Betriebssysteme

Vorlesung im Herbstsemester 2008

Universität Mannheim

Kapitel 2: Hardware-Grundlagen

Prof. Dr. Felix C. Freiling
Lehrstuhl für Praktische Informatik 1
Universität Mannheim

Motivation

- Betriebssysteme sind der Klebstoff zwischen Hardware und den Anwendungsprogrammen
  - Hardware: "nackter" Prozessor, Speicher, Festplatte, Controller
  - Anwendungsprogramme: Textverarbeitung, Datenbanken, Grafikprogramme, Spiele
- In diesem Teil geht es um das, worauf Betriebssysteme aufbauen
  - Was bietet die nackte Rechnerhardware an?
  - Wie kann man sie programmieren?
- Ziel: Einheitliches Verständnis für die angenommene Grundlage für die Konzepte der Vorlesung
  - Für einige ist das eine Wiederholung

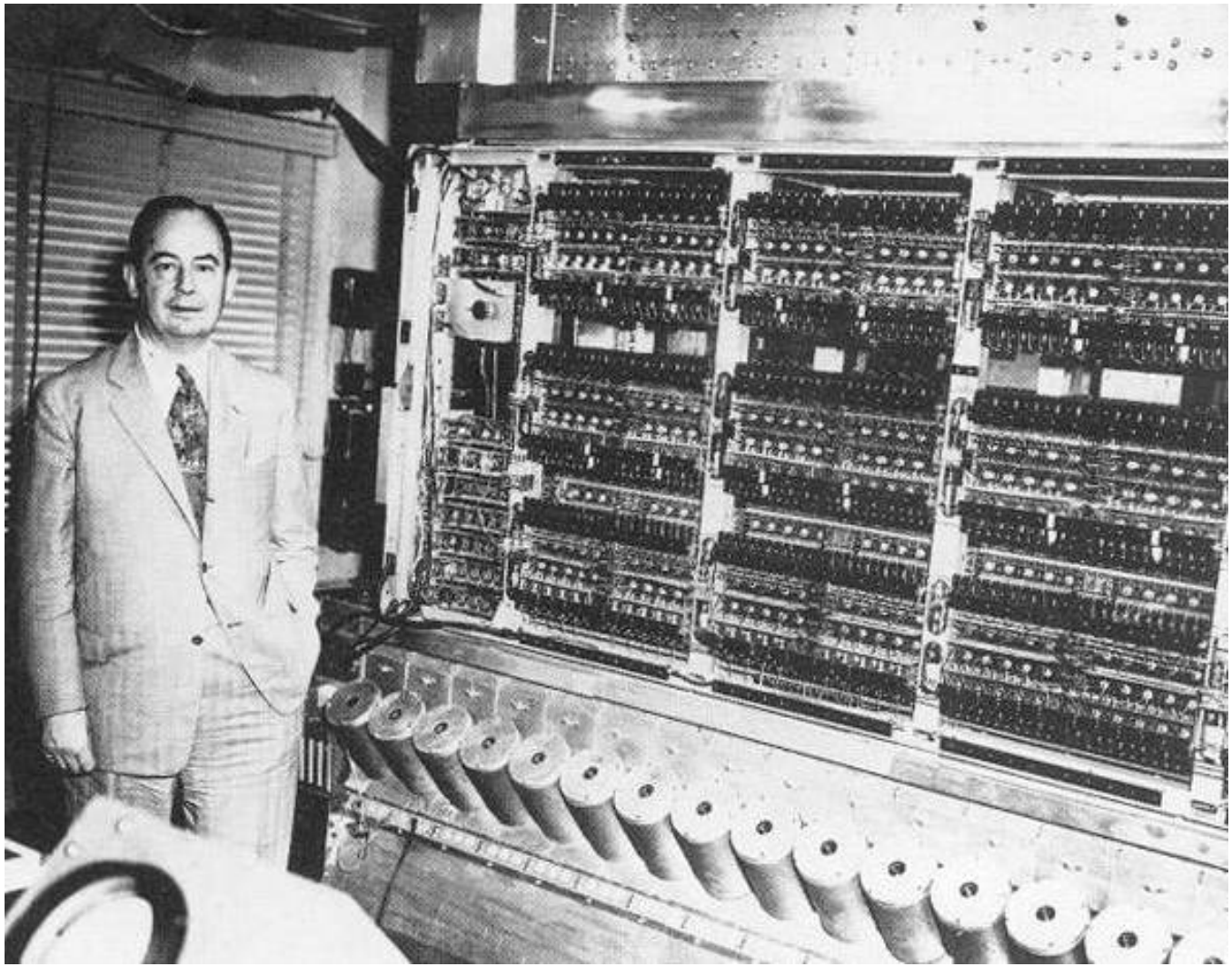
Einordnung in die Vorlesung

- Gliederung der Vorlesung:
  1. **Auf was baut die Systemsoftware auf?**
     Hardware-Grundlagen
     [langweilig für Technische Informatiker]
  2. Was wollen wir eigentlich haben?
     Laufzeitunterstützung aus Anwendersicht
  3. Verwaltung von Speicher: Virtueller Speicher
  4. Verwaltung von Rechenzeit: Virtuelle Prozessoren (Threads)
  5. Synchronisation paralleler Aktivitäten auf dem Rechner
  6. Implementierungsaspekte

Gliederung des Kapitels

- **Einführung (Von Neumann-Architektur)**
- Prozessor
- Speicher
- Ein-/Ausgabe
- Nebenläufigkeit

