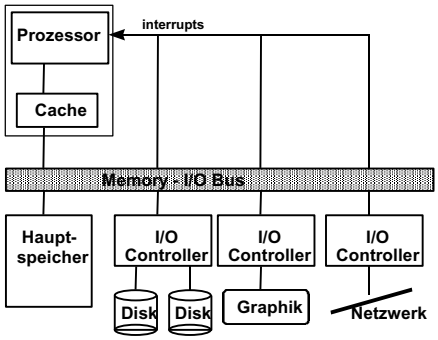
Überblick

- Einleitung
  - Lit., Motivation, Geschichte, v.Neumann-Modell, VHDL
- Befehlsschnittstelle
- Mikroarchitektur
- Speicherarchitektur
- Ein-/Ausgabe
- Multiprozessorsysteme, ...

Kap.4

Speicherarchitektur

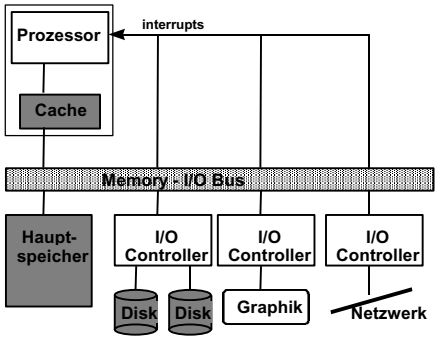
Prinzipieller Aufbau eines Rechners



Bestandteile:

- Prozessor mit Cache
- Hauptspeicher
- Externe Speicher
- Eingabegeräte (Tastatur, Maus)
- Ausgabegeräte (Bildschirm, Drucker, Plotter)
- Busse

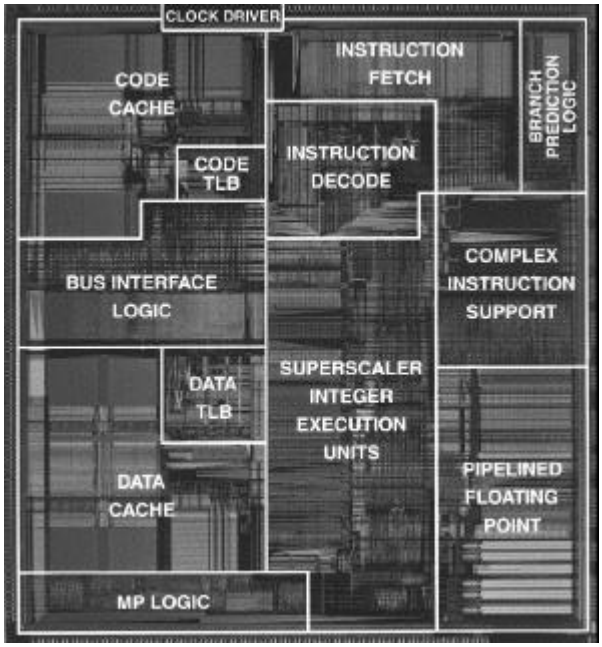
Prinzipieller Aufbau eines Rechners



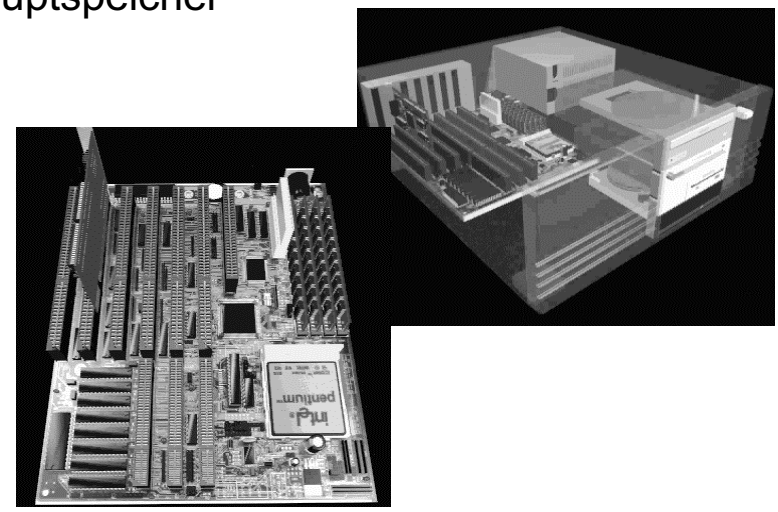
Bestandteile:

- Prozessor mit **Cache**
- **Hauptspeicher**
- **Externe Speicher**
- Eingabegeräte (Tastatur, Maus)
- Ausgabegeräte (Bildschirm, Drucker, Plotter)
- Busse

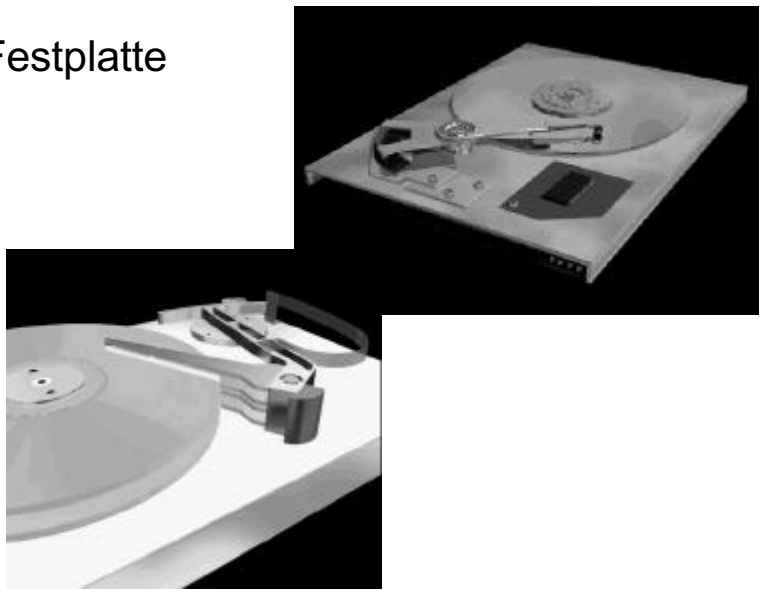
Cache



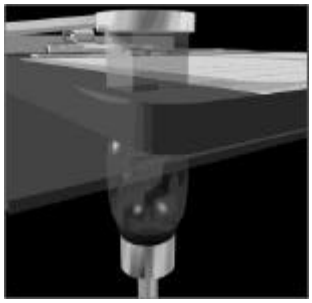
Hauptspeicher

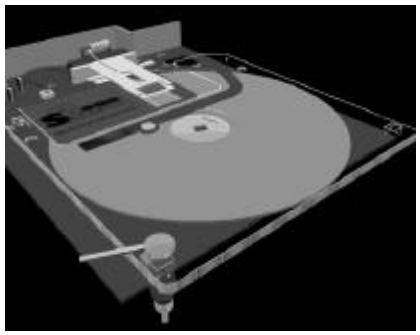


Festplatte

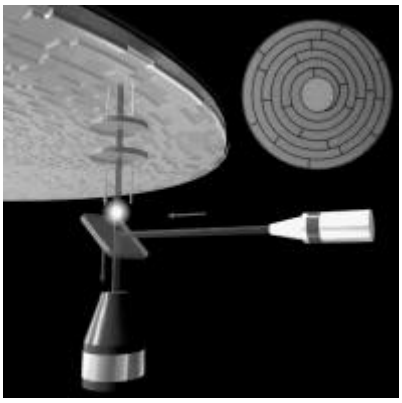


Floppy





CD-ROM / DVD



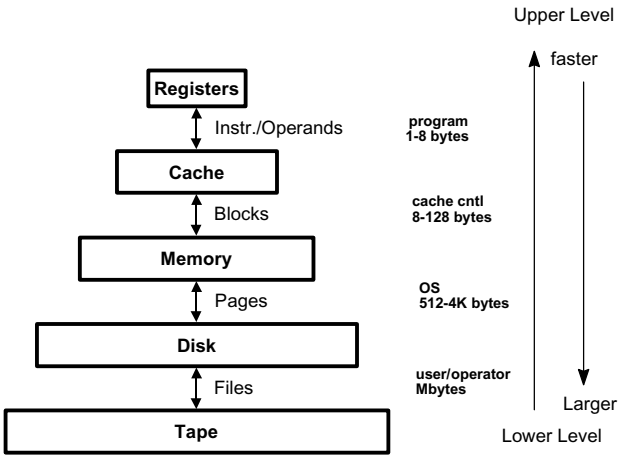
Speicher

- Ansammlung von Speicherzellen
- Speicherzelle:
  - speichert Information aus k Bits
  - k ist fest und heißt Wortbreite
  - kann über Adresse angesprochen werden
  - ist kleinste adressierbare Einheit

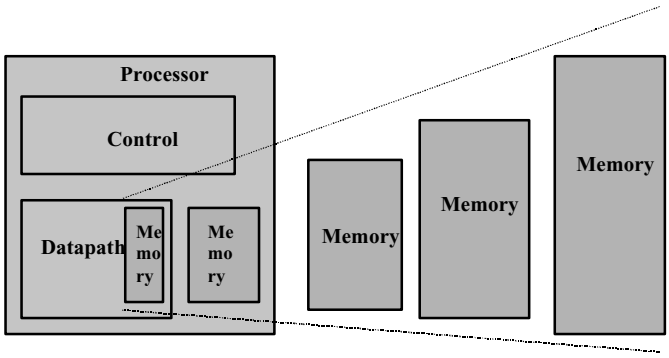
Speicher (4)

