TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik | Institut Systemarchitektur | Professur Datenschutz und Datensicherheit

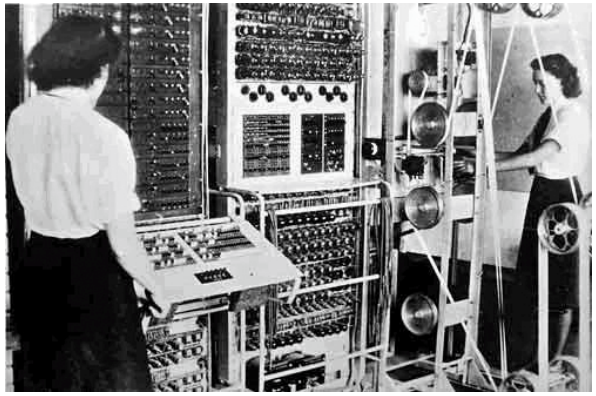
WS 2017/2018

3. Rechnerarchitektur

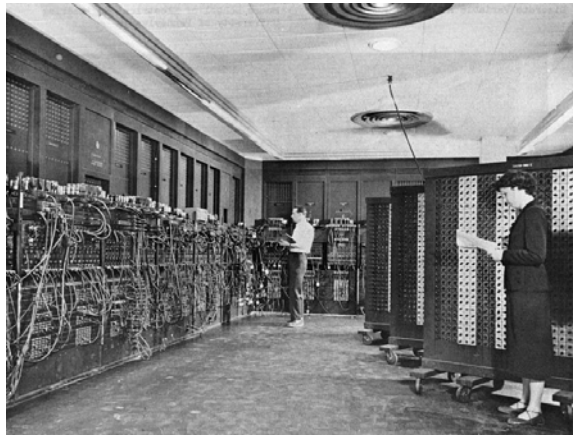
Dr.-Ing. Elke Franz
Elke.Franz@tu-dresden.de

3 Rechnerarchitektur – Anfänge der Computertechnik

Zuse Z3 (Deutschland)
1941



Colossus Mark II (UK)
1943

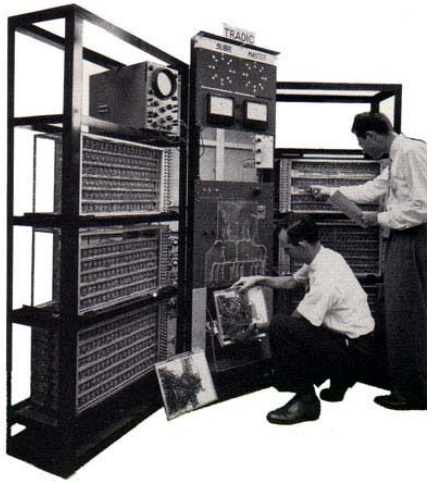


ENIAC (USA)
1946

- Basieren auf Relais für Rechen- und Speicherwerk
- Werden durch manuelles Ändern der Steckverbindungen programmiert
- Ein- und Ausgabe über Lampen

1940 1950 1960 1970 1980 1990 2000 2010 2020

3 Rechnerarchitektur – Entwicklung zum Heimcomputer



TRADIC - 1955

- Erste Transistorrechner
- Programmierung über Lochkarten
- Ausgabe über Drucker

PDP 1 - 1960

- Miniaturisierung der Rechentechnik
- Programmierung über Tastatur
- Ausgaben nun auch über Röhrenbildschirme

Apple II - 1977

- Aufkommen der ersten Heimcomputer
- Möglich durch Entwicklung von Mikroprozessoren und integrierten Schaltkreisen

1940 1950 1960 1970 1980 1990 2000 2010 2020

3 Rechnerarchitektur – Entw. zur heutigen Rechentechnik

IBM PC - 1981

- Blütezeit der Heim- oder Personal-Computer
- Viele konkurrierende Modelle

Intel Pentium CPU - 1993

- Beginn des Internet-Booms
- Rasante Weiterentwicklung der Rechentechnik

Apple iPad - 2010

- Weitere Miniaturisierung
- Verlagerung auf Mobile Computing und starke Vernetzung von Geräten