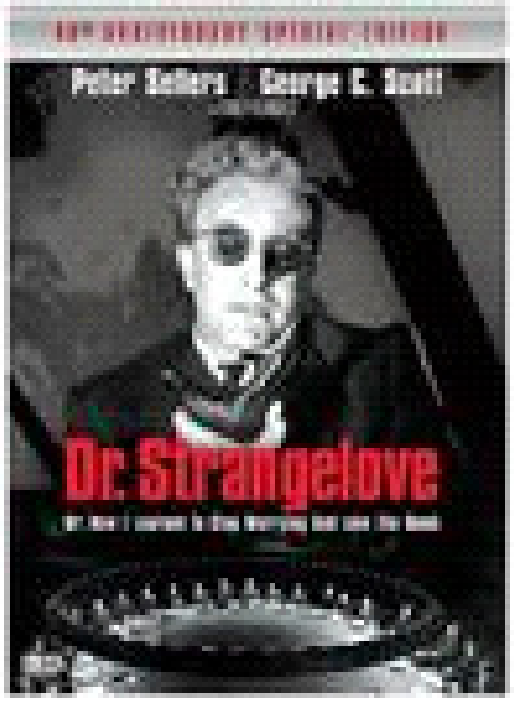
John von Neumann (1903-1957)



John von Neumann

- Eigentlich geboren als János Neumann (1903 in Budapest, gestorben 1957 in Washington, DC)
- schon als Kind ein mathematisches Genie, veröffentlichte mit 17 Jahren seinen ersten mathematischen Artikel
- studierte an verschiedenen Universitäten Europas (u.a. in Deutschland)
- ab 1930 Professor für Mathematik an der Universität Princeton (Institute for Advanced Study), Kollege von Albert Einstein
- emigrierte nach der Machtergreifung der Nazis dauerhaft in die USA
- entwickelte die Von-Neumann-Architektur und beschrieb sie 1945 in einem zunächst unveröffentlichten Bericht
- Ideen der Von-Neumann-Architektur wurden bereits 1936 von Konrad Zuse ausgearbeitet und dokumentiert. Von Neumanns Verdienste liegen wesentlich in der Mathematisierung und Verwissenschaftlichung der Rechenmaschinen
- Von Neumann war am Bau der amerikanischen Atombombe beteiligt.
- Er war ein starker Befürworter exzessiver Rüstungsanstrengungen, um den Ost-West-Konflikt durch mitlitärische Überlegenheit zu entscheiden.

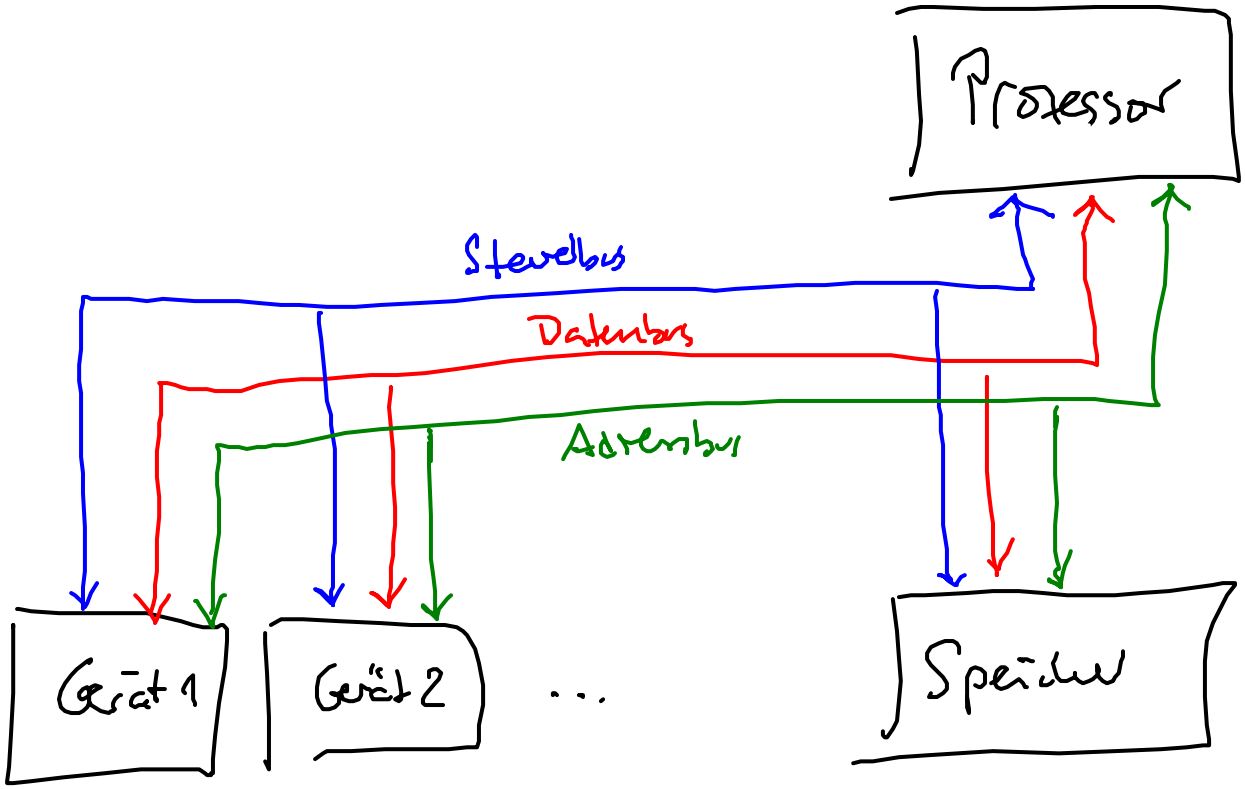
Dr. Strangelove (1964)



- Stanley Kubrick drehte 1964 die Film "Dr. Seltsam - oder wie ich lernte die Bombe zu lieben" (*Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb*)
- Der Film erzählt die Geschichte des "ganz normalen Atomwahnsinns": Ein US-General (Jack D. Ripper) will eine sowjetische Verschwörung aufhalten, indem er eine Atombombe auf die Sowjetunion abwirft
- Was er nicht weiss: Die Sowjets haben eine Weltvernichtungsmaschine gebaut, die durch einen solchen Angriff automatisch ausgelöst wird
- Alle Rettungsversuche scheitern schließlich am defekten Funkgerät des B52-Bombers
- ***Peter Sellers glänzt in 3 Rollen, u.a. als Dr. Seltsam, einem verrückten deutschen Wissenschaftler (Figur teilweise angelehnt an John von Neumann)***

Happy End?!

