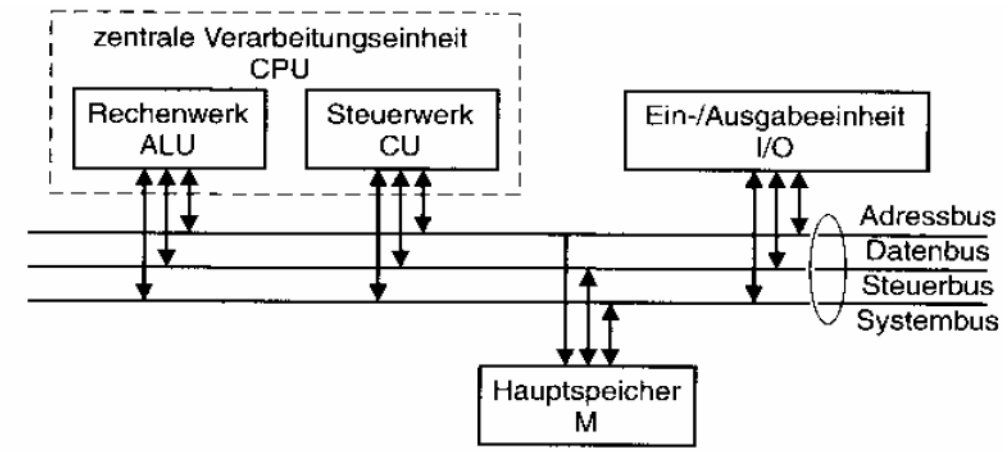
Matthias Jauernig www.linux-related.de
03IN

Zusammenfassung Rechnersysteme

1.) von-Neumann-Architektur/-Rechner

- Grundprinzip: Hardware-Aufbau unabhängig von der Aufgabenstellung/den zu bearbeitenden Problemen ⇨ es liegt eine Universalmaschine vor.
  Nur durch die im Speicher abgelegten Befehlsfolgen und die zu verarbeitenden Informationen können unterschiedliche Verhaltensweisen ermöglicht werden.
- Sequentielle Verarbeitung externer Daten und Befehle (vom Steuerwerk nacheinander aufgerufen).
- Alle Prozesse im Rechner werden prinzipiell von der CPU überwacht ⇨ echter Parallelbetrieb nicht möglich
- Binäre Signalcodierung, Worte fester Länge
  Taktgesteuerte Verarbeitung, automatische Arbeit nach Systemstart

1.1) Komponenten eines von-Neumann-Rechners:



Komponenten des Von-Neumannn-Rechners

- **Speicher**:
  - Aufnahme sowohl von Programmbefehlen als auch von Daten, prinzipiell kein Unterschied zwischen Daten und Befehlen
  - Daten/Befehle binär verschlüsselt, jeder Befehl besteht aus Adresse und Operator
  - Unterteilung in Speicherplätze (Speicherzellen, Speicherworte), die über binäre Adressen angesprochen werden
  - Ansteuerung synchron mit Systemtakt bzw. asynchron per Steuersignal
  - Datenhaltung statisch (SRAM) oder dynamisch (DRAM)
  - Hierarchische Strukturen
  - Leistungsparameter bestimmt durch:
    Zugriffsbreite, Adressierung
    Bandbreite, Kapazität
    Prinzip (elektrisch, mechanisch, optisch)
    Zugriffszeiten für verschiedene Arten (Cache, RAM, Platte)

- **Steuerwerk, Leitwerk** (Befehlsprozessor)
  - Organisiert die Steuerung der gesamten Anlage
  - steuert Programmablauf
    ⇨ Steuerung der Befehlsabfolge, Entschlüsselung der Befehle, Maßnahmen zur Einleitung der Befehlsausführung ⇨ Steuerung/Überwachung der Befehlsausführung
  - Interpretation und Verarbeitung der Befehle und Statussignale;
    Speicher (enthält Befehle) ⇨ Befehlsregister ⇨ Folge von Steuerworten
  - Wichtigster Teil ist die *Ablaufsteuerung*: kann als festverdrahtetes Schaltwerk (RISC) oder mikroprogrammiertes Steuerwerk (CISC) realisiert sein
  - Adressierung der Befehle über Befehlszähler
  - Befehlsaufbau:
    Einwortbefehle:

OpCode	Adressfeld

    Mehrwortbefehle:

OpCode	Adressfeld
Adresse 2. Operand	
Adresse Ergebnis	

  - Befehlsklassen:
    - Speicherbefehle:
      Datentransfer zwischen Speicher/Cache und Prozessor
      Lesen (load):
      PC-Wert/Befehlsadresse bestimmen ⇨ Operanden-Speicheradresse bestimmen ⇨ Speicher lesen ⇨ Inhalt in Register ablegen
      Schreiben (store):
      PC-Wert/Befehlsadresse bestimmen ⇨ Zieloperanden-Speicheradresse bestimmen ⇨ Quellregister lesen ⇨ Speicheradresse und Inhalt ausgeben ⇨ Datum schreiben
    - Verknüpfungsbefehle:
      PC-Wert/Befehlsadresse bestimmen ⇨ Lesen der Operanden aus Register ⇨ Verknüpfung in ALU ⇨ Ergebnis-Eintrag in Register
    - Verzweigungsbefehle:
      PC-Wert/Befehlsadresse für Verzweigung bestimmen ⇨ Prüfen der Verzweigungsbedingung ⇨ Überschreiben des PC-Wertes mit neuem Wert
- **Rechenwerk** (Datenprozessor, ALU)
  - Rechenoperationen mit Operanden aus Speicher (arithmetisch/logisch)
  - verknüpft und verändert zu bearbeitende Daten
  - Operationsumfang:
    - Mathematische Operationen
    - Logische Operationen
    - Bit-Manipulationen, Bit-Verknüpfungen
    - Register-Manipulationen
    - Rotations-/Schiebeoperationen
  - Register enthalten:
    - Operanden aus Speicher
    - Ergebnisse der Operationen
    - Statusanzeigen (Flag-Bits) für besondere Ergebnisse bzw. zur Auswertung im Steuerwerk (z.B. Overflow-, Carry-, Zero-, Negativ-Flag)
  - Steuerwerk und Rechenwerk bilden die zentrale Verarbeitungseinheit (CPU)

- **Ein- und Ausgabe-Einheit** (E/A-Prozessoren)
  - bildet Schnittstelle zur Außenwelt (Peripherie)
    Verbindung Peripheriegeräte mit Prozessor/Speicher
  - i.A. hohe Prozessorauslastung, Ausweg: direkter Speicherzugriff (DMA), separater Prozessor (IOP)
  - Verfahren:
    Befehls-Steuerung: Komm. Prozessor – Peripherie über spezielle Befehle
    Interrupt-Steuerung: über Signalleitungen werden die Anforderungen der I/O-Einheiten an den Prozessor gegeben
    Polling-Steuerung: zyklische Abfrage der externen Geräte