1940 1950 1960 1970 1980 1990 2000 2010 2020

Arten von Computern (Auswahl)

PC (Personal Computer)

- Individuell für einen Benutzer vorgesehen

Server und Clients

- Server stellt Daten für Vielzahl von Clients zur Verfügung

Mainframes

- Großrechner für Tausende von Benutzers

Supercomputer

- Computer mit extrem hoher Rechenleistung für komplexe Berechnungen

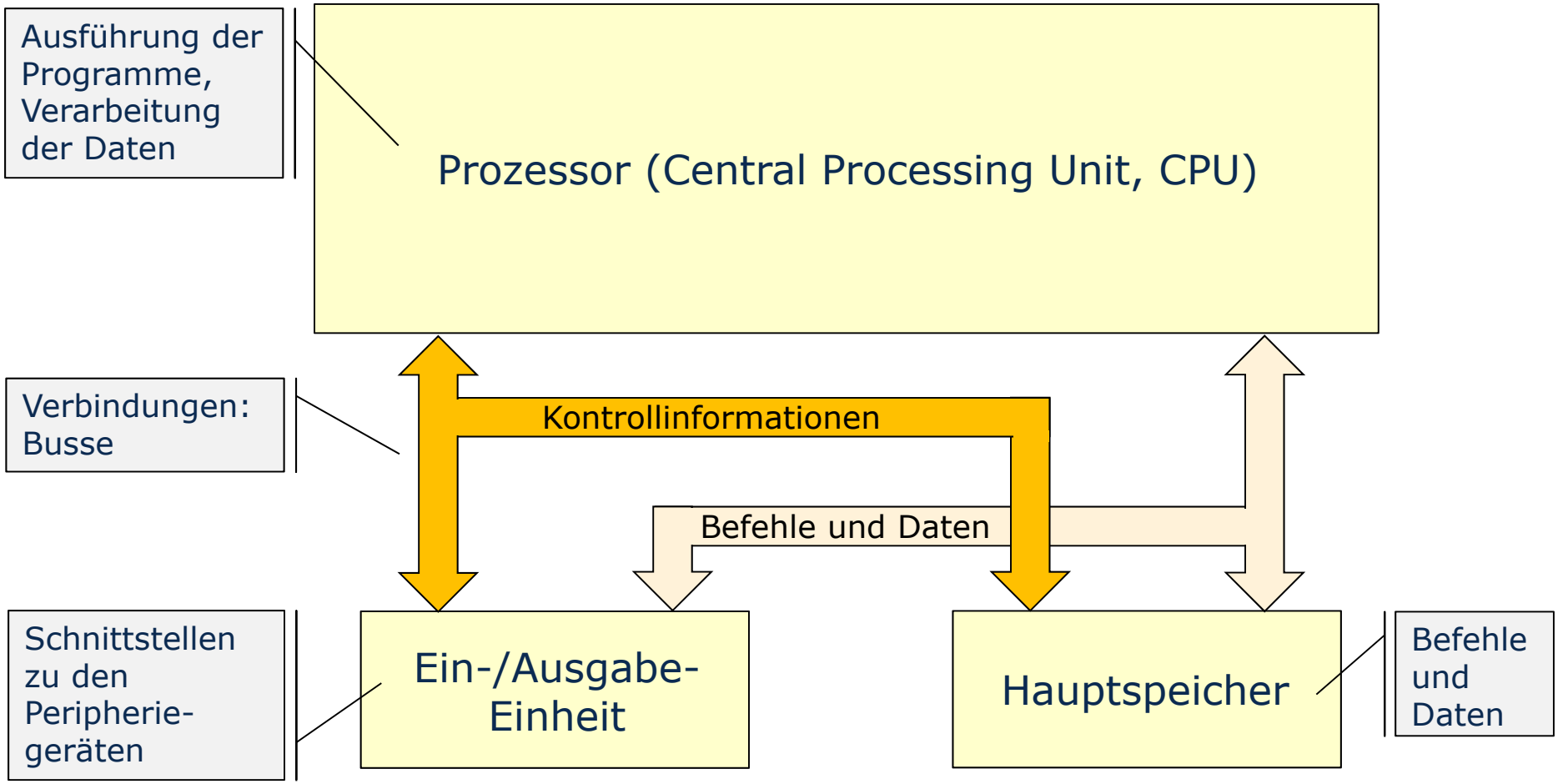
Embedded Systems (Mikrocomputersysteme)