Von Neumann-Architektur



Von Neumann-Architektur

- Computer besteht aus Prozessor, Speicher und Ein-/Ausgabegeräten (E/A-Geräte)
  - Verbunden über drei digitale Busse: Adreßbus, Steuerbus, Datenbus
- Adreßbus adressiert einzelne Datenzellen des Speichers oder der E/A-Geräte
  - Breite des Busses legt die maximale Anzahl von adressierbaren Zellen fest (Breite des Busses = Anzahl der digitalen Signalleitungen)
    - Beispiel: 32 Bit Busbreite = 2^{32} Zellen (bei einen Byte pro Zelle also 4 GB)
- Steuerbus koordiniert Lese-/Schreibzyklen zwischen den Komponenten
- Datenbus dient der Informationsübertragung
  - Breite des Datenbusses bestimmt, wieviele Einzelzyklen die Übertragung eines Datums bestimmter Länge benötigt

Von Neumann-Prinzip

- Ein Programm besteht aus einer sequentiellen Abfolge von Befehlen
- Alle Befehle befinden sich zusammen mit den zu verarbeitenden Daten im Speicher des Rechners (Von Neumann-Prinzip)
- Der Prozessor liest sequentiell den jeweils nächsten Befehl aus dem Speicher und führt die darin spezifizierte Aktion auf den darin angegebenen Daten aus

Gliederung des Kapitels

- Einführung (Von Neumann-Architektur)
- **Prozessor**
- Speicher
- Ein-/Ausgabe
- Nebenläufigkeit

Der Prozessor

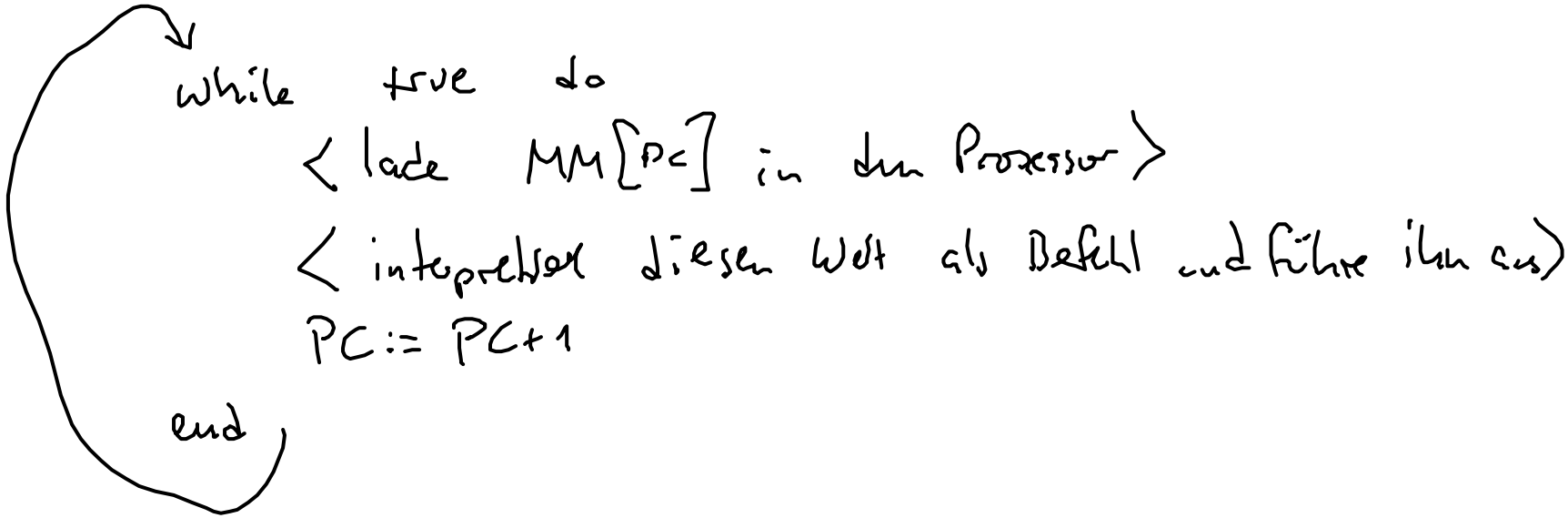
- Der Prozessor ist das Herz jedes Rechnersystems
- Er ist die wesentliche Quelle von Aktivität
  - (Fast) alle anderen Komponenten sind passiv und werden durch den Prozessor gemaßregelt
- Der Prozessor ist ein kleiner Rechner in sich selbst. Er besteht mindestens aus:
  - Registersatz: Eine Menge von speziellen Speicherzellen (Register)
  - Arithmetisch-logische Einheit (ALU): Schaltwerk zur Manipulation von Daten und Adressen (Arithmetik, logische Operationen, etc.)
  - Steuerwerk: Ein Schaltwerk zur Interpretation von Befehlen und Koordination der Aktivitäten in Registersatz und ALU

Spezielle Register

- Im Registersatz gibt es spezielle Register:
  - Programmzähler (*program counter*, PC): enthält die Adresse des nächsten auszuführenden Befehls im Speicher
  - Kellerregister (*stack pointer*, SP): Zeiger auf das obere Element des Laufzeitkellers im Speicher
    - wird zur Umsetzung von Unterprogrammen benutzt
  - Prozessorzustandsregister (*program status word*, PSW): "Rechenzustand" des Prozessors
    - Enthält spezielle Zustands-Bits des Prozessors, die etwas über das Ergebnis der letzten ausgeführten Operation aussagen
    - Beispiel: Ein Bit, das anzeigt, ob das letzte Rechenergebnis aus der ALU den Wert 0 ergab

Der Befehlszyklus

- Der Prozessor geht immer nach dem gleichen Schema vor (fest eingebranntes Verhalten):



- Notation: `MM[x]` = Wert des Hauptspeichers (*main memory*) an der Adresse x

Instruktionssatz

- Die Menge aller vom Steuerwerk verstandenen Befehle definiert den Instruktionssatz des Prozessors
- Es gibt drei Hauptgruppen von Instruktionen:
  - Lade- und Speicheroperationen
  - Arithmetik-, Logik- und Schiebeoperationen
  - Operationen zur Beeinflussung der Ausführungsreihenfolge
- Bei Lade- und Speicheroperationen unterscheidet man verschiedene Adressierungsarten
  - Registeradressierung
  - Absolute Adressierung
  - Relative Adressierung (Spezialfälle sind PC-relative und Kelleradressierung)
  - usw.