- **Kapazität**
  - Anzahl der Speicherzellen
  - Speicherwortbreite (z.B. 4 Byte=32 Bit)
- Aktivierung eines Speicherbausteins durch Speichermodul-Auswahl-Signal (CS=chip select)
  - Bei Schreib-Lese-Speichern gibt es Schreib-Lese-Signal (R/W=read/write)
  - Zugriffszeit beim Lesen und Schreiben nennt man auch Speicherzyklus
- Geschwindigkeit <--> Preis
  - daraus folgt **Speicherhierarchie**

Speicherorganisation heute

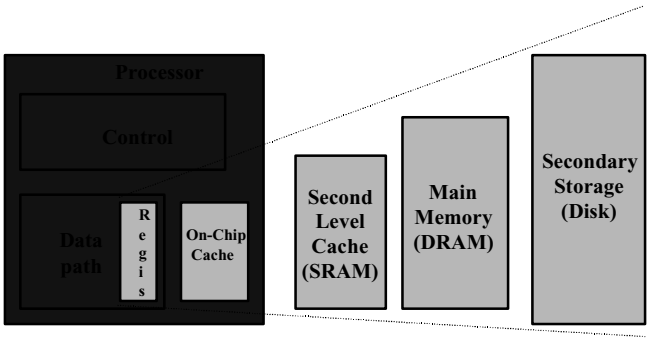


Speicherhierarchie



Speed:	Fastest	Slowest
Size:	Smallest	Biggest
Cost:	Highest	Lowest

Speicherhierarchie ff



Speed:	1 ns	1-3 ns	10 ns	10 ms
Size (bytes):	100	Kilo	Mega	Giga

Fragen

- Wieso ist ein Zugriff auf eine Hauptspeicherzelle langsamer als ein Zugriff auf ein Register ?
- Wie kann verhindert werden, dass ein Rechner durch Hauptspeicherzugriffe zu sehr verlangsamt wird ?
- Wieso stellt man dem Prozessor nicht einfach einige Mbyte Register zur Verfügung ?
- Wie kommen Massendaten in den Arbeitspeicher?

Gliederung

- **4.1** Speicherhierarchie
- **4.2** Arbeitsspeicher (Halbleiterspeicher)
  - Speicherhardware
  - Caches
  - Virtuelle Speicher
- **4.2** Massendaten-Speicher
  - Festplatte und Floppy
  - CDROM/DVD
  - Magnetband

4.2 Arbeitspeicher

- Speicherhardware
- Zwei verschiedene Speicherarten:
  - **Festwertspeicher** (ROM=read only memory)
  - **Schreib-Lese-Speicher** (RAM=random access memory)

Festwertspeicher (1)

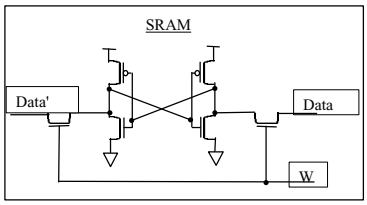
- Nur-Lese-Speicher
- Wird vom Halbleiterhersteller nach Maßgabe des Kunden *irreversibel* programmiert
- EPROM=Erasable Programmable ROM kann durch Einstrahlung von UV-Licht gelöscht und danach *neu* programmiert werden (im Betrieb nur Lesen)

Festwertspeicher (2)

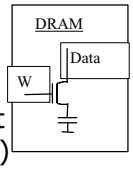
- EEPROM=Electrically Erasable Programmable ROM
  - Löschen geschieht auf *elektrischem* Wege
- weitere Varianten: Flash, NVRAMs
- Festwertspeicher beinhalten oft Initialisierungs- und Startprogramme (z.B. Selbsttest des Systems, Laden des Betriebssystems)

Schreib-Lese-Speicher

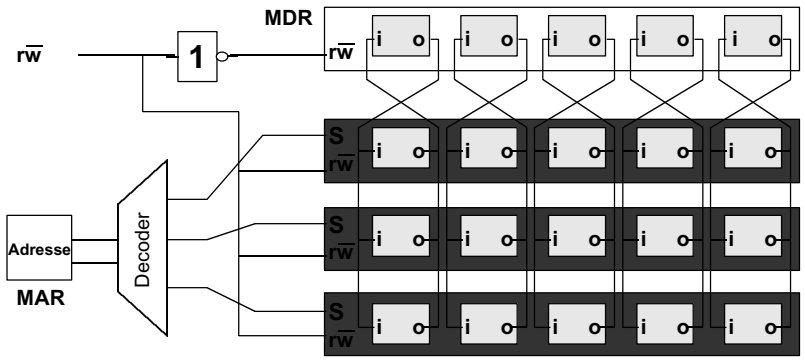
- Halten die Information, solange Versorgungsspannung anliegt
- Statische Speicher (SRAM=static RAM)
  - Speicherzelle aus regulären Flip-Flops
  - 6 Transistoren



- Dynamische Speicher (DRAM=dynamic RAM)
  - 1 Transistor
  - stellt durch Ladung Information dar ($\rightarrow$ Kondensator)
  - refresh ist notwendig
  - Moderne DRAMs haben Kapizität von einigen Mega-Bit (SRAMs und ROMs vergleichbarer Größe gibt es nicht)



Aufbau eines RAMs



Quelle: Gumm/Sommer, 1998

Grund für komplexe Speicherorganisation

- **Wieso ist ein Zugriff auf eine Hauptspeicherzelle langsamer als ein Zugriff auf ein Register ?**
  - Hauptspeicherzellen sind DRAM-Zellen.
    während Register in der Regel SRAM-Zellen sind !
  - Bei einem Registerzugriff kommt man ohne Bus-Operation aus !
- **Wieso stellt man dem Prozessor nicht einfach einige Mbyte Register zur Verfügung ?**
  - SRAM-Zellen sind wesentlich grösser als DRAM-Zellen (Faktor >=4)

  So abwegig ist die Idee nicht !

  Mit der weiteren Technologieentwicklung (noch kleinere Strukturen) wird die verfügbare Chip-Fläche vorwiegend dazu benutzt werden, um schnellen Speicher zu integrieren.

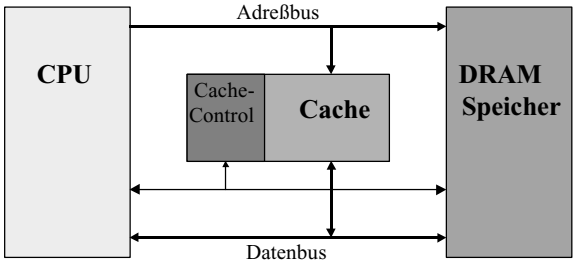
Gliederung

- **4.1** Speicherhierarchie
- **4.2** Arbeitsspeicher (Halbleiterspeicher)
  - Speicherhardware
  - Caches
  - Virtuelle Speicher
- **4.2** Massendaten-Speicher
  - Festplatte und Floppy
  - CDROM/DVD
  - Magnetband

Cache

- Cache kommt aus dem Französischen: **cacher** (verstecken)
- Er kann durch ein Anwendungsprogramm nicht explizit adressiert werden
- Er ist **software-transparent**, d.h. der Benutzer braucht nichts von seiner Existenz zu wissen.

Lage des Caches



In der Regel besitzt ein Rechner einen getrennten Cache für Instruktionen (**Instruktionscache**) und für Daten (**Datencache**)

Ziel des Cache - Einsatzes

- Versuche stets die Daten im Cache zu halten, die als nächstes gebraucht werden
  - ➡ der Prozessor kann die Mehrzahl der Zugriffe auf dem Cache und nicht auf dem langsamen DRAM Speicher ausführen.
- Voraussetzung, um dieses Ziel erreichen zu können: **Lokalitätsprinzip**
  - d.h. zu jedem Zeitpunkt während eines Programmablaufs werden bestimmte Speicherzellen bevorzugt und wiederholt angesprochen (siehe z.B. Schleifen)

Prinzipielle Funktionsweise: Lesezugriff

Lese Datum aus dem Arbeitsspeicher unter Adresse *a*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- Falls ja (**cache hit**)
  - so entnimmt die CPU das Datum aus dem Cache. Die Überprüfung und das eigentliche Lesen aus dem Cache erfolgt in einem Zyklus, ohne einen Wartezyklus einfügen zu müssen.

Prinzipielle Funktionsweise: Lesezugriff

Falls nein (**cache miss**),

- so greift die CPU auf den Arbeitsspeicher zu
- lädt das Datum in den Cache und
- lädt das Datum gleichzeitig in die CPU.
- **Anmerkung**: Insbesondere bei Großrechnern wird mit jedem Datum auch dessen umgebender Block von Daten geladen in der Erwartung, daß folgende Zugriffe auf diese Daten erfolgen (hier aber nicht betrachtet).