- In technischen Kontext eingebettet (z.B. Waschmaschine, Auto)

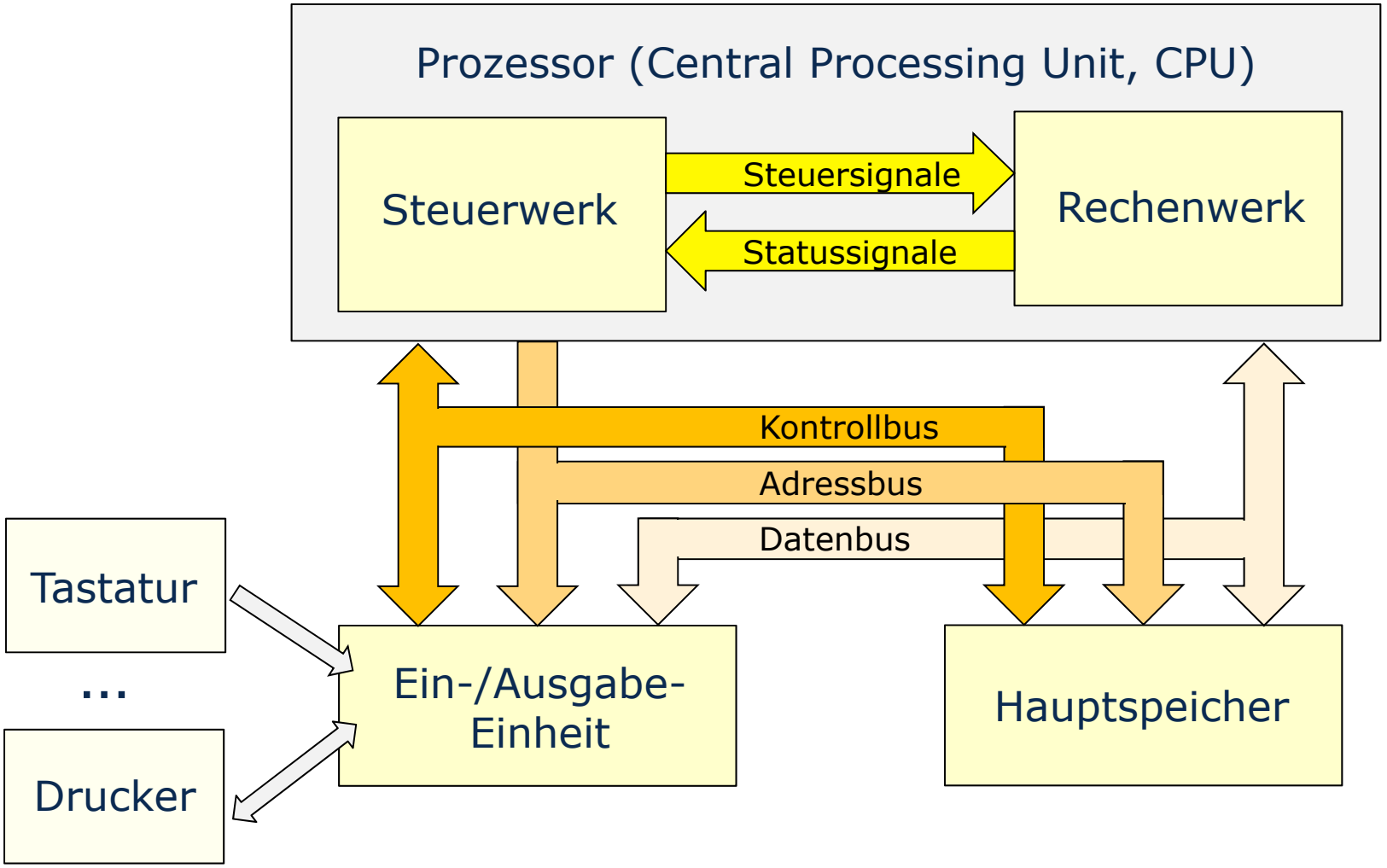
Rechenleistung (Performance)