Sprungbefehle

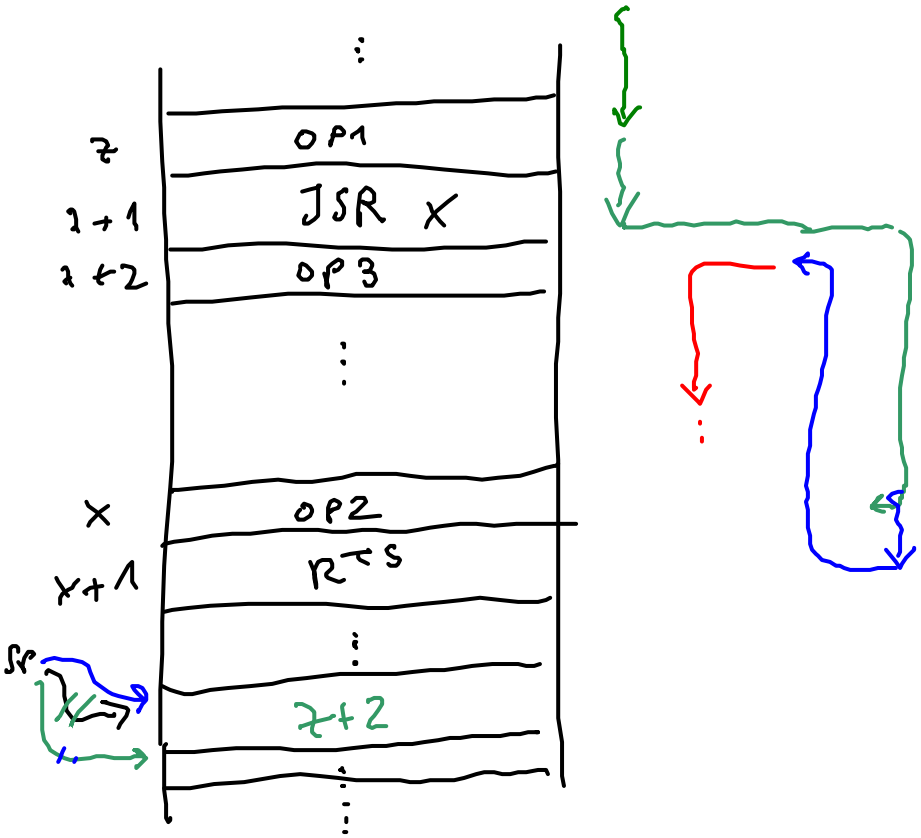
- Normalerweise lädt der Prozessor nach Abarbeitung des aktuellen Befehls den im Speicher direkt folgenden Befehl und führt ihn aus (an der Adresse PC + 1)
- Sequentielle Ausführung kann man durch Manipulation des PC ändern:
  - Sprungbefehle laden einen neuen Wert in den PC
  - Ausführung wird an dieser Stelle fortgeführt (Folge des Befehlszyklus)
- Arten von Sprüngen:
  - bedingt/unbedingt: Sprung abhängig von einem Bit im PSW
  - absolut/relativ: Sprungziel ist (un)abhängig vom aktuellen Wert des PC
    - Wie PC-relative Adressierung

Unterprogramme und Keller

- Prozessoren unterstützen das Konzept von Unterprogrammen
  - Unterprogramm = Sammlung von logisch zusammenhängenden Befehlen
- Die relevanten Befehle sind
  - `JSR` (Jump to Subroutine)
  - `RTS` (Return from Subroutine)
- Befehle benutzen einen Laufzeitkeller
  - Keller = klassische LIFO-Datenstruktur (last-in-first-out), oft Stapel genannt (stack)
  - Zwei Operationen: `push` und `pop`
    - `Push`: etwas auf den Stapel legen
    - `Pop`: das zuletzt auf dem Stapel abgelegte Element holen
  - Nächstes Element des Prozessorstacks ist durch SP adressiert

Funktion von JSR und RTS

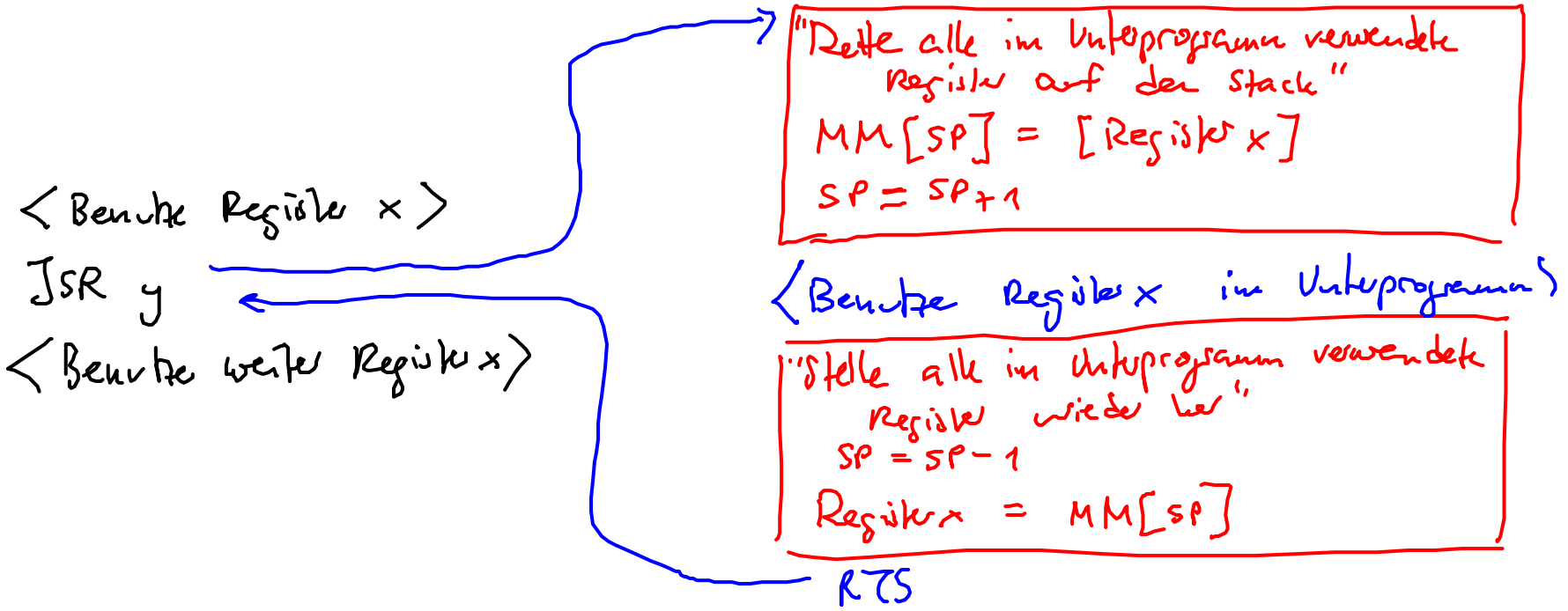
- JSR x
  - MM[SP] := PC + 1
  - SP := SP + 1
  - PC := x
- RTS
  - SP := SP - 1
  - PC := MM[SP]