1.2) 2-Phasen-Befehlsabarbeitung (Taktung):

1. *Holephase (Fetch):*
   entsprechend Befehlszähler Befehl aus Hauptspeicher holen,
   Decodierung im Steuerwerk
2. *Ausführungsphase (Execute):*
   Operand aus Hauptspeicher holen,
   Befehl ausführen,
   Resultat in Hauptspeicher schreiben

1.3) Erweiterungen zum von-Neumann-Rechner:

- Trennung von Befehlen und Daten
- Cache, Speicherhierarchie
- Pipeline für Befehle
- Spezialprozessoren für E/A, Grafik, Numerik, …
- Mehrere CPUs bzw. Funktionseinheiten

1.4) Vorteile der von-Neumann-Architektur:

(Dies sind die Hauptgründe für ihre Langlebigkeit)

- Einfachheit (übersichtlich, minimaler HW-Aufwand)
- maximale Flexibilität (bei genügend elementaren Befehlen)
- Prinzip des minimalen Speicheraufwandes

1.5) Nachteile der von-Neumann-Architektur:

- nur ein Prozessor
- nur ein Verbindungsweg zwischen CPU und Speicher (zwischen CPU und Speicher wird immer nur ein Wort transportiert)
- sequentielle Verarbeitung von Befehl und Datum ⇨ v. Neumann-Flaschenhals
  ⇨ Lösung durch getrennte Speicher und Busse
- 1 Befehlszähler, 1 Befehlsregister
  feste Befehlsfolge und Steuerung, ungeeignet für UP-Technik/Multitasking/Multiuser
  ⇨ Lösung durch Universalregister, vereinfachte Steuerung, RISC
- keine Parallelität der Arbeitsschritte
  ⇨ Lösung durch Parallelverarbeitung (Instruction Level Parallelism, Multithreading, Multiprocessor)

Durch neue Rechnerarchitekturen kann man die Nachteile des von-Neumann-Rechners beseitigen, was aber erst durch die stürmische Entwicklung auf dem Hardware-Sektor (sinkende Preise, höhere Integrationsdichte, schnellere Chips, höhere Zuverlässigkeit der Komponenten) möglich wurde. Die neuen Architekturen werden (jedenfalls zur Zeit) den von-Neumann-Rechner nicht ablösen, sondern nur in den Gebieten sinnvoll ergänzen, in

denen sie wirkliche Vorteile bringen. Ein Weg ist z.B. der RISC (Reduced Instruction Set Computer), der durch den vereinfachten Befehlssatz wesentlich schneller arbeitet. Ein Teil der neuen Architekturen ist bereits in kommerziell vertriebenen DVS vorhanden, ein Teil befindet sich im Experimentierstadium und ein letzter Teil ist noch in der Entwicklungsphase.

1.5.1) Die Semantischen Lücken

- von-Neumann-Architektur basiert auf generischer Informationsdarstellung
  ⇨ Objekte des (einzigen) Maschinen-Datentyps des von-Neumann-Rechners sind Bitketten, die im Prinzip jede Art von Information repräsentieren können und nicht dahingehend gekennzeichnet sind, welchen Informationstyp sie im Einzelfalle darstellen.
  Bei Verarbeitung eines Programms werden maschinen-intern nur Bitketten manipuliert, unabhängig davon, ob die aktuelle Anweisung ein Datum oder einen Befehl repräsentiert.
  Von-Neumann-Rechner ist nicht in der Lage, die von der auf ihm implementierten höheren Programmiersprache verwendeten Typen-Attribute für Datenobjekte zu erkennen und zu verarbeiten.
- prinzipiell uneingeschränkte Zugriffsmöglichkeiten des von-Neumann-Rechners auf jedes beliebige Speicherobjekt
  ⇨ dadurch ist der Rechner nicht in der Lage, unerlaubte Zugriffe, die als Folge von Hardware- oder Software-Fehlern auftreten können, zu erkennen und zu verhindern

1.5.2) Das Auftreten von Seiteneffekten

- Uneingeschränkte Zuweisungsmöglichkeit von Werten an Objektnamen, da der von-Neumann-Rechner intern mit typenlosen Bitketten arbeitet ⇨ dies kann zu schwer durchschaubaren Seiteneffekten bei der Programmausführung führen.
  ⇨ Ist nur aufgrund der durch die von-Neumann-Architektur vorgegebenen streng sequentiellen Programmverarbeitung beherrschbar.

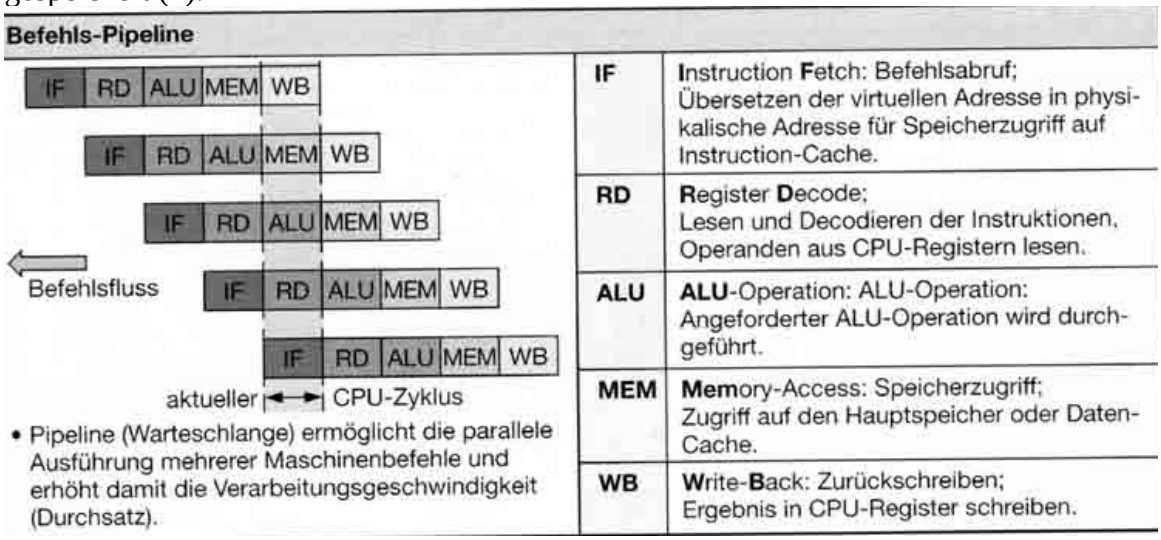
1.5.3) Der von-Neumann-Flaschenhals

- In jedem Rechenschritt kann immer nur der Zustand genau eines Speicherobjektes transformiert werden ⇨ Maschine muss *zuerst einen Befehl* für die auszuführende Operation aus dem Speicher holen, um mit dem Befehl *dann den Datenzugriff* vornehmen zu können.
  ⇨ Für die Verarbeitung einer einzelnen Programmanweisung sind also in der Regel mehrere Zugriffe durch das Rechenwerk auf den Speicher erforderlich, weshalb diese Verarbeitungsweise als der von-Neumann-Flaschenhals bezeichnet wird.