- Wichtiges Kriterium für die Beurteilung von Computern
- Vergleich verschiedener Rechner bzgl. Performance schwierig:
  Verschiedene Operationen sind möglich, Bedeutung hängt vom Einsatzgebiet des Rechners ab
- Benchmark-Programme: Beispielprogramme, die stellvertretend für verschiedene Aufgaben stehen
- Grobe allgemeine Schätzung der Performance:
  - MIPS (Million Instructions Per Second)
  - MFLOPS (Million FLoating Point Operations Per Second)

3 Rechnerarchitektur – von-Neumann-Rechner



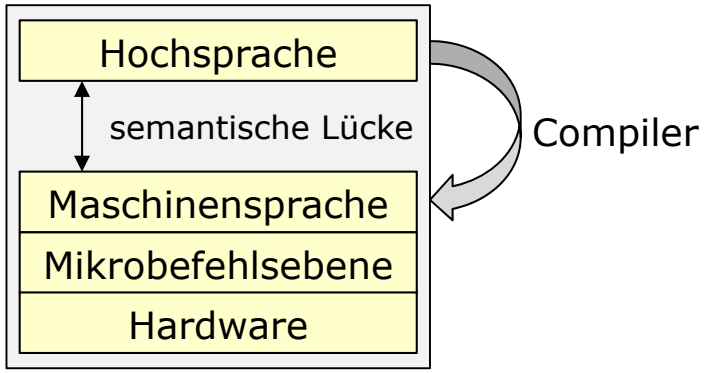
3 Rechnerarchitektur – von-Neumann-Rechner



Prinzipien

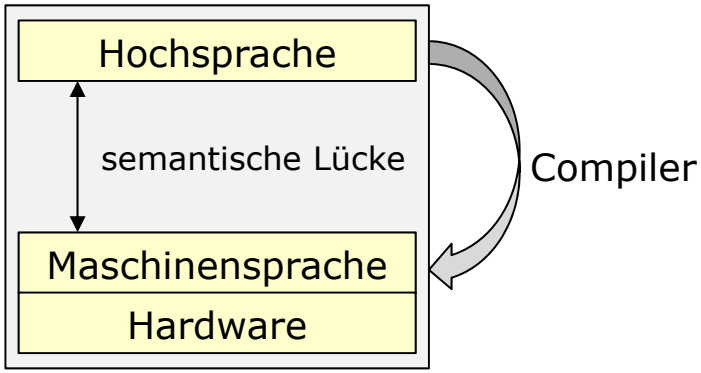
- Universeller Rechner – Abarbeitung beliebiger Programme
- Prozessor arbeitet taktgesteuert; Takt: Dauer eines Verarbeitungsschrittes
- Intern verwendete Signalmenge (Befehle und Daten) binär kodiert
- Programm und Daten im Speicher abgelegt, Programm und Daten können gelesen und geändert werden
- Einteilung des Hauptspeichers in gleichgroße Zellen, die fortlaufend nummeriert sind (Nummer = Adresse)
- Ein- und Ausgabegeräte: E/A-Baugruppen für verschiedene Schnittstellen; pro Baugruppe E/A-Register und Steuerregister; E/A-Register nummeriert (Adresse bzw. Port)
- Bearbeitung der Programmbefehle in der Reihenfolge der Speicherung, Änderung der Reihenfolge durch Sprungbefehle möglich