PC = z
OP1 lade und ausführe
PC = z+1
JSR x lade und ausführe
MM[SP] = z+2
SP = SP + 1
PC = x
OP2 lade und ausführe
PC = x+1
RTS lade und ausführe
SP = SP - 1
PC = MM[SP]
OP3 lade und ausführe
PC = z+3
:

Retten des Prozessorzustandes

- Aufruf von Unterprogrammen sollte Seiteneffektfrei sein
  - Aufrufer soll nach dem Aufruf "unveränderte" Situation (wie vor dem Aufruf) vorfinden
  - Im Unterprogramm verwendete Register müssen gesichert werden

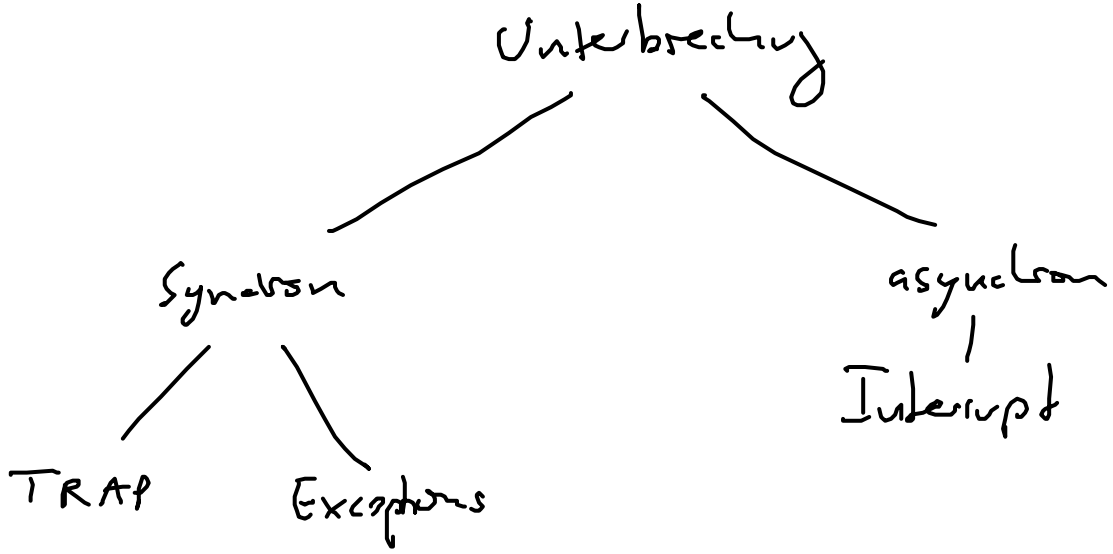


Unterbrechungen

- Die normale Programmausführung des Prozessors kann durch mehrere Arten von Unterbrechungen verändert werden
  - Eine Unterbrechung führt in der Regel zu einem "erzwungenen" Unterprogrammaufruf (Effekt im Prinzip wie beim Befehl `JSR`)
- Verschiedene Arten von Unterbrechungen:
  - Synchrone Unterbrechungen: unmittelbare Folge einer aktuellen Befehlsausführung
    - Expliziter Befehl (TRAP)
    - Implizit (z.B. Division durch 0)
  - Asynchrone Unterbrechungen (Interrupts):
    - Stehen in keinem kausalen Zusammenhang zu einem aktuell ausgeführten Befehl
    - Typischerweise angefordert durch Ein-/Ausgabe-Geräte über spezielle Steuerleitungen

Unterbrechungsformen

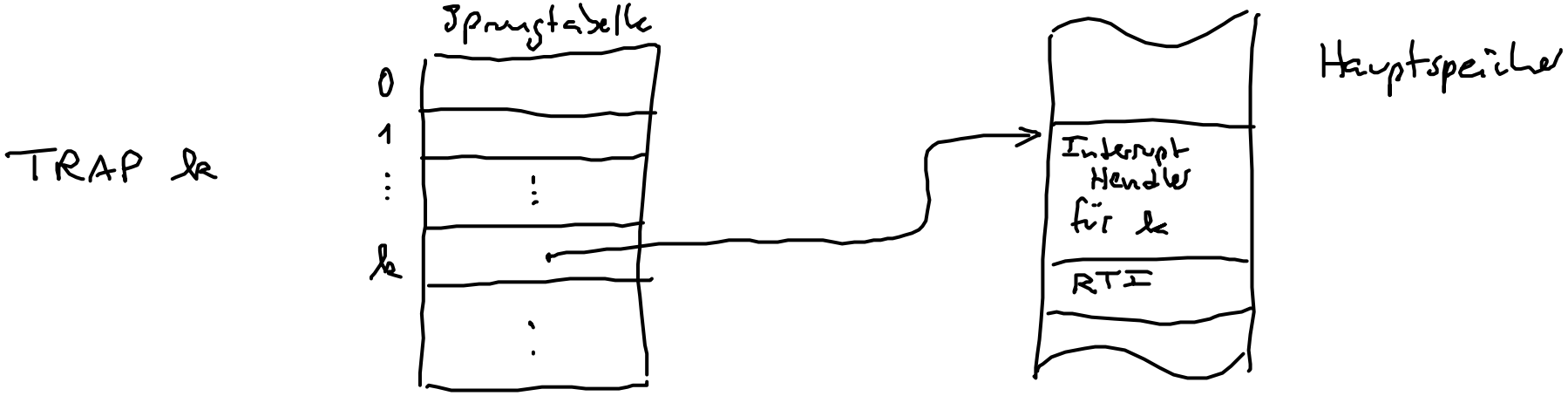
- Übersicht über die verschiedenen Unterbrechungsformen:



- Prozessoren haben in der Regel mehrere getrennte und in ihrer Wichtigkeit gestaffelte Interruptleitungen
  - Interrupts können durch spezielle Befehle maskiert werden (d.h. die Reaktion des Prozessors kann für eine bestimmte Zeit unterbunden werden)

Ablauf bei einer Unterbrechung

- Was passiert bei der Bearbeitung einer Unterbrechung?
  - Jede Unterbrechung hat eine bestimmte Nummer
  - Retten der Rücksprungadresse auf den Stack
  - Indirekter Sprung zu einer Unterbrechungsbehandlungsroutine über eine im Speicher befindliche Sprungtabelle



- Rückkehr in den normalen Programmfluss durch speziellen Befehl (`RTI`, analog zu `RTS`)
  - Seiteneffektfreiheit der Behandlungsroutinen wichtig

Ausführungsmodi

- Prozessoren unterstützen zur Umsetzung von Schutzkonzepten mindestens zwei Ausführungsmodi:
  - Privilegierter Modus (*supervisor mode*): Alle Befehle können ausgeführt werden
  - Normalmodus (*normal mode*): nur ein eingeschränkter Befehlsvorrat kann ausgeführt werden
- Aktueller Ausführungsmodus durch spezielle Bits im PSW angezeigt
  - Ausführung eines privilegierten Befehls im Normalmodus führt zu einer Unterbrechung (Privilegverletzung)
  - Privilegierte Befehle sind Beispielsweise das Maskieren von Interrupts, die Manipulation der Modus-Bits im PSW
- Das Auftreten einer Unterbrechung führt zwangsweise zu einem Umschalten in den privilegierten Modus

Umschalten des Modus

- Spezielle Register wie SP und PSW sind oft für jeden Modus getrennt vorhanden
  - User Stack: SP im Normalmodus
  - Supervisor Stack: SP im privilegierten Modus
- Ein gezieltes Umschalten in den privilegierten Modus ist nur durch den TRAP-Befehl möglich (Rückkehr auch nur durch speziellen Befehl)
  - Mehr dazu in der Übung

Gliederung des Kapitels

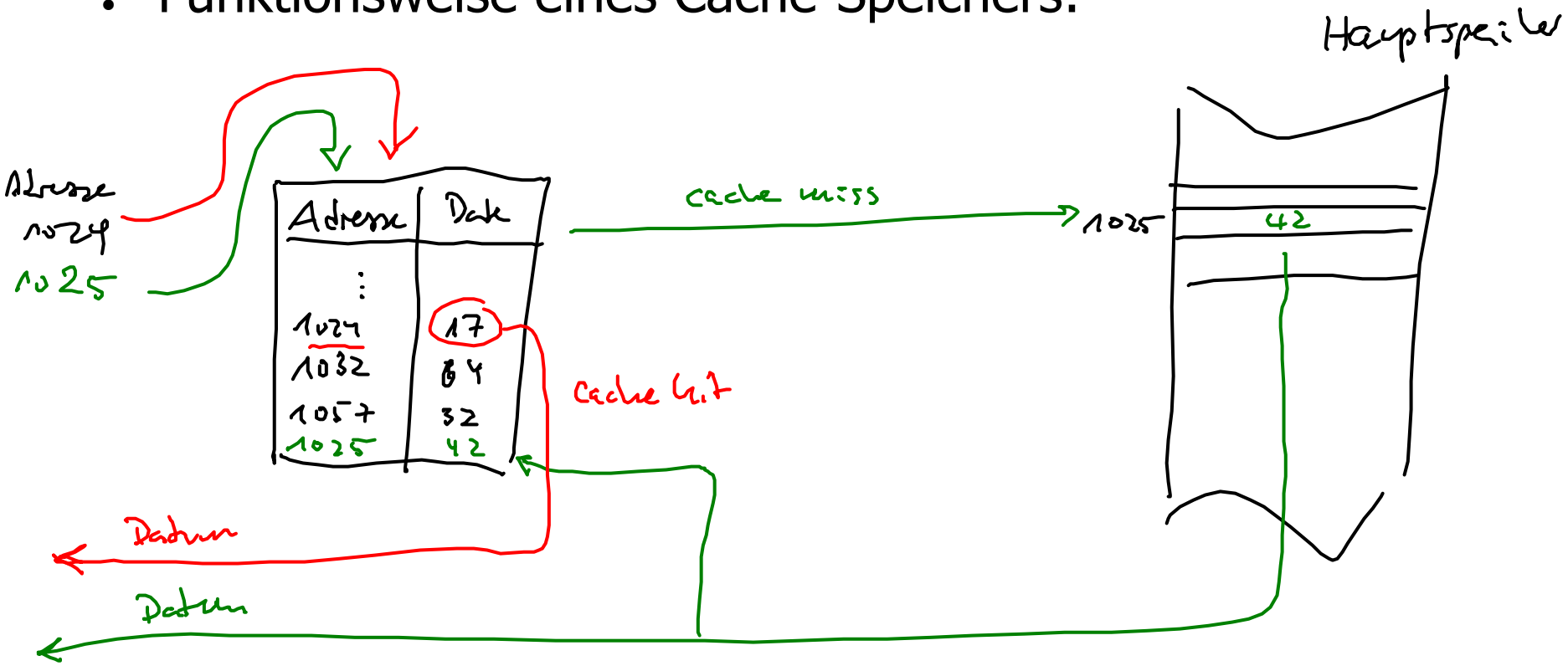
- Einführung (Von Neumann-Architektur)
- Prozessor
- **Speicher**
- Ein-/Ausgabe
- Nebenläufigkeit