Illustration des Lesezugriffs: cache hit

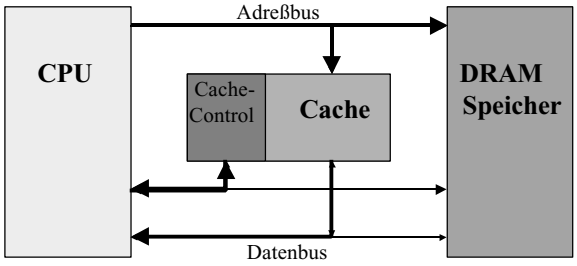
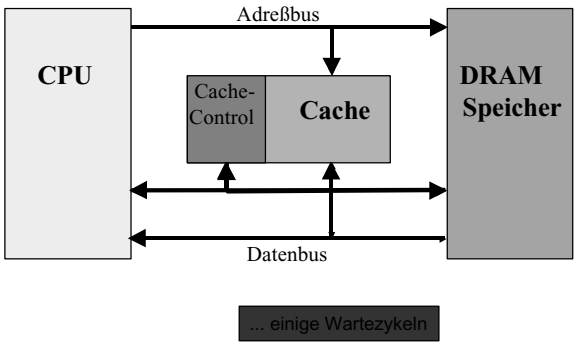


Illustration des Lesezugriffs: cache miss



Mittlere Zugriffszeit beim Lesen

- c : Zugriffszeit des Caches
- m : Zugriffszeit beim Hauptspeicher
- h : Trefferrate

- durchschnittliche Zugriffszeit: c+(1-h)×m

Rechenbeispiel
für c=50 ns und m=200 ns

Trefferrate h	∅–Zugriffszeit
50%	150 ns
60%	130 ns
70%	110 ns
80%	90 ns
90%	70 ns
95%	60 ns

Verdrängen alter Daten aus dem Cache

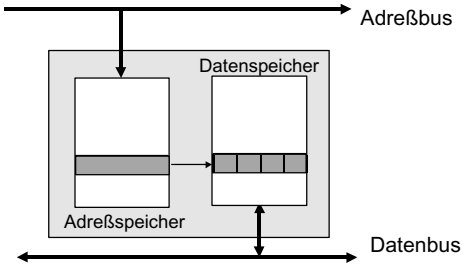
Szenario
- cache miss
- alle Speicherbereiche des Caches belegt

Ausweg
- verdränge Daten aus dem Cache
- lade an diese Stelle die gerade benötigten Daten

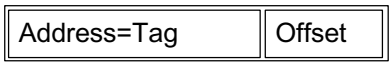
Denkbare Verdrängungsstrategien (später mehr Details)
- least recently used
  verdränge das Datum, das am längsten nicht benutzt wurde
- least frequently used
  verdränge das Datum, auf das am wenigsten zugegriffen wurde
- first in, first out (FIFO)
  verdränge das Datum, das am längsten im Cache ist

Aufbau eines Caches

- Ein Cache besteht aus zwei Speicher-Einheiten, die wortweise einander fest zugeordnet sind.



- Die Daten im Datenspeicher können mehrere Bytes enthalten: Cache line/block



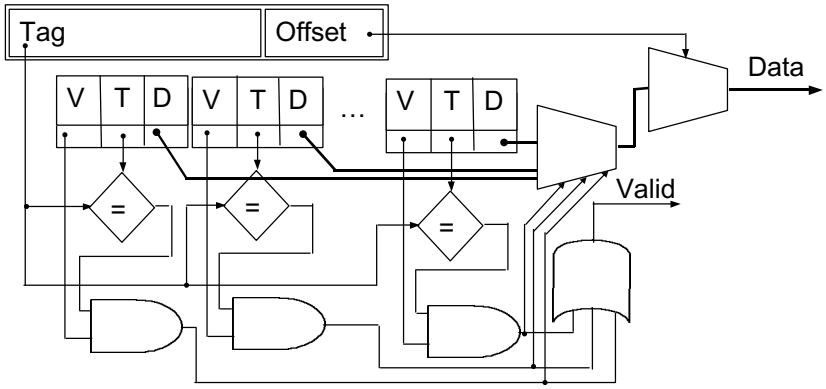
Cache als assoziativer Speicher

- **Idealfall**: assoziativer (inhaltsorientierter) Speicher
- Die von der CPU angelegte Adresse wird **parallel** mit allen im Adreßspeicher des Caches vorhandenen Adressen verglichen.
- Ein neues Datum kann an jeder beliebigen freien Stelle im Cache abgelegt werden.

➡ Aufwendige Logik für den parallelen Vergleich!

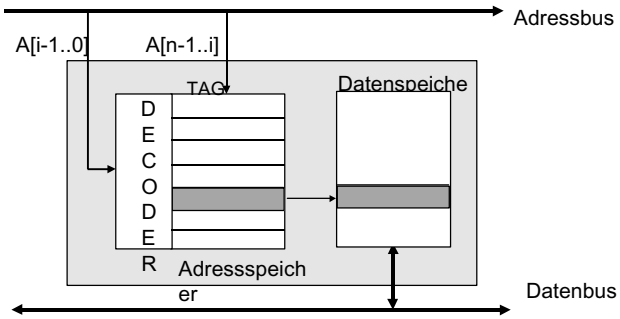
➡ Assoziative Speicher nur für kleine Cache-Größen.

Vollassoziativer Cache Speicher

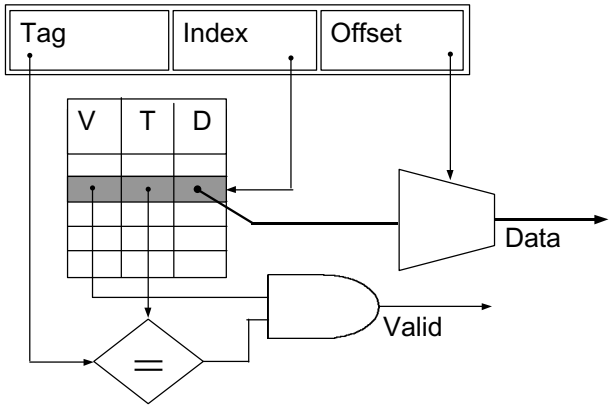


Direct Mapped Cache

- **Feste Abbildung** der Hauptspeicher-Adressen auf die Cache-Adressen
  - ➡ Kein assoziativer Speicher nötig
  - ➡ Keine Verdrängungsstrategien nötig



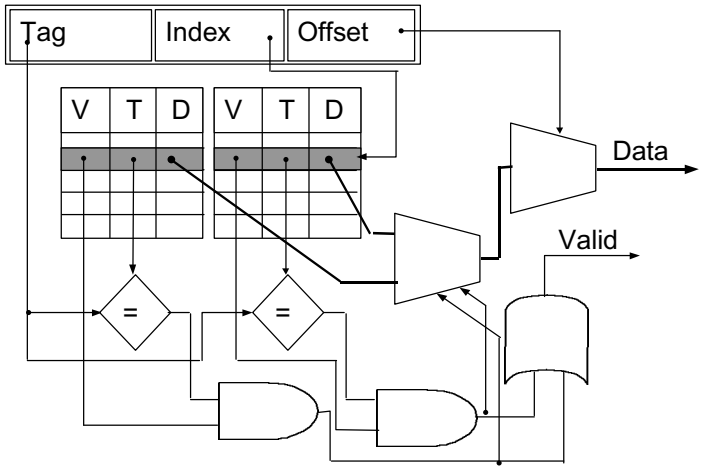
Direct Mapped Cache Speicher



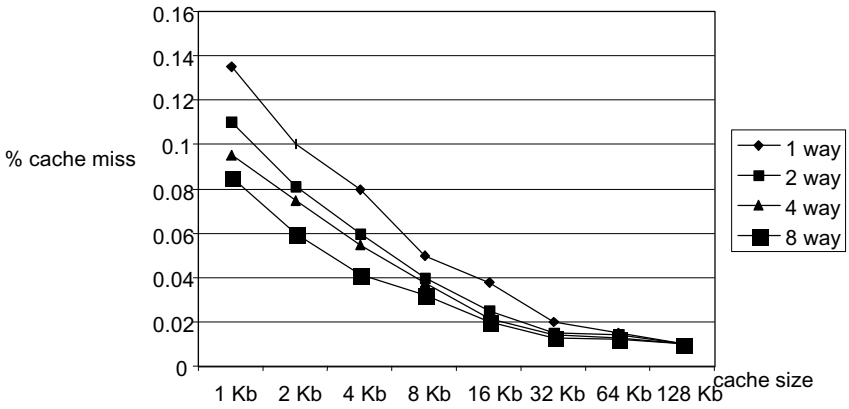
Satzassoziativer Cache

- Kompromiß zwischen direct-mapped und voll assoziativ
- Index wählt einen Satz aus
- jeder Satz enthält mehrere Cache Lines
- Tags in einem Satz werden parallel verglichen
- Cache mit Satz der Größe N heißt N-fach satzassoziativ
  - 2-fach, 4-fach, 8-fach üblich