2.) CISC/RISC

2.1) RISC

- RISC steht für „Reduced Instruction Set Computer“
- nach Tabak sind 8 Kriterien zu berücksichtigen
  1. weniger als 50 Maschinenbefehle
  2. weniger als 4 Adressierungsarten
  3. weniger als 4 Befehlsformate
  4. Speicherzugriffe nur über LOAD/STORE – Befehle
  5. mehr als 32 Prozessor-Register
  6. festverdrahtetes Steuerwerk für Maschinenbefehle
  7. Befehls-Pipelining
  8. Optimierende Compiler
- Ziel von RISC: pro Takt sollte ein Maschinenbefehl vollständig ausgeführt werden
- Steht für Mikroprozessoren mit vereinfachter interner Hardware-Organisation und reduziertem Befehlsvorrat (40…200 Befehle)
  ⇨ Beruht auf Beobachtung, dass nur 20% der Maschinenbefehle in 80% der Fälle genutzt werden (ausschl. einfache Maschinenbefehle wie BC, L, CMP, MVI, …)
- Prozessoren kommen mit einem relativ einfachen, z.T. fest verdrahteten Steuerwerk aus, Operationscode liegt also immer an derselben Stelle ⇨ direkte Befehlsdecodierung per Hardware möglich
- Pipelining: ein RISC-Prozessor verarbeitet mehrere Befehle quasi gleichzeitig.
  Da ein Befehl sich in verschiedenen Phasen der Bearbeitung befinden kann, können mehrere Befehle in ihrer jeweiligen Phase vom Prozessor abgearbeitet werden.
  Wird ein Befehl geholt (1), kann der vorherige Befehl dekodiert werden (2). Der Befehl davor wird schon ausgeführt (3) und die Ergebnisse des Befehls davor werden gespeichert (4).



- Optimierende Compiler für höhere Programmiersprachen möglich
- Beispiele: SUN SPARC, DEC Alpha, IBM PPC, HP Precision, Intel i860

2.2) CISC

- CISC steht für „Complex Instruction Set Computer“ und für eine Architektur mit vielen und komplexen Befehlen (umfangreicher Befehlsvorrat, 400-500 Befehle) ⇨ komfortabler zu programmieren durch viele Befehlsformate, benötigt aber Speicherplatz
- Rechnerfamilien möglich aufgrund einheitlicher Befehlssätze für verschiedene Hardware durch Mikroprogramme
- Heute sind die RISC-Prinzipien auch in Prozessoren mit vielen Befehlen aufgenommen worden, so dass die Architekturen immer mehr verschmelzen.
  ⇨ Vor allem bei modernen superskalaren Architekturen.
- CISC-Prozessoren besitzen ein aufwendiges Mikroprogramm-Steuerwerk.
- Beispiele: Intel x86, Pentium, DEC VAX

2.3) Worin unterscheiden sich RISC und CISC besonders?

Eigenschaften	**CISC**	**RISC**
Register	Wenige Register (ca. 20)	Viele Register (bis zu 200) und Registerfenster
Befehlssatz	ca. 300 Befehle und mehr als 50 Befehlstypen	Nur rund 100 meist registerorientierte Befehle (außer LOAD / STORE)
Adressierungsarten	ca. 12 verschiedene	Nur 3 bis 5 Arten und nur LOAD/STORE zum Speicher
Caches	Gemeinsame Caches, aber später auch Getrennte	Getrennte Daten- und Befehlscaches nach Harvard
CPI	1 bis 20, durchschnittlich 4	1 bei Basisoperationen, im Schnitt 1.5
Befehlssteuerung	Mikrocode im Speicher, aber auch hartverdrahtet	Meistens hartverdrahtete Mikroprogramme ohne Mikroprogrammspeicher
Beispielprozessoren	Intel x86, AMD, Cyrix	Sun UltraSparc, PowerPC

2.4) Vor-/Nachteile beider Modelle:

	Vorteile	**Nachteile**
RISC	o Optimiert für Lade-/Speicheroperationen o Arbeit mit Pipelining ⇨ parallele Maschinenbefehlsausführung ⇨ schnellere Befehlsabarbeitung o Großer physikalischer Adressraum o Kein aufwendiges Mikroprogramm-Steuerwerk, direkte Befehls-Decodierung o Bessere Compiler-Optimierung möglich, da wenig Alternativen bei Code-Generierung existieren o Übersichtlicher Befehlsvorrat	o Schlechte Nutzung des Speicherplatzes o es werden mehr Maschinenbefehle als bei CISC für eine bestimmte Aufgabe ausgeführt o deutlich längere Programmtexte o nicht sehr komfortabel zu programmieren
CISC	o komfortabel zu programmieren o effiziente Übersetzung durch Compiler möglich o einfache Fehlerbehandlung o Erleichterung von Assemblerprogrammierung o Kurze Maschinenprogramme o reduzierte Zahl der Speicherzugriffe zum Holen der Befehle	o Komplexe Adressierungsarten sehr aufwendig in Bezug auf Hauptspeicherplatz o Benötigt Speicherplatz durch großen Befehlssatz o Komplexes Mikroprogramm-Steuerwerk benötigt o Unübersichtlicher Befehlsvorrat

3.) Paging

- Problem: Programme benötigen mehr Speicher als zur Verfügung steht
- Unterschiedliche Abhilfemöglichkeiten:
  - Overlay-Struktur (wird durch Paging ersetzt):
    Overlays nach Bedarf ein-/ausladen; 1 Teil im Speicher, die anderen auf Platte
  - Virtuelle Speicherverwaltung:
    Programm mit größerer Speicherforderung als Hauptspeicher möglich ⇨ es werden dazu nur Teile im Hauptspeicher gehalten und temporär ein-/ausgelagert, d.h. nur ein Teil der Daten befindet sich im Hauptspeicher, aber alle Daten sind auf der Festplatte vorhanden; funktioniert nur aufgrund der Lokalität von Programmen

3.1) Funktionsweise des Pagings:

- Virtuelle Speicherverwaltung erfolgt durch eine Kombination aus MMU (zur Adressumsetzung) und Betriebssystem (verwaltet die Seitentabelle)
- Programme erhalten einen virtuellen Adressraum, der ihrer Größe entspricht ⇨ stellt aus Programmierersicht einen zusammen hängenden, linearen Speicher dar
- virtueller Adressraum wird in Blöcke einheitlicher Größe (Pages, Seiten) unterteilt, die im physikalischen Adressraum (Hauptspeicher) auf entsprechende Blöcke (Frames, Seitenrahmen) gleicher Größe abgebildet werden (üblich: 512 Bytes ... 8 KByte)
- ein Rahmen im Hauptspeicher kann den Inhalt einer Seite aufnehmen
- Bei jedem Speicherzugriff wird die virtuelle Adresse in die zugehörige reale Adresse übersetzt ⇨ Adressumsetzung (erfolgt mit Hilfe von Seitentabelle(n)) durch MMU