CISC (Complex Instruction Set Computer)



- Umfangreicher Maschinenbefehlssatz (Befehlssatz: Gesamtheit der Maschinenbefehle)
- Semantische Lücke geringer
- Compiler einfacher und effizienter
- Programme kürzer
- Zusätzlich Mikrobefehlsebene

RISC (Reduced Instruction Set Computer)



- Reduzierter Maschinenbefehlssatz
- Semantische Lücke größer
- Hardware optimiert, üblicherweise nur ein Takt pro Maschinenbefehl
- Cache vergrößert, mehr Register
- Compiler wesentlich komplexer
- Programme länger

3 Rechnerarchitektur – Verbreitete Architekturen

Von-Neumann Architektur

- Nur ein einziger Speicher für Daten und Programme
- Klassische vN-Architektur: *ein* Steuerwerk, *ein* Rechenwerk, *ein* E/A-System, *ein* Bussystem

Nachteil:
- Von-Neumann-Flaschenhals, bedingt durch das Bussystem

Nachteile bei der klassischen vNA:
- Code und Daten nicht getrennt
- Sequentielle Abarbeitung

- Dominierende Architektur im PC-Bereich

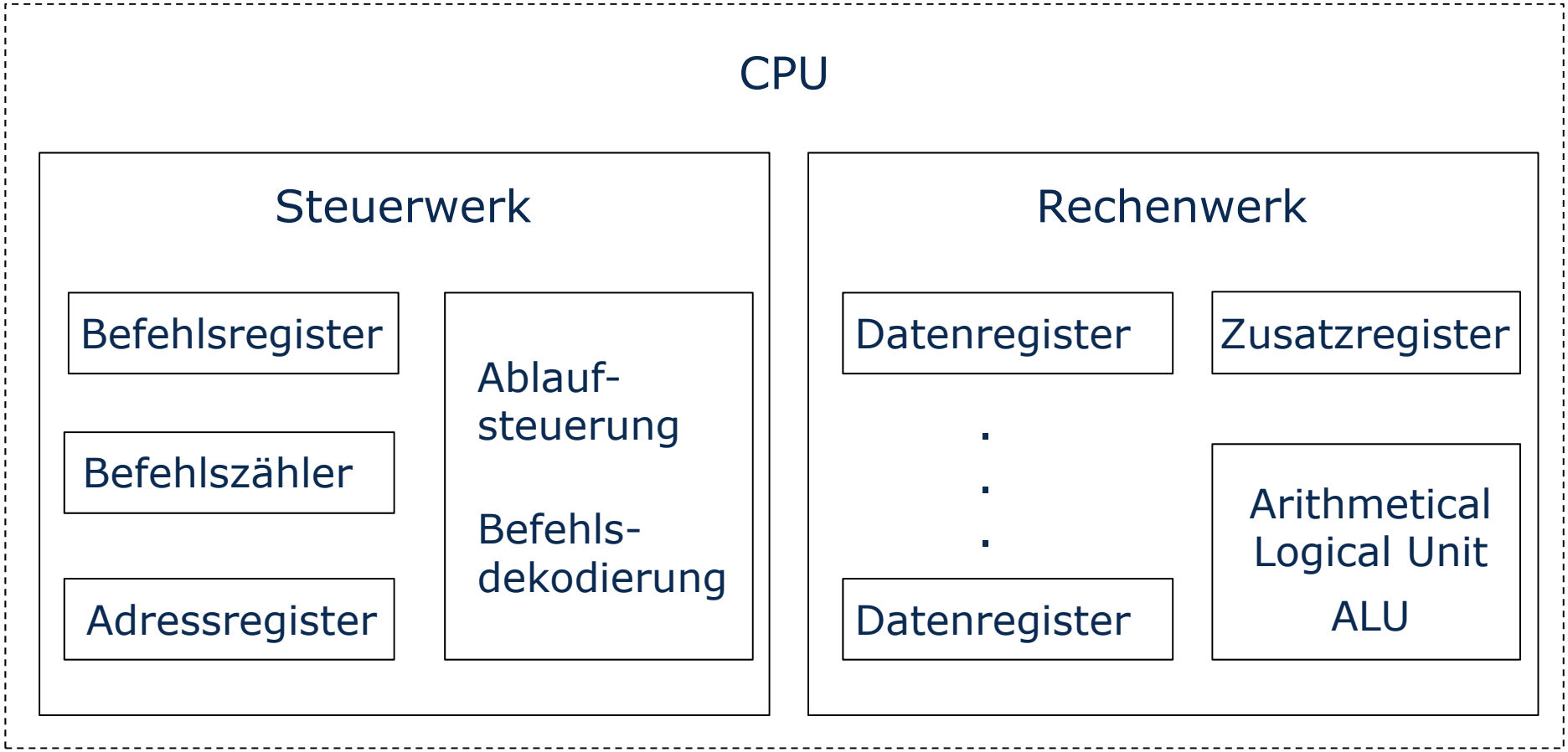
Harvard-Architektur

- Getrennte Speicher für Daten und Programme
- Getrennte Bussysteme
- Überwindet die Nachteile der von-Neumann Architektur

Nachteil:
- Aufteilung des Speichers muss vorab bekannt sein

- Vor allem dort verwendet, wo Anwendungen vorab bekannt sind (z.B. bei Embedded Systems)

CPU (Central Processing Unit)



Register

- Spezielle Speicherplätze in der CPU, auf die besonders schnell zugegriffen werden kann, mit Namen bezeichnet (z.B. AX, BX)
- Speicherung von Daten und Adressen von Daten, die von der ALU verarbeitet werden sollen
- Speicherung von Befehlen (Befehlsregister) und der Adresse des nächsten Befehls (Befehlszähler / *instruction pointer*)
- Breite der Register → Bezeichnung des Prozessors
- Wichtigstes Zusatzregister: Flagregister
  - Inhalt abhängig vom Ergebnis der ausgeführten Operation
  - Beispiele: Carry-Flag (Übertrag), Zero-Flag (Ergebnis = 0)

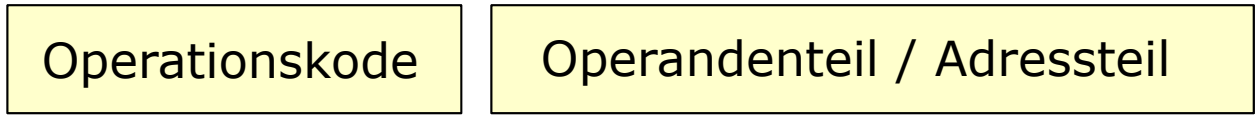
Start des Steuerwerks, danach

→ Steuerschleife (von-Neumann-Zyklus)
(zyklische Arbeit des Steuerwerks, Beendigung durch HALT-Befehl)

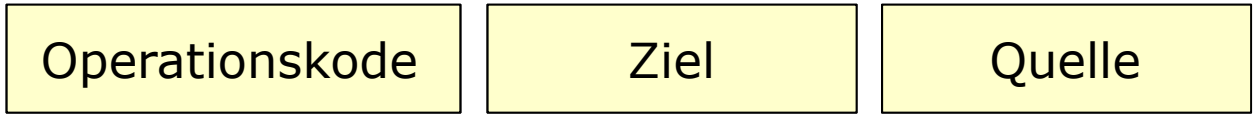
1. Holen des Befehls in das Befehlsregister
2. Dekodieren des Befehls
3. Holen der Operanden aus Speicher
4. Befehlsausführung
5. Rückschreiben der Ergebnisse
6. Bestimmen der Adresse des Nachfolgebefehls