Satzassoziativer Cache



Cache Performanz



Ursachen für Cache-Misses

- drei Ursachen für Cache-Misses: (Cache-three-C-Problem)
  - compulsory (cold start misses, first reference misses):
    - erster Zugriff zu einem Block: aus Prinzip nicht vermeidbar!
  - capacity
    - Programm braucht mehr Blöcke, als in Cache passen: kann durch größere Caches verbessert werden
  - conflict
    - Misses die dadurch entstehen, daß Stellen, an die Blöcke übertragen werden können, u.U. bereits belegt sind: Erhöhung der Assoziativität

Prinzipielle Funktionsweise: Schreibzugriff (write-through-Verfahren)

Schreibe Datum in den Arbeitsspeicher unter Adresse *a:*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- **write-through Verfahren**:
  - **cache miss:** CPU schreibt das Datum in die Hauptspeicherzelle mit Adresse *a*. Der Inhalt des Cache wird nicht verändert.
  - **cache hit:** die Kopie der Hauptspeicherzelle im Cache wird aktualisiert, die Hauptspeicherzelle selbst wird sofort aktualisiert
  - kombinierbar mit write buffer

Prinzipielle Funktionsweise: Schreibzugriff (write-back-Verfahren)

Schreibe Datum in den Arbeitsspeicher unter Adresse *a:*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- **write-back Verfahren**:
  - **cache miss:** CPU schreibt das Datum in die Hauptspeicherzelle mit Adresse *a*. Der Inhalt des Cache wird nicht verändert.
  - **cache hit:** die Kopie der Hauptspeicherzelle im Cache wird aktualisiert und durch dirty bit als verändert markiert, die Hauptspeicherzelle selbst wird erst später aktualisiert, wenn die Kopie aus dem Cache verdrängt wird
  - kombinierbar mit write buffer

Prinzipielle Funktionsweise: Schreibzugriff (write-allocation-Verf.)

Schreibe Datum in den Arbeitsspeicher unter Adresse a:

CPU überprüft, ob eine Kopie der Hauptspeicherzelle a im Cache abgelegt ist

- **write-allocation Verfahren**:
  - **cache miss**: CPU schreibt Datum in den Cache (markiert mit dirty bit), aber nicht in den Hauptspeicher (dort erst aktualisiert, wenn Kopie aus dem Cache verdrängt).
  - **cache hit**:die Kopie der Hauptspeicherzelle im Cache wird aktualisiert (markiert durch dirty bit), die Hauptspeicherzelle selbst wird erst später aktualisiert, nämlich wenn die Kopie aus dem Cache verdrängt wird

Write-through versus Write-back/Write-alloaction

Vorteile von Write-back/Write-alloaction

- Schreibzugriffe auf Cache bei cache hit ohne Wartezyklus möglich (bei Write-allocation sogar bei cache miss)
- Belastung des Systembusses kleiner, wenn das Rückschreiben in den Hauptspeicher erst nach mehreren Schreibvorgängen erfolgen muß

Nachteile von Write-back/Write-alloaction

Schwierigkeit bei der Datenkonsistenz, wenn andere Komponenten auf den Hauptspeicher zugreifen können, z.B. ein DMA-Controller oder ein zweiter Prozessor in einer Shared Memory Machine. Zu vermeiden ist, dass die anderen Module veraltete Werte vorfinden oder der Cache mit veralteten Werten rechnet

Verwendung von Schreibpuffern

- verkürzt write-back Zeit durch unabhängigen späteren Transfer zwischen Cache und Speicher
- Probleme:
  - Puffer ist evtl. noch nicht übertragen bevor Lesezugriff erfolgt
  - 2 Auswege:
    - bei lesendem Zugriff auf noch nicht eingelagerten Block warten bis Puffer leer
    - Inhaltsvergleich, ob auf gleiche Daten zugegriffen wird
    - Analogie zu Load und Store-Puffern beim dynamischen Scheduling

Out-of-order Laden des Cache

- Idee: nicht warten, bis Block komplett im Cache ist
- Ansätze:
  - *early restart:* beim Nachladen wird das benötigte Wort beim Eintreffen im Cache gleich an CPU geschickt
  - *out-of-order-fetch/wrapped fetch:* es wird zuerst das benötigte Wort geholt, dann der Rest aufgefüllt

2-Ebenen-Caches

- Kompromiß zwischen Geschwindigkeit und Größe
- mittlere Speicherzugriffszeit:

  hit time + miss rate * miss penalty =
  hit $time_{L1}$ + miss $rate_{L1}$ * miss $penalty_{L1}$ =
  hit $time_{L1}$ + miss $rate_{L1}$ * (hit $time_{L2}$ + miss $rate_{L2}$ * miss $penalty_{L2}$)
  für L1: "first-level-cache", L2:"second-level-cache"
- Hauptunterschied von second-level Cache zu first-level cache:
  - Gesamtgröße: es muß keine so große Rücksicht auf Taktfrequenz des Prozessors genommen werden
  - Blockgröße: Argument kleiner Blockanzahl entfällt bei großem L2 Cache

Charakteristika des "second level"-Cache

Block- (Zeilen-) Größe	32 - 256 Bytes
Hit time	4 - 10 Taktzyklen
Miss penalty: Zugriffszeit Übertragungszeit	30 - 80 Taktzyklen: 14 - 18 Taktzyklen 16 - 64 Taktzyklen
Cache-Größe	256 KB - 4 MB

- im allgemeinen gilt die "multilevel-inclusion-property"
  - alle Daten aus L1-Cache sind in L2-Cache

Beispiel: 68040 Prozessor

- 2 unabhängige Caches: für Daten und Instruktionen
- beide sind 4 kB groß
- beide sind 4-fach satzassoziativ, mit 64 Sätzen, mit jeweils 16 Bytes pro Block (64 * 4 * 16 = 4096)
- jede Cacheline hat ein
  - valid Bit
  - und 4 dirty bits (eines für jedes 32-Bitwort)
- der Speicher unterstützt sowohl write-through als auch write-back

Beispiel: Digital Alpha

- 3 on-chip Caches
  - Instruktionen, Daten und externer (2nd level) Cache
  - Einsatz eines 3rd level (off-chip) Caches möglich
- Instruktionscache: 8 kB groß
  - Cacheline ist 32 Bytes lang, direct mapped
- Datencache ist gleich
  - zusätzlich dual-read und single write
  - write-through
    - Write-through mit write buffer
    - dieser umfaßt 6 32-Byte Einträge
    - Hier werden Daten für das Rückschreiben abgelegt
    - write merging : der write buffer wird durch neue Schreibebefehle aktualisiert

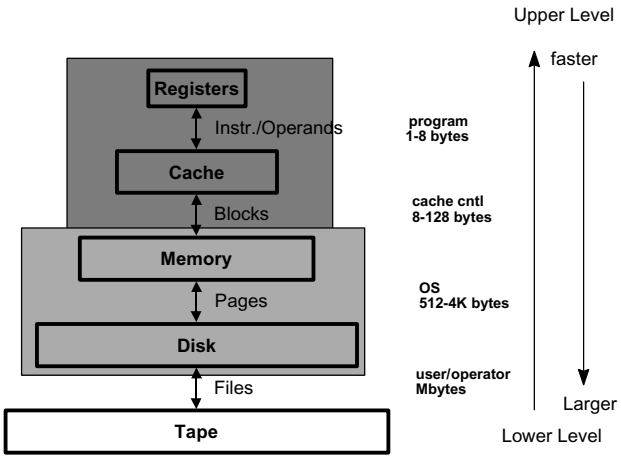
Beispiel: Pentium

- Mehrere Cache Levels (wie Alpha)
- Verschiedene Versionen mit:
  - 8, 16, 32 kB für Daten- und Instruktionscache
  - 256 kB or 512 kB für externen Cache
- Unterstützt write-back und write-through
- Pentium 4:
  - Datencache hat zwei Ebenen:
    - L1 8 kB, 4-fach satzassoziativ, 64 Bytes/Cacheline. write-through (in L2 Cache). 2 Takte Ladezeit
    - L2 256 kB, 8-fach satzassoziativ, 128 Bytes/Cacheline write-back . Lattenz: 7 Takte.
  - Instruktionscache: Trace cache (μOP). 12K μOPs

Gliederung

- **4.1** Speicherhierarchie
- **4.2** Arbeitsspeicher (Halbleiterspeicher)
  - Speicherhardware
  - Caches
  - Virtuelle Speicher
- **4.2** Massendaten-Speicher
  - Festplatte und Floppy
  - CDROM/DVD
  - Magnetband

Virtuelle Speicher



Das Problem mit dem Hauptspeicher

- Adressraum von heutigen Rechnern ist sehr groß !
- Bei Busbreite *n* sind 2^n Speicherzellen adressierbar !
- ... soviel Hauptspeicher kann man nicht bereitstellen !

Busbreite	adressierbare Speicherzellen	Kosten bei 100 DM / 32 MByte
16	65536	--
32	$4{,}3 \cdot 10^9$	13.000 DM
64	$1{,}8 \cdot 10^{19}$	$5 \cdot 10^{13}$ DM

Festplatte als virtuellen Speicher

- **Benutzersicht**
  - Programm und alle Daten befinden sich im Hauptspeicher, sofern der Adressraum nicht ausgeschöpft ist
- **Wirklichkeit**
  - multi-user System, d.h. nicht jedem Benutzer kann der ganze Hauptspeicher zur Verfügung stehen.
  - Programme werden nicht für einen spezifischen 32-Bit Rechner mit maximaler Hauptspeichergrösse geschrieben. Sie sollen auch auf 32-Bit Rechner mit kleinerem Hauptspeicher lauffähig sein.
- **Ausweg**
  - Benutze die Festplatte, um die "Hauptspeicherdaten" eines Prozesses zu speichern, die aufgrund von Kapazitätsgründen nicht im vorhandenen Hauptspeicher liegen können.