Der Speicher

- Enthält die vom Computer ausgeführten Programme und die zugehörigen Daten
- Physischer Adressraum aus Sicht des Prozessors:
  - Menge an physisch ansprechbaren Speicherzellen
  - Menge ist durch die Breite des Adressbusses beschränkt
    - Beispiel: 32 Bit ergeben 4 GB ansprechbaren Speicher
  - Oft nicht voller physischer Speicher ausgebaut
  - Bei Speicherzugriff in nicht-ausgebauten Bereich erzeugt die Hardware eine synchrone Ausnahmebehandlung
- Unterscheidung in RAM und ROM:
  - RAM (Random Access Memory): flüchtige Lese- und Schreibspeicher
  - ROM (Read-Only Memory): nicht-flüchtiger Lesespeicher
- Zugriffszeiten auf den Speicher sind um eine Größenordnung langsamer als die Prozessorgeschwindigkeit

Caches

- Um die Zugriffszeiten auf einen relativ langsamen Speicher (z.B. Hauptspeicher) zu verbessern, bedient man sich heute überall schneller Zwischenspeicher (Cache-Speicher)
- Funktionsweise eines Cache-Speichers:



Erläuterung für Prozessor-Cache

- Der Prozessor legt eine Hauptspeicheradresse an den Cache
- Der Cache prüft assoziativ (d.h. im ganzen Cache parallel), ob diese Adresse im Cache gespeichert ist
  - Ja (Cache-Hit): Datum wird ohne Verzögerung direkt an den Prozessor zurückgeliefert
  - Nein (Cache-Miss): Datum muss von langsamen Externspeicher nachgeladen werden, Cache-Inhalt wird aktualisiert
- Zusammenhängende Fragestellungen:
  - Cache-Verdrängungsstrategien, z.B. der jeweils älteste Eintrag wird verdrängt
  - Cache-Konsistenzproblem: Wenn der Inhalt einer Speicherzelle modifiziert wird, muss man nicht nur den Cache modifizieren
    - Sofortige Aktualisierung: Write Through Cache (langsam)
    - Verzögerte Aktualisierung (z.B. erst bei Verdrängung): Deferred Write (Risiko temporärer Inkonsistenz)

Nutzen von Caches

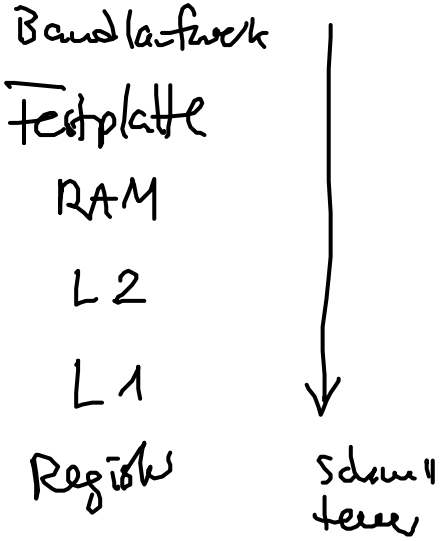
- Nutzen des Caches hängt entscheidend von der Trefferwahrscheinlichkeit p des Caches ab
- Für ein gegebenes p ergibt sich die mittlere Zugriffszeit:

$$t_{average} = p \cdot t_{cache} + (1-p)\, t_{RAM} \qquad t_{RAM} >> t_{cache}$$

  - Annahme: Aktualisierung des Caches bei einem Cache-Miss benötigt keine Zeit
- Schon mit kleinen Caches kann man hohe Trefferwahrscheinlichkeiten erzielen (mehr als 90%)
  - Referenzlokalität von Programmen aufgrund des sequentiellen Verarbeitungsmodells
  - Verbesserung durch Prefetch (voreiliges Laden von Daten, die wahrscheinlich demnächst benötigt werden)
  - Trefferate = Cache-Temperatur (kalter, warmer, heisser Cache)

Speicherhierarchie

- Caches kann man nach Größe und Zugriffszeit auch kaskadieren
  - Auf dem Chip: schneller und teurer Cache (Level 1 Cache)
  - Häufig auch noch unterteilt in Instruktions- und Datencache
  - Auf dem Mainboard: etwas langsamerer und billigerer Cache (Level 2 Cache)
- Idee kann zu einer globalen Speicherpyramide ausgebaut werden
  - Wir werden noch den Hauptspeicher als Cache für Festplattenblöcke kennenlernen
  - Festplatte kann auch als Cache für Bandlaufwerke angesehen werden



Gliederung des Kapitels

- Einführung (Von Neumann-Architektur)
- Prozessor
- Speicher
- **Ein-/Ausgabe**
- Nebenläufigkeit
- ~~Eine abstrakte Prozessorarchitektur~~

Ein-/Ausgabe

- Ein- und Ausgabegeräte erweitern die Fähigkeiten des Computers und erlauben Kommunikation mit seiner Umgebung
  - Bediengeräte: Tastatur, Maus, Joystick, etc.
  - Externe Speicher: Festplatten, CD-ROM, Bandlaufwerke
  - Netzadapter: Erlauben Anschluss des Rechners an ein Kommunikationsnetzwerk
  - und vieles andere mehr (Sensoren, D/A-Wandler, Soundkarten, ...)
- Geräte werden üblicherweise nicht direkt mit dem Prozessorbus verbunden
  - Zuviele unterschiedliche technische Realisierungen von Geräten
  - E/A-Controller als Vermittler zwischen Geräten und dem Rechner
  - Teilweise erlaubt der Controller den Anschluss verschiedener Geräte
    - Beispiel: RS232 für Maus, Tastatur, Modem

