkürzungen wie a-z oder 1-20 möglich sind.

<b>Bsp.:</b>	text.*	text.txt, text.1, text.doc, ...
	tex?.txt	text.txt, tex1.txt, tex2.txt, ...
	text.[1-4]	text.1, text.2, text.3, text.4
	a.[a-c]	a.a, a.b, a.c

**Verzeichnisse:** Dateien werden in der Regel in hierarchisch angelegten Verzeichnissen (Directories) angeordnet. (In UNIX: ein einziger Baum für alle Datenträger.)

/	root directory
/bin	Dienstprogramme
/etc	weitere Programme
/etc/passwd	Passwords
/usr	
/home5/g65/std7372	

Der volle Dateiname besteht aus Pfadname und Dateiname, z. B.:

/home6/westerka/dv.doc

## 2.5 Liste einiger UNIX-Befehle

<code>login &lt;username&gt;</code>	Anmelden
<code>who</code>	Anzeige der aktiven Benutzer
<code>date</code>	Datum und Uhrzeit
<code>pwd</code>	aktuelles Verzeichnis wird angezeigt
<code>cd /</code>	gehe ins Wurzelverzeichnis
<code>cd</code>	gehe ins Home-Verzeichnis
<code>cd ..</code>	gehe ins übergeordnete Verzeichnis
<code>cd &lt;pfadname&gt;</code>	gehe ins Verzeichnis <pfadname>
<code>mkdir</code>	Verzeichnis erstellen
<code>rmdir</code>	Verzeichnis löschen
<code>ls</code>	Listing der Dateien
<code>ls -l</code>	Listing mit ausführlichen Informationen
<code>ls -a</code>	Listing inkl. versteckter Dateien
<code>rm</code>	Löschen von Dateien
<code>cp</code>	Kopieren
<code>mv</code>	Verschieben/Umbenennen von Dateien
<code>cat</code>	Anzeigen von Dateien
<code>more</code>	seitenweises Anzeigen von Textdateien
<code>chmod</code>	Ändern der Zugriffsrechte (ugo±rwx)
<code>textedit</code> oder <code>te</code>	Aufruf des Texteditors
<code>emacs</code>	Aufruf des Editors Emacs
<code>ps</code>	Programm zur Anzeige der Prozesse
<code>passwd</code>	Ändern des Paßwortes
<code>cc</code> oder <code>gcc</code>	Aufruf des C-Compilers
<code>grep muster datei.*</code>	Finden muster in datei.*
<code>mp test.c &gt;test.ps</code>	Erstellen einer Postscript-Datei
<code>ftp</code>	Programm zur Datenübertragung auf Basis des Internetprotokolls TCP/IP
<code>lpr -Pa4ps</code>	Ausdrucken einer Postscript -Datei auf dem Schnelldrucker

## 2.6 Programmiersprachen

Programmiersprachen sind künstliche Sprachen zur präzisen Formulierung von Algorithmen. Ein Algorithmus ist dabei ein allgemeiner Lösungsweg der festlegt, wie man von der Aufgabenstellung schrittweise zur Lösung kommt. Ein Programm ist die Codierung eines Algorithmus in einer Programmiersprache.

### 2.6.1 Klassen von Programmiersprachen

Man unterscheidet folgende Klassen von Programmiersprachen:

Sprachen der 1. Generation: Maschinensprachen

Sprachen der 2. Generation: Assemblersprachen

Sprachen der 3. Generation:

FORTAN (ab 1954, wissenschaftliche und technische Anwendungen)

COBOL (ab 1957, kommerzielle Anwendungen)

ALGOL (1957, wissenschaftliche und technische Berechnungen)

PL/1

BASIC (1963, einfache Problemstellungen, Lehre)

Pascal (1968, allgemeine Probleme, Lehre)

C (1970, allgemeine Probleme, Systemimplementierungsaufgaben)

Ada (1975, Systemimpl. vor allem im sicherheitskritischen Bereichen)

Modula (1975, Lehre)

C++ (1980, objektorientierte Anwendungsprogr., Benutzeroberflächen)

JAVA (1995, objektorientierte Sprache, portabel, netzwerkfähig)

Sprachen der 4. Generation: Toolsprachen, Datenbanksprachen wie SQL

Sprachen der 5. Generation: deklarative Sprachen zur Beschreibung des Problems (Forschungsgegenstand).