Verwaltung des virtuellen Adressraumes

- Jedem Prozess steht ein virtueller Adressraum zur Verfügung
- Diese virtuellen Adressräume werden von dem Betriebssystem verwaltet:
  - Welche Daten bzw. Programmteile werden im Hauptspeicher gehalten ?
  - Welche Daten werden ausgelagert, wenn neue Daten benötigt werden ?
- In diesem Zusammenhang werden die Konzepte
  - **Paging**
  - **Segmentierung**

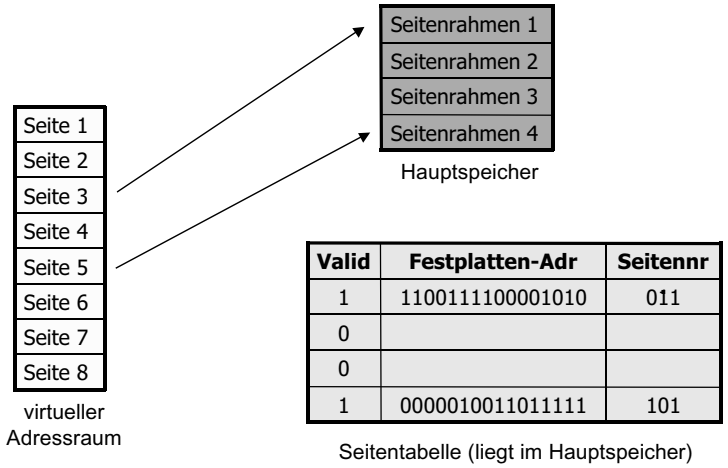
  angewandt

Paging

Grundidee

- Der virtuelle Speicher wird auf dem Sekundärspeicher abgelegt
- ... und in **Seiten** (**pages**) fester Grösse unterteilt.
- Der Hauptspeicher besteht aus **Seitenrahmen** (**page frames**), die jeweils eine Seite aufnehmen können
- Eine **Seitentabelle** (**page table**) gibt an, welche Seitenrahmen durch welche Seiten belegt sind.

Paging ff



Valid	Festplatten-Adr	Seitennr
1	1100111100001010	011
0		
0		
1	0000010011011111	101

Seitentabelle (liegt im Hauptspeicher)

Paging: Zugriff auf Datenseite i

Überprüfe, ob die Datenseite *i* im Hauptspeicher liegt;

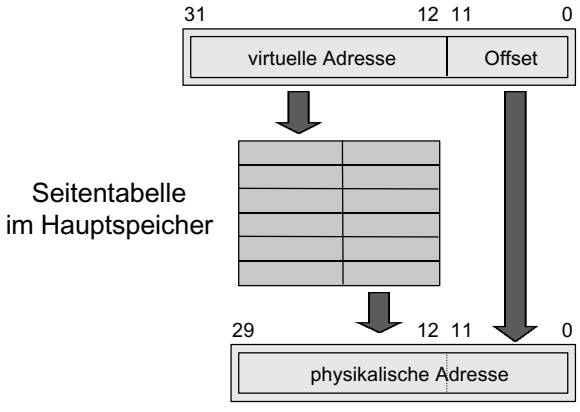
if Seitenfehler
then Überprüfe, ob ein Seitenrahmen im Hauptspeicher leer ist;
    if kein Seitenrahmen leer
    then Verdränge eine Seite aus dem Hauptspeicher und
        aktualisiere die Seitentabelle;
     fi;
    Schreibe die Datenseite *i* in einen freien Seitenrahmen;
    Aktualisiere die Seitentabelle;
fi;

Greife auf Seite *i* im Hauptspeicher zu;

Typische Kenndaten

- Block- (Seiten-) Größe 512-8192 Bytes
- Hit time 1 - 10 Taktzyklen
- Miss penalty (Seitenfehler) 100.000 - 600.000 Taktzyklen:
  - Zugriffszeit: 100.000 - 600.000 Taktzyklen
  - Übertragungszeit: 10.000 - 100.000 Taktzyklen
- Miss rate 0,00001% - 0,001%
- Hauptspeicher-Größe 16 MB - 16.048 MB

Virtuelle Speicheradressierung



- mehrere Speicherzugriffe nötig
- Seitentabellen sehr groß

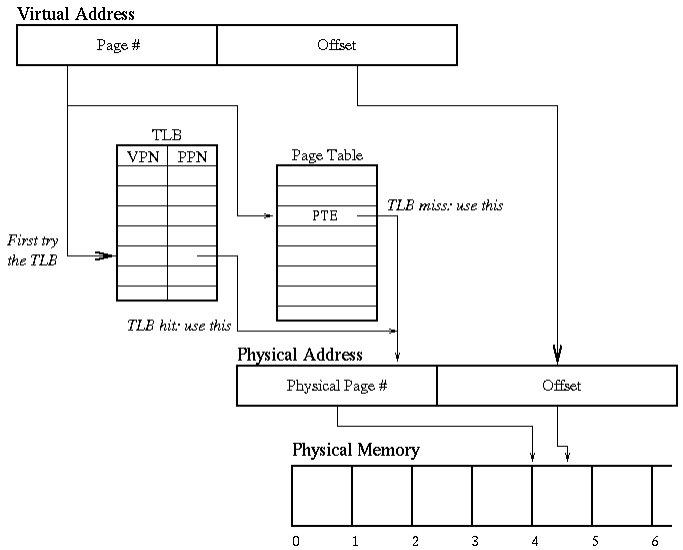
⇒ Hardwareunterstützung

Translation look aside buffer (TLB)

- Hardwareabbildung von virtueller in physikalische Adresse
- die zuletzt berechnete Abbildung von virtuellen auf physikalische Adressen wird hier abgelegt
- vollassoziativer Speicher

Blockgröße:	4 - 8 Bytes (= 1 Eintrag in Seitentabelle)
Hit time:	1 Taktzyklus
Miss penalty:	10 - 30 Taktzyklen
Miss rate:	0,1% - 2%
TLB-Größe:	32 - 8192 Bytes

TLB „Abkürzung“



Seitenersetzungsalgorithmen

- Optimaler Ersetzungsalgorithmus:

> Berechne die Zahl der Instruktionen, die vergehen bis eine bestimmte Seite angesprochen wird. Ersetze immer die Seite mit dem höchsten Wert.

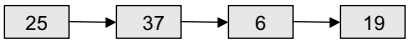
- Dieser Algorithmus ist leicht zu beschreiben, aber unmöglich zu implementieren.
- Implementierbare Algorithmen:
  - Least-Frequently-Used (LFU)
  - First-In-First-Out (FIFO)
  - Least-Recently-Used (LRU)

LFU-Algorithmus

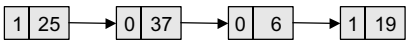
- Jede Seite erhält einen Zähler
  - bei jedem Schreib- oder Lesezugriff wird der Zähler erhöht
- Es wird die Seite entfernt, die den kleinsten Zugriffszählerwert aufweist
- Alter einer Seite wird nicht berücksichtigt

Seitenersetzung nach dem FIFO-Algorithmus

- Das Betriebssystem verwaltet die Seiten in einer Liste. Die jeweils älteste Seite wird entfernt.



- econd-Chance Ersetzung
  - Älteste Seite wird als mögliches Opfer ausgesucht.
  - Ist das R-Bit 0, so wird sie entfernt.
  - Ist das R-Bit 1, so wird es auf 0 gesetzt und die Seite an das Ende der Liste gehängt.



Seitenersetzung nach dem LRU-Algorithmus

- Prinzip ist es, die Seite zu entfernen, die am längsten nicht mehr genutzt wurde.
- Prinzip der Lokalität.
- Implementierungen:
  - Ein Zähler wird mitgeführt, der bei jeder Instruktion erhöht wird. Wird ein Seitenrahmen angesprochen, so wird ihm der Wert des Zählers zugewiesen.
  - Bei einer Maschine mit n Seitenrahmen wird ein nxn-Matrix verwaltet. Wird Rahmen k angesprochen, so werden alle Bits der Zeile k auf 1, alle Bits der Spalte k auf 0 gesetzt. Zeile mit niedrigstem binären Wert repräsentiert die am längsten ungenutzte Seite.

NRU-Algorithmus mit zwei Bits

- NRU = not recently used (Variante von LRU)
- Jede Seite erhält zwei Bits:
  - Referenzbit R
    - Wird bei jedem Schreiben und Lesen der Seite
    - durch die Hardware gesetzt.
  - Modifikationsbit M
    - Wird bei jedem Schreiben in die Seite durch
    - die Hardware gesetzt.
- Initial werden die RM-Bits aller Seiten auf 0 gesetzt. Ferner wird periodisch das R-Bit gelöscht, d.h. auf 0 zurückgesetzt.