- Auf virtuellen Speicher soll r/w zugegriffen werden ⇨ Zugriff auf die Seitentabelle
  ⇨ Präsenzbit 1, dann Seite im HS, Programm kann fortgeführt werden
  ⇨ Präsenzbit 0, dann Seite nicht im HS: Fehlseitenunterbrechung (page fault), Programm unterbrochen; Seitenüberwacher holt fehlende Seite aus externem Seitenspeicher, lädt sie in den HS (evtl. mit Auslagerung einer alten Seite, wenn der HS voll ist); Eintrag in Seitentabelle wird abgeändert, Präsenzbit gesetzt; Programm wird mit Wiederholung des Befehls fortgesetzt
- Seitentabelle (enthält so viele Einträge wie Seiten im virtuellen Adressraum vorhanden sind) beinhaltet die Abbildungsvorschrift virtuelle ⇨ reale Adresse; gibt an, welche Seiten zu einer gegebenen Zeit im Hauptspeicher sind; virtuelle Seitennummer wird dabei als Index der Seitentabelle verwendet
- Aufbau eines Seitentabelleneintrags:
  - Cache-Bit (Caching ein/aus)
  - Referenzierungsbit (sobald ein Eintrag der Seite referenziert)
  - Modifikationsbit (falls schreibend zugegriffen wurde)
  - Schutzbit
  - Present-/Absent-Bit (wenn 1, dann ist Seite im HS)
  - Seitennummer

3.2) Möglichkeiten zur Umsetzung der Seitentabellen durch Hardware

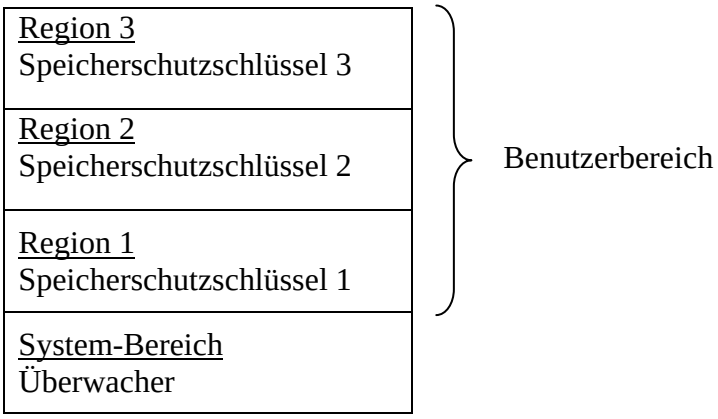
1. Seitentabelle als schnelles Hardware-Register gehalten, adressiert mit virtuellen Seitennummern (schnell, aber teuer); Kontextwechsel bedingt Laden der kompletten Tabelle
2. Seitentabelle komplett im Hauptspeicher gehalten, Register zeigt auf Seitentabelle; keine zusätzliche Hardware, schneller Kontextwechsel, aber: zusätzlicher Speicherzugriff für Adressumwandlung erforderlich
3. Mehrstufige Seitentabellen: hierarchisches System, nur Wurzelverzeichnis im Hauptspeicher

3.3) Demand Paging

- Ist eine Implementierungsform des virtuellen Speichers
- Neue Seiten werden dabei nicht im Voraus, sondern nur auf Anfrage in die Seitenrahmen des Hauptspeichers geladen.
- Mehrere Prozesse haben eigene unabhängige virtuelle Speicher, jeder dieser Speicher kann dabei größer als der eigentliche Hauptspeicher sein.
- Größerer Teil der Seiten eines virtuellen Speichers ist auf einem „externen Seitenspeicher“ ausgelagert, bei Start eines Programms sind alle Seiten ausgelagert
- Beim Zugriff zu eine ausgelagerte Seite (z.B. auf jeden Fall beim Lesen der ersten Instruktionen eines Programms) erfolgt eine „Fehlseitenunterbrechung“

  ⇨ Andere Vorgehensweise ist der Betrieb mit einer „Arbeitsmenge“ (working set), welche auf den Lokalitätseigenschaften von Programmen beruht ⇨ es wird geraten, welche Befehle benötigt werden und diese Arbeitsmenge ist dann in den Hauptspeicher zu laden

3.4) Hauptspeicher-Aufteilung



⇨ Aufteilung des Benutzerbereichs des Hauptspeichers in mehrere Regionen für mehrere parallel ablaufende Programme.

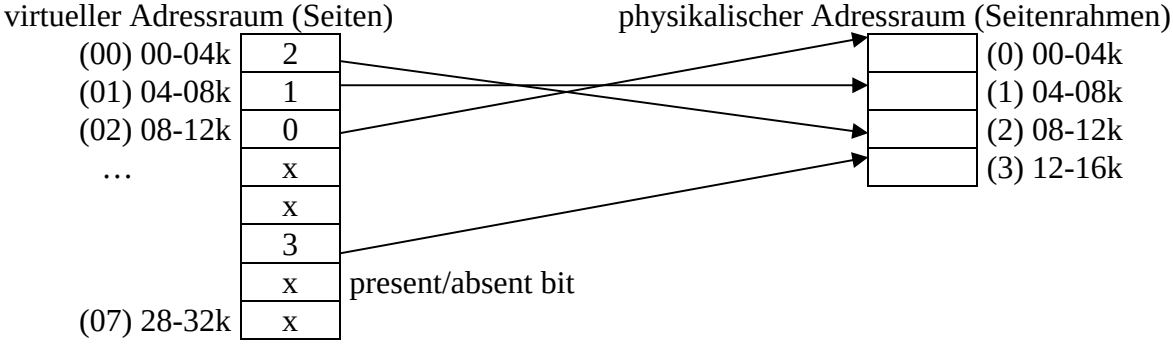
3.5) Probleme der virtuellen Speicherverwaltung

a) Einlagerungszeitpunkt (wann einlagern?):
   - Lösung mit „demand paging“: Einlagerung ausschließlich auf Anforderung, wenn Zugriff zu Seitenfehler führt

b) Zuweisungsproblem:
   - bei Segmentierung muss ausreichend große Lücke vorhanden sein
   - Strategien des Betriebssystems erforderlich

c) Ersetzungsproblem (welche Seite soll ersetzt werden, wenn HS voll?):
   - Verdrängungsstrategien für Seiten erforderlich:
     - FIFO: ersetzt Seite, die sich am längsten im Hauptspeicher befindet
     - LIFO: zuletzt eingelagerte Seite wird ersetzt
     - Second-chance, Uhr-Seitenersetzungsalgorithmus
     - LRU: ersetzt Seite, auf die am Längsten nicht mehr zugegriffen wurde
     - LFU (least frequently used): ersetzt Seite, auf die seit ihrer Einlagerung am seltensten zugegriffen wurde