In der Vorlesung und in den Praktika beschäftigen wir uns kurz mit Assemblersprachen (2. Generation) und ausführlich mit C (3. Generation).

### 2.6.2 Erstellung eines Programms

Die Programmierung einer Aufgabenstellung erfolgt in mehreren Schritten

1. **Spezifikation** der Aufgabenstellung: Textuelle und graphische Darstellung des Problems, welches durch das Programm gelöst werden soll. Eine möglichst genaue Spezifikation des Verhaltens, welches das Programm zeigen soll, verhindert Missverständnisse zwischen dem Auftraggeber (Kunden) und dem Auftragnehmer (Programmierer). Die Spezifikation sollte vom Auftraggeber und vom Auftragnehmer abgenommen werden.
2. Erstellen und Skizzieren eines **Lösungsplans (Algorithmus)**. Ein solcher Lösungsplan kann als **Verlaufsplot**, **Struktogramm** oder in **Pseudocode** beschrieben werden.
3. Erstellen des sogenannten **Quell-Programms** (Source-Code) in der vorgegebenen Sprache mit Hilfe eines ASCII-Editors (bei uns mit dem Texteditor oder Emacs).
4. **Übersetzen/Compilieren** des Quellprogramms in Maschinensprache (Objektprogramm). Dieser Schritt wird durch ein Programm ausgeführt, welches als „Compiler“ bezeichnet wird. Der im Praktikum verwendete Compiler heißt „cc“ oder „gcc“. Beim Compilieren führt das Compiler-Programm mehrere Schritte durch:
  - In einer *lexikalischen Analyse* werden logisch-zusammenhängende Tokens gelesen.
  - In einer *syntaktischen Analyse* wird die Struktur des Quellprogramms aufgrund von Regeln erkannt.
  - Schließlich wird der *Maschinencode* erzeugt und anschließend *optimiert*.

5. **Binden (Linken):**Das Objektprogramm (evtl. auch mehrere) wird durch Hilfsprogramme (bei uns mit cc oder gcc) zu einer ausführbaren Datei gebunden bzw. gelinkt. Die ausführbare Datei heißt bei UNIX und Verwendung des cc- oder gcc-Compilers standardmäßig „a.out“.
6. **Ausführen und Testen:** Das Programm wird in den Hauptspeicher des Rechners geladen und ausgeführt. Es wird getestet, ob Spezifikation erfüllt wird, sonst muss nachgebessert werden (Zurück zu Punkt 3, Punkt 2 oder sogar zu Punkt 1, je nach Fehler).  
Zum Testen von Programmen dienen häufig „Debugger“-Programme, die ein „Hineinsehen“ in den Programmcode und die Inspektion von Variablen während des Programmlaufs ermöglichen.
7. **Dokumentation:** Nach erfolgreichem Test des Programms muss es dokumentiert werden. Die Dokumentation soll es einem fremden Programmierer ermöglichen, in kurzer Zeit den Quellcode und die hinterliegenden Algorithmen zu verstehen, um das Programm warten und modifizieren zu können.
8. **Erstellung einer Release:** In einer Release stellt man alle Informationen (Spezifikation, Lösungsplan, Quellcode, Anweisungen zum Übersetzen des Quellcodes, verwendete Compiler, Dokumentation) eines Programms zusammen, um zu einem späteren Zeitpunkt aus dem Release wieder das Programm erstellen zu können.  
Hilfe bei der Release-Erstellung bieten Konfigurationsmanagement-Tools (Versionsverwaltung). Sie speichern die Dateien eines aktuellen Entwicklungsstands mit Versionsinformation und ermöglichen jederzeit ein Abrufen genau dieses Standes.