Seitenersetzung nach dem LFU-Algorithmus

- Durch Kombination der RM-Bits ergeben sich mehrere Kombinationen:
  - Klasse 0: nicht angesprochen, nicht modifiziert
  - Klasse 1: nicht angesprochen, modifiziert
  - Klasse 2: angesprochen, nicht modifiziert
  - Klasse 3: angesprochen, modifiziert
- Der NRU-Algorithmus wählt zur Ersetzung eine beliebige Seite aus der niedrigsten nicht-leeren Klasse aus.
- Vorteil:
  - leicht verständlich,
  - einfach und effizient zu implementieren.

Segmentierung

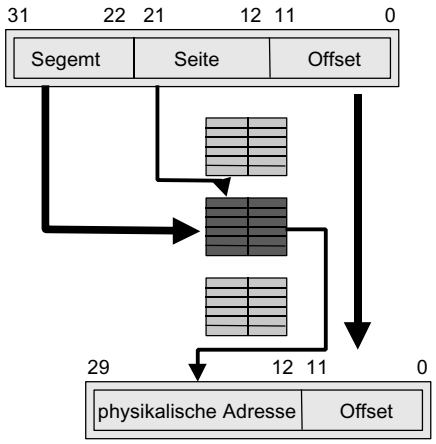
Unterteilung des virtuellen Speichers in Segmente unterschiedlicher Größen

- Zum Beispiel ein Segment für
  - den Programmcode, den Stack, die statischen Variablen ...
  - Für jeden parallel arbeitenden Prozeß
- Schutz durch Zugriffsrechte

- Falls ein angefordertes Segment nicht im Hauptspeicher ist: **segment fault**
- Das Betriebssystem kann dafür Sorge tragen, dass bestimmte Segmente dauernd im Hauptspeicher liegen und nicht verdrängt werden können.

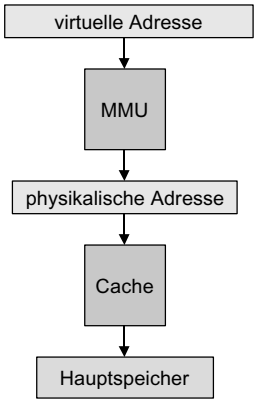
Kombination von Paging und Segmentierung

- Benutze Segmentierung
- Jedes Segment wird für sich mit Paging realisiert

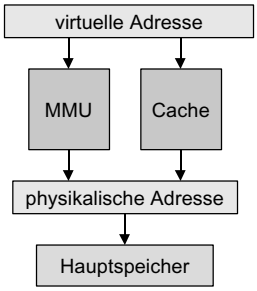


Physikalische/Virtuelle Cache Adressierung

Physikalische Adressierung



Virtuelle Adressierung



Vorteil: Umsetzung und Cachezugriff parallel

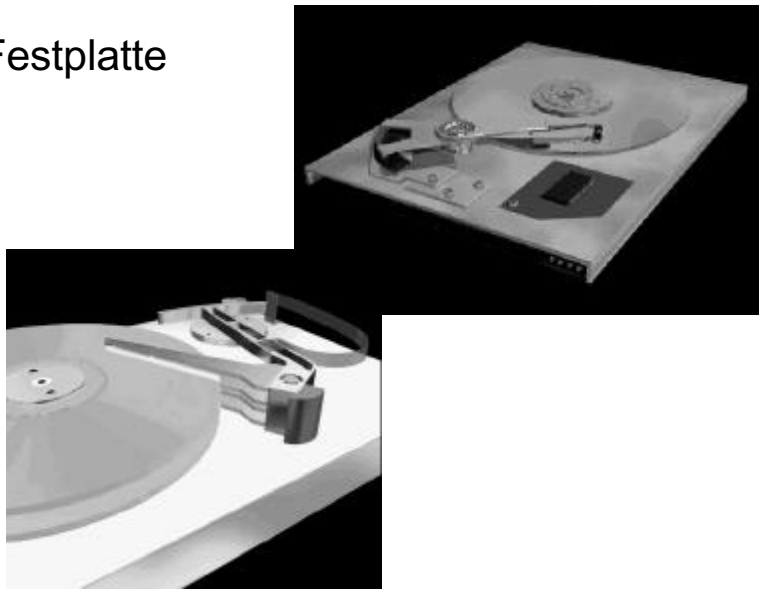
Nachteile der virtuellen Cache-Adressierung

- bei Prozeßwechsel wird Cache ungültig und muß neu geladen werden
  - Verbesserungsmöglichkeit:
    - PID (process identifier)-Tag wird zusätzlich berücksichtigt
    - Segmentierung
- "synonym"-Problem:
  - gleiche virtuelle Adresse (versch. Prozesse) für versch. reale Adr.
    - Lösung: Process-ID hinzufügen (=globaler Adreßraum!)
  - verschiedene virtuelle Adressen in versch. Proz. für gleiche reale
- I/O- und multiprocessing-Konsistenz schwieriger zu implementieren, snooping nicht ausreichend!, reale und virtuelle Adressen werden gebraucht
- reale Cache-Adressierung ist daher üblich (Ausnahme: SPARC)

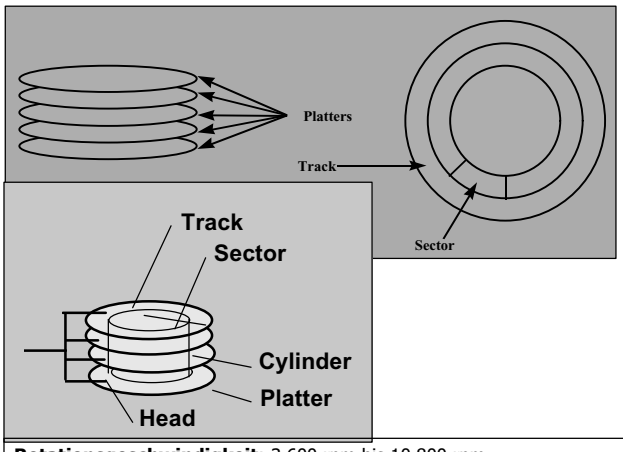
Gliederung

- 4.1 Speicherhierarchie
- 4.2 Arbeitsspeicher (Halbleiterspeicher)
  - Speicherhardware
  - Caches
  - Virtuelle Speicher
- **4.2** Massendaten-Speicher
  - Festplatte und Floppy
  - CDROM/DVD
  - Magnetband

Festplatte



Festplatte (Harddisc)



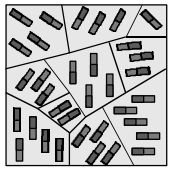
- Eine **Festplatte** besteht aus
  - mehreren Platten (1-8)
  - mehreren Schreib-/ Leseköpfen (2-16)
- Eine **Platte** besteht aus konzentrischen Spuren
- Eine **Spur** besteht aus Sektoren
- Ein **Sektor** ist die kleinste beschreibbare Einheit
- Ein **Zylinder** besteht aus übereinander-liegenden Spuren

Rotationsgeschwindigkeit: 3.600 rpm bis 10.800 rpm
Sektorgröße: 128 byte bis 1 kbyte
Schreibdichte: 50.000 bis 250.000 bits/cm
Spurendichte: 800 bis 20.000 Spuren/cm

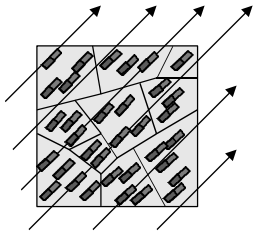
Magnetismus

- Diamagnetismus
  - Abschwächung eines äußeren Megnetfeldes
  - Effekt tritt bei Substanzen ohne ungepaarte Elektronen auf (z.B. Edelgase, Metallsalze)
  - nicht dauerhaft
- Paramegnetismus
  - Verstärkung eines äußeren Megnetfeldes
  - durch Ausrichtung der Atome
  - temperaturabhängig: je tiefer desto stärker
  - nicht dauerhaft

Ferromagnetismus

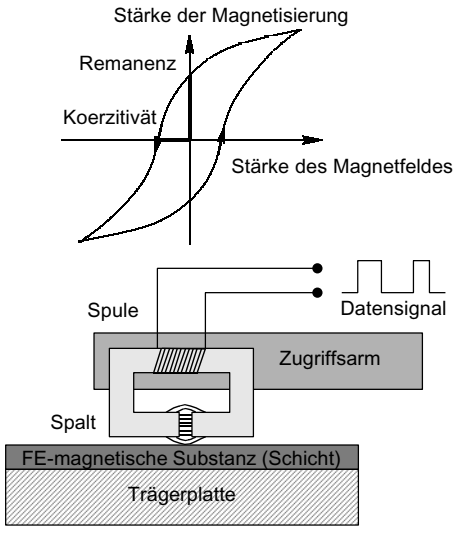


Weißsche Bezirke



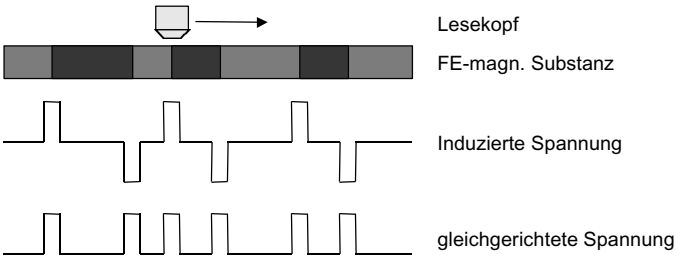
- Verstärkung des externen Magnetfeldes
- Remanenz: Effekt bleibt erhalten (ohne ext. Magnetfeld)
- Koerzitivät: externes Magnetfeld um Remanenz rückgängig zu machen