Beispiel:
Virtueller Speicher: 32k — Anzahl Seiten: 8
physikalischer Speicher: 16k — Anzahl Seitenrahmen: 4
Seiten- / Seitenrahmengröße: 4k



4.) Cache-Speicher

- 2 Cache-Arten:
  - 1.) First-Level-Cache, L1-Cache: in Prozessorbaustein integriert
    → sehr kurze Zugriffszeit (3..4ns, wie Register), aus technologischen Gründen in ihrer Kapazität begrenzt (meist 16KB)
  - Second-Level-Cache, L2-Cache: prozessorexterner Cache
    wird für kürzest mögliche Buszyklen ausgelegt, meist 512KB, Zugriffszeit 10ns
- effizienter Einsatz von Caches nur aufgrund der Lokalitätseigenschaft typischer Programme (siehe Cache-Prinzipien)
- 5-20 mal schneller als der Hauptspeicher, 50 - 1000 mal kleiner als der Hauptspeicher
- heutige Rechner haben Cache-Speicher-Größen bis zu 2 MByte und mehr
- Der Hauptspeicher ist (für den Benutzer unsichtbar) in Blöcke gleicher Größe aufgeteilt (Cache Lines). Diese werden in Block-Rahmen gleicher Größe im Cache-Speicher abgebildet.
- Cache-Eintrag besteht aus 2 Teilen:
  - Adress-Tag (Etikett) – höherwertige Adressbits jedes Blocks: signalisiert ob Blöcke verändert wurden und zurück geschrieben werden müssen; zeigt an, ob der gewünschte Block in der Cache-Zeile vorhanden ist
  - Block: wird durch Bits charakterisiert

4.1) Funktionsweise eines Caches

a) Lesezugriffe:
- Mikroprozessor schaut im Cache nach, ob Datum abgelegt und gültig ist (Valid-Bit);
  Wenn ja: „cache hit“ ⇨ Zugriff auf den Cache wird durchgeführt;
  Wenn nein: „cache miss“ ⇨ Platz in Cache vorhanden? Wenn nicht, dann erst Block aus Cache in HS auslagern, bevor Cache Line vom HS in Cache eingelesen wird und der Cache-Zugriff erfolgt.
- „Cache miss“ nicht so aufwendig wie „Page Fault“, da Steuerung hier meist durch schnelle Hardware erfolgt (bei „Page Fault“ größtenteils durch Software)

b) Schreibzugriffe:
- ist adressierte Speicherstelle nicht im Cache (Miss), wird Datum nur in den Hauptspeicher übertragen
- bei Hit:
  da Cache nur eine Kopie des HS ist und angenommen wird dass die gültigen Daten im HS liegen muss sicher gestellt werden dass Veränderungen in den HS zurück geschrieben werden ⇨ Kohärenzsteerung
  1. *Write-through-Verfahren*: (Durchschreibe-Verfahren)
     Daten werden in Cache und Hauptspeicher modifiziert ⇨ damit ist sicher gestellt, dass Cache und Hauptspeicher immer identische Daten haben; Write-Buffer können die Daten temporär aufnehmen; damit wird die Verzögerung bei Schreibzugriffen klein gehalten
     Vorteil: es stehen immer die richtigen Daten im Hauptspeicher, keine Kohärenzprobleme
     Nachteil: Schreibzugriffe sind langsam ⇨ Leistungsverlust
  2. *Write-in-cache-Verfahren*: (Write-Back-/Rückschreibe-Verfahren)
     bei jedem Schreibzugriff wird ein Steuerbit gesetzt („change bit“/„dirty bit“), welches die Cache-Line als modifiziert markiert ⇨ soll diese später überschrieben werden, so muss ihr Inhalt vorher in den Hauptspeicher zurück geschrieben werden

Vorteil: kein Leistungsabfall
Nachteil: komplexe Daten-Kohärenzsteuerung; erfordert Invalidierung von Cache Lines bei Änderungen im Speicher durch E/A (Snooping)

4.2) Snooping / Datenkohärenz

- Bei einer E/A-Operation (mit DMA) können Hauptspeicher-Daten andere Werte als Cache-Daten annehmen.
  ⇨ Ansteuerlogik des Caches überwacht („snoop“) ständig E/A-Zugriffe zum Hauptspeicher.
- Wenn die Adresse identisch mit der Adresse einer im Cache befindlichen Cache-Line ist, wird erst die Kohärenz der Daten sichergestellt, ehe E/A-Operation zugelassen wird ([modifizierte] Cache-Line wird in Hauptspeicher zurück geschrieben)

4.3) Cache-Strukturen

- Cache-Adressierung ist blockorientiert, Zugriff auf ganzen Block, z.B. 4 Doppelwörter
- Blockgrenzen im Arbeitsspeicher sind durch Vielfache der Blocklänge festgelegt
- Adressierung eines Caches: Abbilden der Hauptspeicheradresse auf einen Adressraum geringeren Umfangs
- verschiedene voll- bzw. teilassoziative Strukturen, die sich im Hardwareaufwand und in der Flexibilität der Positionierung von Blöcken unterscheiden
- 3 wesentliche Cache-Strukturen, abhängig von der Assoziativität zwischen Blöcken im Hauptspeicher und den Block-Rahmen im Cache;
  ⇨ reduzierte Assoziativität bedeutet:
  - pro Zugriff sind weniger Block-Rahmen im Cache zu durchsuchen
  - zusätzliche Einschränkung, wenn Blöcke gleichzeitig im Hauptspeicher untergebracht werden können

4.3.1) Voll-assoziativer Cache

- Heißt, dass ein Block des Hauptspeichers in jedem Block-Rahmen Im Cache abgebildet werden kann.
- Vorteil: hohe Flexibilität
  Nachteil: hoher Hardwareaufwand, aufwendige Blockersetzungsstrategie erforderlich
- Ein Cache-Eintrag besteht aus 2 Teilen: einem Adress-Tag (Etikett) und einem Block. Zu jedem Block im Cache werden dessen höherwertige Adressbits als Tag gespeichert.
  Beim Cache-Zugriff wird die anliegende Adresse mit allen im Cache gespeicherten Tags auf einmal verglichen. Bei Tag-Hit werden Adressbits A3 und A2 zur Doppelwortanwahl im selektierten Rahmen herangezogen. Adressbits A1 und A0 geben benötigte Byteposition an. Pro Rahmen 1 Vergleich notwendig → hoher Hardwareaufwand.
- Hohe Flexibilität, denn jeder Block des Hauptspeichers kann im Unterschied zu den anderen Strukturen in jedem Rahmen des Caches eingelagert werden.
  Blockersetzungsstrategie aufwändig, da bei gefülltem Cache der Rahmen für einen neu zu ladenden Block bei jedem Zugriff neu bestimmt werden muss.
  → LRU – Verfahren (Last Recently Used): es wird der Block überschrieben, dessen Zugriff am weitesten zurück liegt → Cache-Hardware beinhaltet Alterungsinformation
- Realisierungsmöglichkeit für LRU ist Benutzungszähler: Verwendung eines dualen Rückwärtszählers, der bis auf „0“ zählt und bei weiteren Zählimpulsen auf „0“ bleibt; bei Zugriff wird Zähler auf größere Zahl gesetzt („11..11“), alle anderen Zählern werden um „1“ vermindert