Ansteuerung von Geräten

- Die Interaktion zwischen Prozessor und E/A-Controller geschieht über den Prozessorbus
  - Jeder Controller stellt einen E/A-Adreßbereich mit speziellen Registern zur Verfügung
  - Kommandoregister erlauben die Übermittlung von Befehlen an den Controller
  - Statusregister erlauben die Abfrage des Controller- oder Gerätezustands
  - Datenregister (manchmal auch E/A-Puffer genannt) dienen dem eigentlichen Informationsaustausch

E/A-Architekturvarianten

- Speicherbasierte E/A
  - Register sind nicht von herkömmlichen Speicherzellen des Hauptspeichers zu unterscheiden
  - Es können normale Lese- und Schreibbefehle verwendet werden
- Dezidierter E/A-Bus
  - Eigene Befehle des Prozessors für E/A nötig
  - Zusätzliche Signale auf dem Steuerbus, die zwischen einem Zugriff auf den Hauptspeicher und einem Zugriff auf den E/A-Adressbereich unterscheiden
- Der Eintritt relevanter Ereignisse wird dem Prozessor über asynchrone Interrupts mitgeteilt

E/A-Bus-Controller

- Spezielle Form von Controllern, die ausgangsseitig einen standardisierten Bus für die eigentlichen Geräte-Controller zur Verfügung stellen
- Vorteile:
  - nur wenige Komponenten hängen am schnellen (und relativ kurzen) Prozessorbus
  - der Bus-Controller kann eine Reihe von Grundfunktionen eigenständig durchführen (z.B. die zentrale Verarbeitung von Interrupts)
  - bei standardisierten Gerätebussen braucht man nur einen Controller für viele Geräte
- Bekannteste Varianten: ISA, EISA, PCI, SCSI
  - teilweise mit Busmaster-Fähigkeit, zur eigenständigen Übernahme der Kontrolle auf dem Bus

Zeichen-/blockorientierte Geräte

- Unterscheidung der Geräte bezüglich der kleinsten Übertragungseinheit
- Zeichenorientierte Geräte
  - Beispiele: Tastatur, Maus, Drucker
  - Anforderung/Ankunft eines einzelnen Zeichens wird dem Prozessor kommuniziert
- Blockorientierte Geräte
  - Beispiele: Festplatten, CD-Laufwerke, viele Netzadapter
  - Die Datenübertragung wird erst bei Vorliegen eines kompletten Blocks (z.B. 1-4 kB) angestossen
- Blockorientierte Geräte unterstützen oft DMA (Direct Memory Access)
  - Beispiel: Direktes Übertragen eines Blocks von der Platte in den Hauptspeicher (ohne Prozessorbeteiligung)

Gliederung des Kapitels

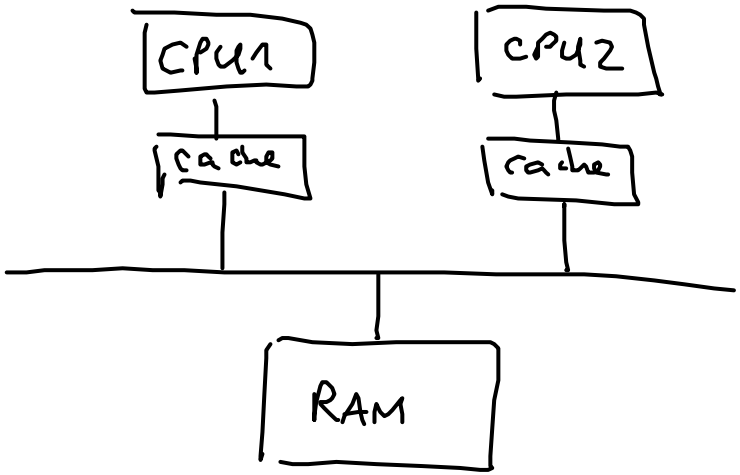
- Einführung (Von Neumann-Architektur)
- Prozessor
- Speicher
- Ein-/Ausgabe
- **Nebenläufigkeit**

Nebenläufigkeit

- Nebenläufigkeit = Gleichzeitigkeit mehrerer Aktivitäten
  - festgelegt durch die jeweilige Hardwarekonfiguration
  - direktes Maß für die potentielle Leistungsfähigkeit des Systems
- Nebenläufigkeit innerhalb des Prozessors
  - Instruktionspipelining
  - mehrere Recheneinheiten
- Nebenläufigkeit im E/A-Bereich
  - durch autonome Busse (z.B. SCSI), z.B. für DMA
- Multiprozessorsysteme
  - mehrere Prozessoren arbeiten gleichzeitig

Multiprozessorsysteme

- Mehrere Prozessoren über Speicherbus an gemeinsamen Speicher angeschlossen
  - Caches erhöhen den Datendurchsatz
  - Cache-Kohärenzproblem durch mehrere Caches
    - Lösung durch *snoopy caches*



- Klassifikation:
  - Asymmetrischer Multiprozessor
    - Entstanden aus einfachen Monoprozessorsystemen
    - Ein Master-Prozessor und mehrere Slave-Prozessoren
  - Symmetrischer Multiprozessor
    - Jeder Prozessor gleichberechtigt, insbesondere bzgl. Zugriff auf E/A
- Leistungsfähigkeit des Systembusses ist häufig der Flaschenhals bei Multiprozessorsystemen

Zusammenfassung

- Einführung (Von Neumann-Architektur)
- Prozessor
- Speicher
- Ein-/Ausgabe
- Nebenläufigkeit

- Wir werden in der “grossen” Vorlesung eine konkrete aber einfache Hardware kennen lernen, auf der ULIX programmiert wurde

Ausblick

- Jetzt wissen wir, worauf wir aufbauen können
- Was wollen wir eigentlich haben?
  - Was soll ein Betriebssystem eigentlich leisten?
- Im nächsten Kapitel:
  - Grundsätzliche Überlegungen über die vom Betriebssystem angebotenen Dienste und deren Modularisierung
  - Ziel: Grobarchitektur des Laufzeitsystems der Systemsoftware