Hysereseschleife



- Ausnutzen des FE-Magnetismus zum Speichern von Bits
  - FE-magnetische Substanz = Speichermedium
  - ein oder mehrere Weißsche Bezike speichern Information durch Remanenz
  - Anlegen eines Magnetfeldes um Weißsche Bezirke zu megnetisieren, bzw. umzupolen

Lesen durch Induktion



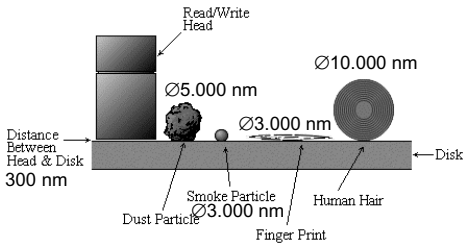
- Jeder Wechsel der Flußdichte (Magnetisierung) erzeugt einen Spannungsimpuls im Lesekopf

Schreiblesekopf

- Ferrit-Köpfe
  - Spule mit Ferritkern
  - Größe nach unten begrenzt
  - schwer
- Dünnfilm-Köpfe
  - Halbleitertechnik
  - leicht und klein
- MR-Köpfe
  - magnetoresistive Köpfe
  - empfindlicher
  - leichter, kleinner
  - höhere Speicherdichte

Schreiblesekopf

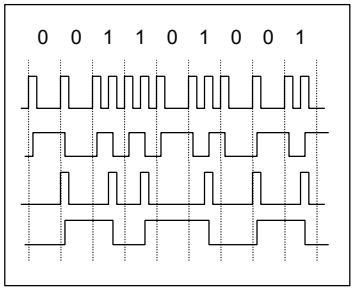
- Schreiblesekopf fliegt durch den Leuftstrom der Rotation getragen über die Platte
  - kleiner Abstand
  - Änderung des Luftdruckes hat Einfluß auf Flughöhe
  - Parkspur
  - Luftstrom wird durch Filter geführt



Kodierung der Daten

Synchronisation durch Taktimpuls

- FM
  - Taktsignal zu Beginn einer Periode
  - Impuls in der Mitte bei 1-Bits
- MFM
  - Taktsignal am Anfang nur wenn in der vorigen und in der aktuellen Periode kein Impuls
  - Impuls in der Mitte bei 1-Bits
  - doppelte Datendichte im Vergleich zu FM bei gleicher Dichte der Flußwechsel
  - kompliziertere Aufzeichnung und Synchronisation

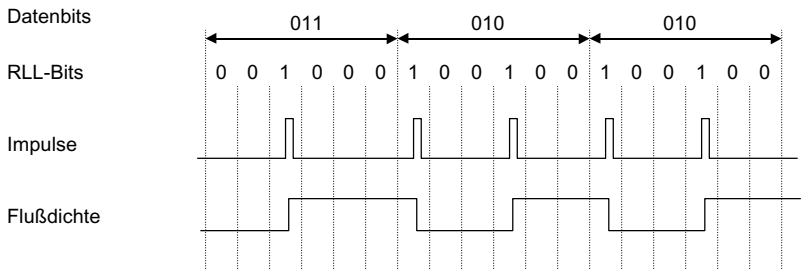


Kodierung durch RLL

- RLL = run length limited
- Anzahl konsekutiver 0-Bits beschränkt
- 1-Bit ist immer umgeben von 0-Bits
- Flußwechsel nur bei 1-Bits
- kein Taktsignal
- Synchronisation durch Laufzeit zwischen 1-Bits
- RLL 2,7-Kodierung
  - minimal 2 0-Bits
  - maximal 7 0-Bits
  - zwischen zwei 1-Bits

Datenbits	RLL 2,7-Code
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
0011	00001000

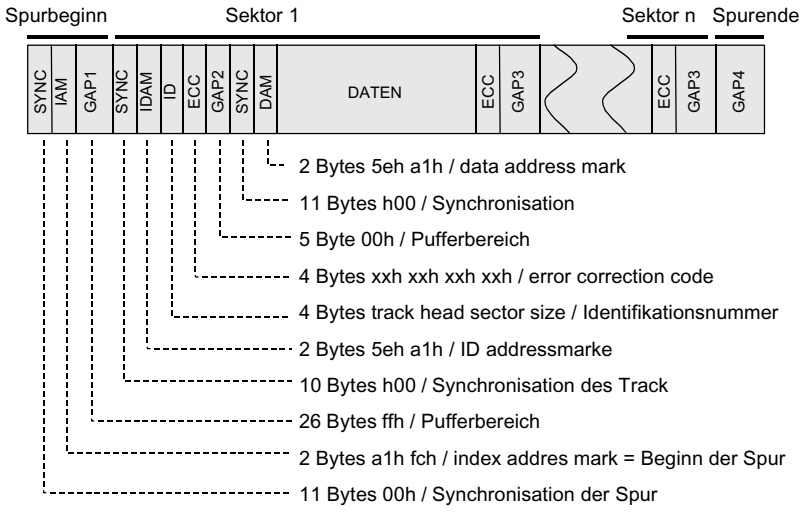
Kodierung durch RLL 2,7



- Aufwendige (De-)Kodierung
- 3x mehr Daten als bei MFM
- Code ist doppelt so lang wie MFM

➔ 50% Mehr Daten bei gleicher Flußwechseldichte

Low-Level Formatierung



Ausführung eines Festplattenzugriffs

- Bewege die Köpfe zu dem richtigen Zylinder (→ Dauer: ca. 10 msec)
- Warte bis der gesuchte Sektor zum Kopf rotiert (→ Dauer: $0.5 \times rpm^{-1}$)
- Übertrage den Inhalt des Sektors (Transferrate: 5-20 Mbyte/sec)

Beispielrechnungen bei Sektorgrösse 512 Byte

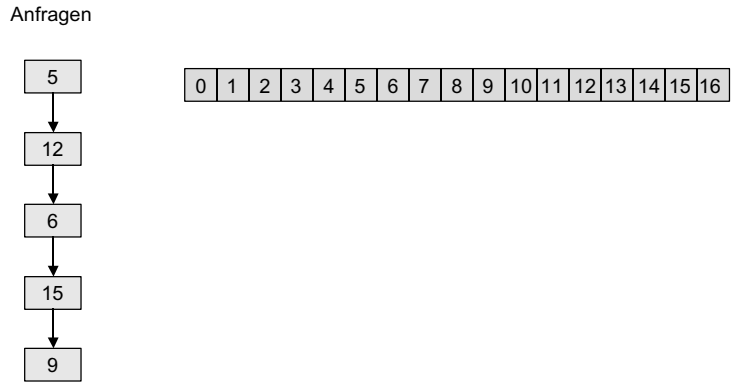
vernachlässigbar

	∅ Positionierungszeit	∅ Latenzzeit	Transferzeit	∅ Gesamtzugriffszeit
3600 rpm, 5 Mb/sec	10 msec	8 msec	102 µsec	18 msec
5400 rpm, 5 Mb/sec	10 msec	6 msec	102 µsec	16 msec
7200 rpm, 5 Mb/sec	10 msec	4 msec	102 µsec	14 msec
10800 rpm, 5 Mb/sec	10 msec	2 msec	102 µsec	12 msec
3600 rpm, 20 Mb/sec	10 msec	8 msec	25 µsec	18 msec
5400 rpm, 20 Mb/sec	10 msec	6 msec	25 µsec	16 msec
7200 rpm, 20 Mb/sec	10 msec	4 msec	25 µsec	14 msec
10800 rpm, 20 Mb/sec	10 msec	2 msec	25 µsec	12 msec

Suchstrategieen bei mehreren Anfragen

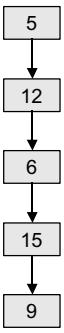
- Wichtig ist, die Suchzeit zu optimieren
- Bekannte Algorithmen:
  - First-Come-First-Served (FCFS)
  - Shortest-Seek-First (SSF)
  - Fahrstuhlalgorithmus

First-Come-First-Served



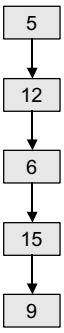
Shortest-Seek-First

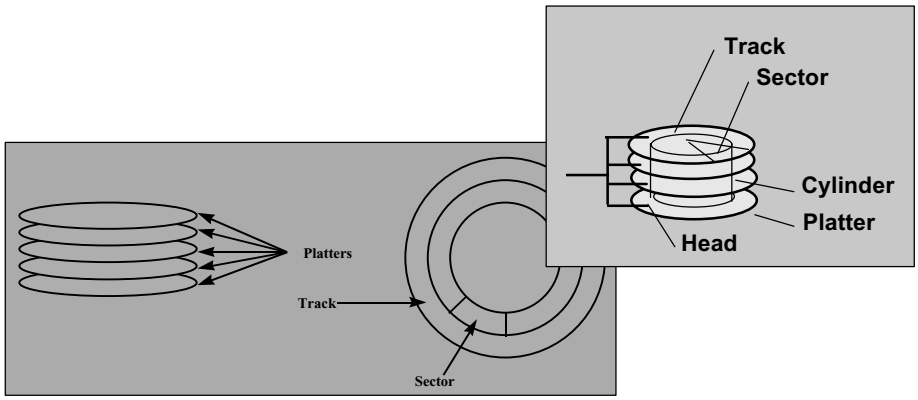
Anfragen



Fahrstuhlalgorithmus

Anfragen





Wie ist eine Festplatte logisch aufgebaut ?
D.h. wie werden die Files auf einer Festplatte abgelegt ?