→ Bsp.: 4-stufiger Zähler

1. Extremfall: bei jedem Cachezugriff wird ein anderer Block referiert

Zugriff \ Block	0	1	2	3	...	15
1	15					
2	14	15		...		
3	13	14	15		...	
4	12	13	14	15		...
...						
16	0	1	2	3	...	15

→ es gibt einen Zähler der auf „0“ steht

2. Extremfall: es wird immer wieder der selbe Block referenziert → alle Zähler bis auf einen sind 0

Praxis: Zählerstand zwischen beiden Extremfällen

- häufig Einsatz als so genannter „Translation Codeaside Buffer“ (enthält Seitendeskriptoren der aktuell benutzten Seiten)

4.3.2) Direct–mapped–Cache (1-Weg set-assoziativ)

- Heißt, dass ein Block des Hauptspeichers nur in einem Block-Rahmen abgebildet werden kann.
- Vorteil: geringer HW-Aufwand, paralleles Nachladen von z.B. Programmtext und Daten möglich
  Nachteil: durch Abbildung auf die gleiche Adresse des Caches kann es zum „Cache Trashing“ mit stark reduzierter Leistung kommen
- jeder Block des Hauptspeichers wird auf einem bestimmten Rahmen des Caches abgebildet → bei 2n Rahmen werden die n niederwertigsten Bits der Blockadresse als Frame-Index zur Anwahl des Rahmens bestimmt
- durch feste Zuordnung von Block zu Rahmen wird keine Rücksicht auf Cache-Vergangenheit genommen

4.3.3) (n-Way) set-assoziative Caches

- Heißt, dass ein Block des Hauptspeichers in n Block-Rahmen (1<n<c, c…Anzahl der Block-Rahmen im Cache) abgebildet werden kann (Kompromiss aus 1 und 2)
- Kompromiss aus Flexibilität und HW-Aufwand
- Es werden 2, 4 oder mehr Rahmen zu so genannten Sets (s) zusammengefasst; pro Set existieren n Block-Rahmen (n…Assoziativität), womit gilt: Anzahl der Block-Rahmen im Cache c = s*n
- Set-Index adressiert entsprechende Rahmenzahl
- Auswahl des Rahmens durch Ersetzungsstrategie nimmt in kleinem Umfang Rücksicht auf Vergangenheit
- Verwendung einer Abbildungsfunktion f, um alle Hauptspeicherblöcke in Äquivalenzklassen aufzuteilen; Anzahl der Sets ist gleich der Anzahl der Äquivalenzklassen; bei Zugriff auf einen Block x selektiert f(x) ein Set ⇨ jeder Block-Rahmen dieser Gruppe wird getestet, ob er der adressierte Block ist
- Two–Way–Set–associative Caches; Four–Way - ..., allgemein n–Way

4.4) Cache-Prinzipien

- Nicht beliebige, sondern ganz bestimmte Speicherbereiche eines Programms werden zeitweise benutzt ⇨ diese Lokalitätseigenschaften beeinflussen die Wirksamkeit von Cache-Speichern
- *zeitliche Lokalität:*
  bedeutet, dass Programme häufig in Schleifen arbeiten und Daten zu Listen gehören. Deswegen hat Information, welche in der jüngsten Vergangenheit genutzt wurde, eine höhere Wahrscheinlichkeit der Wiederbenutzung als ältere Informationen. Daher wird davon ausgegangen dass der nächste Zugriff auf eine Speicherzelle erfolgt, auf die kurz zuvor bereits zugegriffen wurde.
- *räumliche Lokalität:*
  bedeutet, dass die Maschinenbefehle und Daten häufig Adressen haben, die mehr oder weniger benachbart sind ⇨ nächster Zugriff erfolgt auf eine benachbarte Speicherzelle
- Ein Cache arbeitet mit vorausschauenden Annahmen, welche Adressen in der nahen Zukunft verwendet werden. Diese Vorausschau beruht auf der Annahme, dass Verhaltensmuster der jüngsten Vergangenheit sich fortsetzen werden.

5.) Unterbrechungen

- Def.: eine Unterbrechung ist eine Änderung des Status der Zentraleinheit (ZE) als Folge von Bedingungen, die außerhalb oder innerhalb der ZE auftreten
- Unterbrechungen bewirken Ausführung von speziellen Unterprogrammen außerhalb des normalen Programmablaufs; Befehlsausführung erfolgt in mehreren Schritten
- Unterbrechung/Interrupt ist Befehl wie jeder andere, tritt aber asynchron, zu unvorhersehbaren Zeiten auf
- bei jedem Befehls-Ende Prüfung der Unterbrechungsbedingung; falls erfüllt, Ausführung der Unterbrechung, nicht des nächsten Befehls

5.1) Unterbrechung bewirkt

1. Abspeichern des Programm-Status-Wortes /PSW) und Befehlszählers auf einem Stapel, im Hauptspeicher oder in einem Register
2. Laden des Befehlszählers mit Anfangsadresse einer Unterbrechungsroutine
3. Status-Initialisierung im Status-Wort (z.B. Überwacherstatus setzen)
4. In der Regel: Abspeichern der Mehrzweckregister durch die Unterbrechungsroutine, evtl. auch Steuerregister und Gleitkommaregister

Typischerweise enthält ein Feld im Hauptspeicher zusätzliche Informationen über die Ursache der Unterbrechung.

5.2) Ablauf einer E/A Unterbrechung

Beispiel: Abschluss einer E/A Operation

1. Unterbrechungsleitung aktivieren
2. CPU führt Unterbrechung aus, nachdem die Ausführung des laufenden Befehls abgeschlossen ist
3. Befehlszähler und PSW retten (in spezielles Steuerregister, Stack oder Hauptspeicher)
4. Interrupt Byte einlesen (von CPU Bus, Hauptspeicher oder Spezialregister)
5. Interrupt Byte als Zeiger in Vektortafel benutzen
6. Befehlszähler mit Adresse eines Eintrages der Vektortafel laden
7. PSW laden (1. Befehl in der Vektortafel, Inhalt eines Steuerregisters oder Hauptspeicherwortes oder reset = 0)

8. Start der Unterbrechungsroutine (maskiert gegen weitere Unterbrechungen)
   alten Befehlszähler und altes PSW abspeichern
   Mehrzweckregister retten
   evtl. demaskieren
   …
   IRET

Prioritäten-Reihenfolge:

- kritische Maschinenfehler
- Reset
- E/A, extern
- Programm
- Systemaufruf

6.) Datenspeicherung

- Strukturierung von Daten in Blöcken
- Abspeicherungsprinzipien:
  - *kontinuierliche Abspeicherung:*
    - Datei = kontinuierliche Folge von Blöcken
    - Kennzeichnung durch Lage des ersten Blocks und Gesamtzahl Blöcke

Beispiel:

Datei	Anfang	Ende
A	0	2
B	14	3
C	6	2

Festplatte

A	0	1	2	3	
	4	5	6	7	C
	8	9	10	11	
	12	13	14	15	B
	16	17	18	19	
	. . .				

    - Vorteile: einfach realisierbar, schnelles Lesen von der Platte durch minimale Plattenkopfbewegung, indizierter Zugriff auf i. Block einer Datei möglich (ohne diese sequentiell zu durchlaufen)
    - Nachteile: Fragmentierung, Problem der Festlegung der Dateiposition auf der Platte bei unbekannter Länge der Datei
      ⇨ Ausweg: max. Größe vorher festlegen
  - *verknüpfte Abspeicherung:*
    - jede Datei als Liste von Blöcken
    - erstes Wort jedes Blocks für Verknüpfung benutzt

Beispiel:

Datei	Anfang	Ende
A	9	18

Block	Zeiger
9	16
16	1
1	10
10	18
18	-1

Festplatte

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
. . .			

- Vorteile: keine Fragmentierung, Dateigröße nicht statisch
- Nachteile: Zugriff auf i-ten Block nur sequentiell, Platzbedarf für Verweise, viele Lese-/Schreibkopf-Umsetzungen nötig (kostet Zeit)
- Variante: FAT (File Allocation Table)
  - enthält Eintrag für jeden Block, Blocknummern-Index
  - Blockeintrag enthält Blocknummer des nächsten Blockes
  - Vorteil: Zugriff auf i-ten Block ohne sequentiellen Durchlauf, nur Zugriff auf FAT
  - Nachteil: langsam, wenn FAT nicht in Cache

o *Abspeicherung mit Indizes:*
- Verweise auf nächste Blöcke in Indexblock gesammelt
- Für jede Datei existiert eigener Indexblock fester Größe
- i-ter Eintrag im Indexblock zeigt auf i-ten Datenblock der Datei
- Verzeichnis enthält Verweis auf Indexblock

Name	Index
A	12

12
9
16
1
10
4
-1
-1

Festplatte

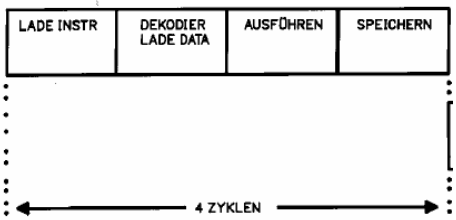
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
. . .			

- Vorteil: kein sequentieller Durchlauf nötig, keine Fragmentierung
- Nachteile: ungenutzter Platz im Indexblock kleiner Dateien, etwas höherer Platzbedarf als bei verknüpften Abspeichern, für jede Datei existiert ein eigener Indexblock

o Speichern großer Dateien: es werden mehrere Indexblöcke benötigt
- Zusammenhängen von Indexblöcken
- mehrstufige Indexblöcke
- kombiniertes Schema

7.) Pipelines

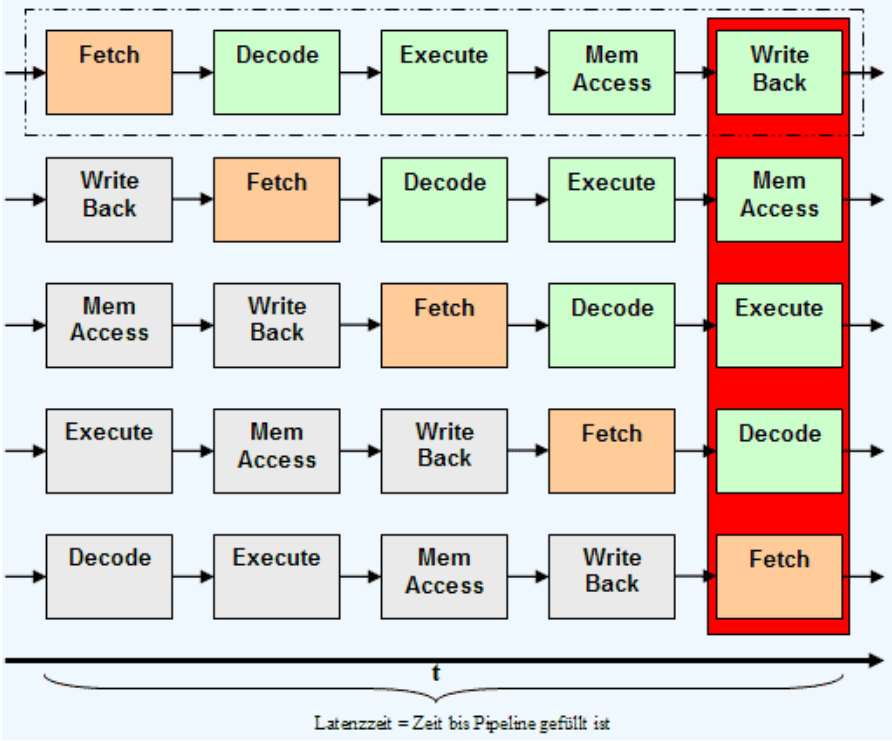
- Pipelining soll es ermöglichen Befehle überlappt auszuführen; dazu ist ein einheitliches Befehlsformat fester Länge Grundlage, weshalb Pipelining nur auf Register getätigt wird. Für Speicheroperationen wird die LOAD/STORE-Philosophie verfolgt, um langsame Hauptspeicherzugriffe zu minimieren.
- Voraussetzung:
  Befehlsverarbeitungsphase muss in mehrere voneinander unabhängige Phasen unterteilbar sein, z.B.:
  - Befehl holen (Instruction Fetch – IF)
  - Befehl dekodieren (Register Decode – RD)
  - Befehl ausführen (Execute – EX)
  - Auf Speicher zugreifen (Memory Access – MEM)
  - Ergebnis in Register schreiben (Write Back – WB)
- Pipeline Operationen:
  - ohne Überlagerung der Ausführungsschritte:



  - mit Überlagerung der Ausführungsschritte:
    Befehlsausführung in mehreren (meist 4 oder 5) Schritten
    Beispiel 4-stufige Pipeline:
    1. Maschinenbefehl aus dem Hauptspeicher einlesen
    2. Befehl dekodieren, Operanden holen
    3. Befehl ausführen
    4. Ergebnis abspeichern

    Verweilzeit eines Befehls in der Zentraleinheit: 4 Maschinenzyklen
    Verarbeitungsgeschwindigkeit: 1 Maschinenzyklus / Befehl
    ⇨ Zeitgewinn um Faktor 4
- komplexere Pipelines in modernen CPUs ⇨ einzelne Stufen wiederum in mehrere sich überlappende Phasen aufgeteilt (Super-Pipeline)