Logischer Plattenaufbau bei MS-DOS

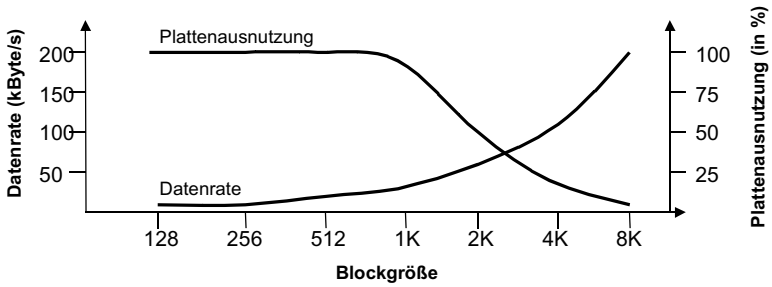
- Eine Platte ist nach dem Formatieren logisch unterteilt in
- den Systembereich bestehend aus
  - dem Urladerbereich (Boot block): Spur 0, Seite 0, Sektor 1 (Spur 0 ist die äussere Spur einer Festplatte)
  - der Dateizuordnungstabelle (File allocation block): Adresse (0,0,2-4)
  - dem Dateiverzeichnis (directory): Adresse (0,0,5-9) und (0,1,1-2)
- den Datenbereich: ab Adresse (0,1,3)

Urladerbereich bei MS-DOS

- **besteht aus einem kurzen Programm (Länge kleiner als 512 byte), das nach dem Einschalten des Rechners den Prozess zum Laden des Betriebssystems in den Arbeitsspeicher aktiviert**

Speicherung von Dateien

- Dateien werden üblicherweise in Blöcken verwaltet.

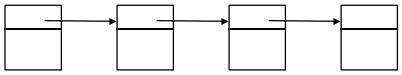


- Typische Blockgrößen sind 512 Bytes, 1k und 2k.
- Ähnlich wie in der Speicherverwaltung werden auch hier die freien Blöcke verwaltet.

Quelle: Tanenbaum, 1990

Speicherung von Dateien

- Dateien benötigen oftmals mehrere Blöcke. Um dies zu ermöglichen bieten sich unterschiedliche Lösungen an:
  - Kontinuierliche Allokation
    - + hohe Performance und leicht implementierbar
    - - Länge muß zur Erzeugung bekannt sein
    - - Platte wird fragmentiert
  - Allokation mittels verknüpfter Liste

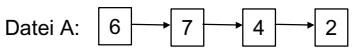


    - + Fragmentierung wird vermieden
    - - Wahlfreier Zugriff sehr langsam
    - - Zeiger benötigt auch ein paar Bytes

File-Allocation-Table (FAT)

- Die genannten Nachteile lassen sich durch eine Kombination von Index und verketteter Liste beheben.
- Lösung lautet FAT (DOS, Windows):

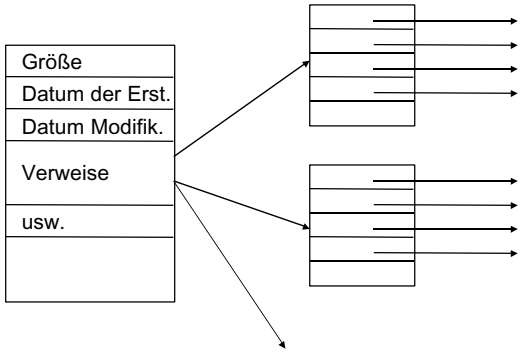
0	
1	
2	**EOF**
3	**free**
4	**2**
5	**bad**
6	**7**
7	**4**
...	



Nachteil: Bei einer 500MB-Platte und 1k-Blöcke benötigt die FAT ca. 2MB im Hauptspeicher!

I-Nodes

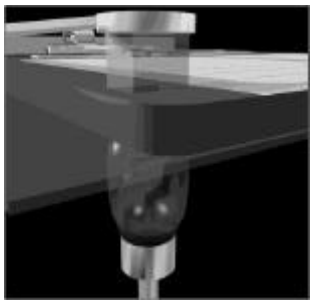
- Eine andere Lösung verwaltet zu jeder Datei eine kleine Tabelle, sog. I-Nodes (Linux, Unix, AIX, Solaris).

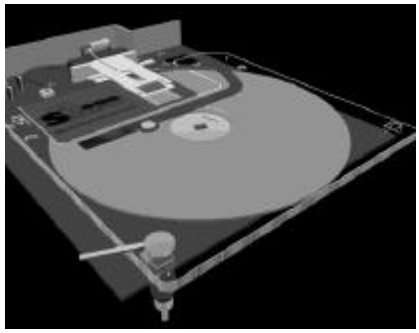


Weitere Sekundärspeicher

- **Floppy**
- **CD-ROM**
- **DVD**
- **Magnetband**

Floppy

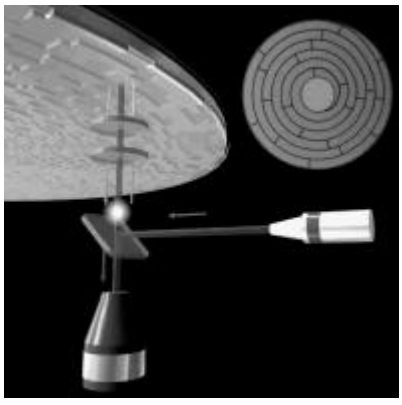




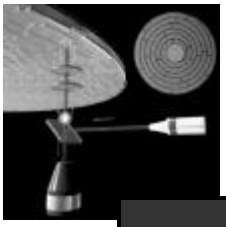
Unterschiede zu einer Festplatte
- Nur 1 Platte, die zudem flexibel
- Lese/Schreibköpfe sind im Laufwerk getrennt von der Diskette untergebracht

- Beim Zugriff fester Kontakt zwischen Lesekopf und Diskettenoberfläche
- Umdrehungsgeschwindigkeit 360 rpm
- Langsamere Zugriffszeit: 150 ms
- Speicherkapazität: bis zu 2 Mbyte
- Sehr viel billiger

CD-ROM / DVD



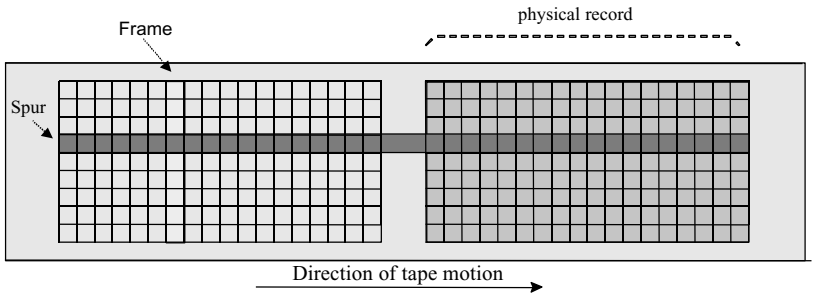
- Daten werden auf einer spiralförmigen Spur abgelegt.
- Beschriebene Oberfläche besteht aus **Gruben** (**pit**) und **Böden** (**land**), über die die Information kodiert ist:
  - Übergang Grube-Grube: 0
  - Übergang Boden-Boden: 0
  - Übergang Grube-Boden: 1
  - Übergang Boden-Grube: 1
- Das Laufwerk besteht aus einem rotem Laser und einem Sensor, der das empfangene Licht auswertet.
  - Gruben reflektieren schwach
  - Böden reflektieren stark



CD-ROM / DVD

	CD-ROM	DVD	Festplatte
Scheiben-Durchmesser	120 mm	120 mm	40-200 mm
Grösse der Gruben	0,80 μm	0,40 μm	0,040 μm
Spur-Abstand	1,60 μm	0,74 μm	0,5 μm
Wellenlänge des Lasers	0,78 μm	0,65 μm	-
Kapazität	650 MByte	17,4 GByte	mehrere GByte
Einfache Übertragungs-geschwindigkeit	150 KByte/sec	1,4 Mbyte/sec	10 Mbyte/sec

Magnetband



- Bandlänge: 700-1000 m
- Schreib / Lesedichte: 1600-6250 bpi (**b**yte **p**er **i**nch)
- Schreibgeschwindigkeit: 10^6 Byte/sec
- Speicherworte werden in Frames abgespeichert bestehend aus
  - 8 Datenbits
  - 1 Paritybit (Überprüfungsbit)
- Information wird byteseriell gelesen / geschrieben
- Die Aufzeichnung erfolgt blockweise