- allgemeiner Aufbau einer (fünfstufigen) Pipeline:
  - Konstruktion so, dass Phasen unabhängig voneinander parallel arbeiten können



  ⇨ Optimalfall: pro Takt ein Befehl fertig (Ziel bei RISC)
- Probleme bei Pipeline-Verarbeitung:
  - Datenfluss-Konflikt:
    - Ergebnis wird in Mehrzweckregister gespeichert, welches gleichzeitig ausgelesen oder verarbeitet werden muss (Ergebnis liegt noch nicht vor, wenn ein nächster Befehl lesend zugreifen will)
    - Lösung: Sprachübersetzer stellt sicher, dass kritische Befehlsfolgen nicht auftreten ⇨ schlägt sich in der Architektur nieder ⇨ Delayed Load
      z.B. Einfügen von NOP Maschinenbefehlen, bis das Ergebnis sicher gelesen werden kann
  - Steuerfluss-Konflikt:
    - bei Sprung- oder Verzweigungsbefehlen steht oft Sprungziel nach der Dekodierung noch nicht fest, weil z.B. Folgebefehl auf Ergebnis eines anderen Befehles warten muss ⇨ Pipeline gerät „außer Tritt“ ⇨ halbfertige Verarbeitung muss evtl. rückgängig gemacht werden ⇨ Leistungsverlust
    - Lösung 1: Delayed Branch: modifizierter Sprungbefehl ⇨ nächster Befehl wird noch ausgeführt, die Verzweigung erst beim übernächsten
    - Lösung 2: Branch Voraussage: statistisches Verfahren, bei dem die Zieladresse eines Sprungbefehls geraten wird ⇨ Voraussage erfolgt mit Hilfe einer „Branch History Tabelle“ (erfordert aber zusätzlichen Aufwand)
    - Pipeline Flush, falls Sprungvorhersage falsch gewesen ⇨ teilweise Verarbeitung von Befehlen in der Pipeline rückgängig machen
- Delayed-Load-Technik:
  - Verzögerung nach einem LOAD Befehl
  - Versuch durch Befehlsumordnungen nach einem LOAD-Befehl einen datenunabhängigen Befehl einzufügen, um den Slot zu füllen

8.) Superskalare Rechner

8.1) Super-Pipeline

- Super-Pipeline bei RISC:
  Erhöhung der Zahl der Pipeline-Stufen ⇨ Erhöhung der Pipeline-Tiefe
  Dazu:
  Aufteilung der Befehlsbearbeitung in mehr als 5 Stufen
  z.B. Aufspaltung zeitintensiver Phasen auf mehrere Takte
  ⇨ Effizienz-Steigerung bisher langsamer Befehle/Stufen

8.2) Superskalare Rechner

- Integration mehrerer ALU auf einem Prozessor (z.B. Integer, Gleitkomma, Speicherzugriffe) für parallele Befehlsverarbeitung, da mit normalen Pipelines nur max. 1 Befehl/Takt möglich
- dadurch gleichzeitiges Ausführen mehrerer Befehle pro Takt
- Probleme:
  - Branches dürfen nicht behindert werden ⇨ Sprungvorhersage
  - Datenabhängigkeiten müssen behandelt werden
  - automatische Erkennung parallel verarbeitbarer Befehle nötig
- u.U. mehrere Pipelines implementiert
- Scoreboard:
  - zusätzliche Steuereinheit mit Verantwortung für Befehls-Aussenden und Erkennen von Konflikten
  - ist zusätzliches Tag-Feld in jedem Mehrzweckregister, welches die Funktionseinheit angibt, die dieses Register gerade benutzt
  - Tests bei jeder Befehlsausführung: functional unit verfügbar, Register verfügbar
  - zählt Lese- und Schreibzugriffe
  - wählt aus Pool potentiell ausführbarer Befehle einen Satz aus

9.) Arbeitsweise CD/DVD

9.1) CD-ROM

- besteht aus einer Folge von Pits (Vertiefungen) und Lands (Erhebungen)
- Übergang von Pit zu Land bildet das Signal
- Lesen erfolgt durch Abtasten der Oberfläche mit einem Laser und Auswertung der Lichtreflexionen (Laser auf Lands fokussiert)
  ⇨ schwaches (gestreutes) Licht entsteht durch Vertiefungen (Pits)
  ⇨ starkes Licht wird durch Lands reflektiert
- Spur verläuft spiralförmig vom Zentrum zum Rand der CD
- Aufzeichnung erfolgt durch Vorbeistreichen der Daten unter einem Laserstrahl mit konstanter Geschwindigkeit
- Zugriffszeit langsamer als bei Festplatte, aber schneller als bei Diskette o.ä.
- CD-R: (WORM – write once read multiply)
  - Ist mit lichtempfindlichem Stoff überzogen, der dieselben Brechungseigenschaften besitzt wie eine normale CD.
  - Schreiblaser erhitzt diese Schicht, sodass sich das Licht hier anders bricht und vom Lesegerät Pits erkannt werden.
- CD-RW: (write multiply)
  - Enthält eine Substanz, die durch Erhitzung amorphe und polykristalline Zustände annehmen kann ⇨ dadurch ist ein Wiederbeschreiben möglich
- CD-MO: (magneto-optical)
  - Wiederbeschreibbar bis 10 Millionen Mal
  - Kapazitäten von 128MB bis 2.6GB
  - Speicherung von Daten beruht auf dem magneto-optischen Effekt: ein Material verändert seine optischen Eigenschaften bei Einwirkung eines Magnetfelds
  - Schreiben: Laserstrahl erwärmt die MO-Schicht über den Kristallisationspunkt hinaus ⇨ nun werden die Teilchen durch ein magnetisches Feld ausgerichtet und kristallisieren wieder
    ⇨ nun ist ein Lesen durch die unterschiedliche Reflexion mit einem Laser geringerer Leistung möglich
  - Nachteile: Schicht kann Blasen bilden, langsames Schreiben, schwere Mechanik

9.2) DVD (digital video/versatile disk)

- DVD-ROM:
  - read-only, ähnlich wie CD-ROM, aber:
    - Datenspuren liegen enger beieinander
    - Minimale Pitlänge nur noch halb so lang wie bei CD-ROM
    - Weniger Verwaltungsdaten
    - Verbesserte Fehlerkorrekturcodes
  - Dazu kommen Laser kürzerer Wellenlänge beim Abtasten zum Einsatz.
  - Mehrere Varianten: Kombination von einseitig, einschichtig und doppelseitig, mehrschichtig
  - Bei mehrschichtig: obere Schichten sind halbdurchlässig, für Zugriff auf untere Schichten ist eine Laser-Fokussierung notwendig
- DVD-RAM, DVD+RW, DVD-RW:
  - Formate für wiederbeschreibbare DVDs mit Phase Change Technologie
- DVD-R, DVD+R:
  - Einmal beschreibbar (WORM - write once read multiply)