3 Rechnerarchitektur – Maschinenbefehle

- Eingabe für das Steuerwerk
- Inhalt allgemein:



- Intel Syntax für die meisten Maschinenbefehle:



➢ Beispiele:

```
ADD     AX, BX   ; AX = AX + BX
MOV     BX, CX   ; BX = CX
DEC     CX       ; CX = CX – 1
INC     DX       ; DX = DX + 1
```

(AX, BX, CX, DX: Register)

Anmerkung: Notation in Assemblercode für eine verständliche Darstellung – intern sind Befehle, Adressen und Daten binär dargestellt.

Klassifikation

- **Arithmetische und logische Befehle**
  Addition, Inkrement, Dekrement, bitweise logische ODER-Verknüpfung, …
- **Bitmanipulationsbefehle**
  Löschen, Setzen, Ändern einzelner Operandenbits
- **Testbefehle** (evtl. implizit in anderen Befehlsarten)
  Testen auf Gerätebereitschaft, Operandenvorzeichen, …
- **Sprungbefehle**
  unbedingte Sprünge zu einer Zieladresse (z.B. für Unterprogramm-Aufruf und Rücksprung)
  bedingte Sprünge für Verzweigungen und Zyklen
- **Transportbefehle**
  Operandentransfer Speicher – Speicher, Speicher – Steuerwerk, …
- **Steuerbefehle**
  HALT-Befehl am Programmende, Schrittbetrieb, …

3 Rechnerarchitektur – Adressierungsarten

- **Unmittelbare Adressierung**
  Operanden sind Bestandteil des Befehls
- **Direkte oder absolute Adressierung**
  Operandenadresse steht im Befehlswort
- **Registeradressierung**
  Befehl bezieht sich auf den aktuellen Inhalt eines Registers
- **Indirekte Adressierung**
  Befehlswort enthält z.B. ein Register (*Register Indirect Adressing*), in welchem sich die Speicheradresse des Operanden befindet
- **Indizierte Adressierung**
  Adresse des Operanden wird zunächst errechnet, z.B. kann ein Register angegeben werden und ein konstanter Wert, der zum Registerinhalt zu addieren bzw. zu subtrahieren ist

3 Rechnerarchitektur – Befehlsabarbeitungsbeispiel

Aufgabe: Addition aller Zahlen von 1 bis 100, Ergebnis soll in Register AX stehen

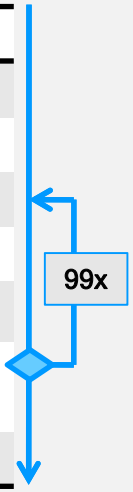
Variante A: Sequentielle Abarbeitung

Adr.	Befehl	Erläuterung
8001	MOV AX, 1	Register AX auf 1 setzen
8002	MOV BX, 2	Register BX auf 2 setzen
8003	ADD AX, BX	Addiere Reg. AX und BX
8004	MOV BX, 3	Register BX auf 3 setzen
8005	ADD AX, BX	Addiere Reg. AX und BX
…	…	…
8200	MOV BX, 100	Register BX auf 100 setzen
8201	ADD AX, BX	Addiere Reg. AX und BX
8202	HALT	Programm stoppen

Fazit
- Speicheraufwändig (202 Befehle)
- unflexibel

Variante B: Nicht sequentielle Abarbeitung

Adr.	Befehl	Erläuterung
8001	MOV AX, 1	Register AX auf 1 setzen
8002	MOV BX, 2	Register BX auf 2 setzen
8003	ADD AX, BX	Addiere Reg. AX und BX
8004	ADD BX, 1	Addiere 1 zu Register B
8005	CMP BX, 101	Vergleich Reg. BX mit 101
8006	JNZ 8003	Wenn BX ≠ 101, springe zurück zu Adresse 8003
8007	HALT	Programm stoppen



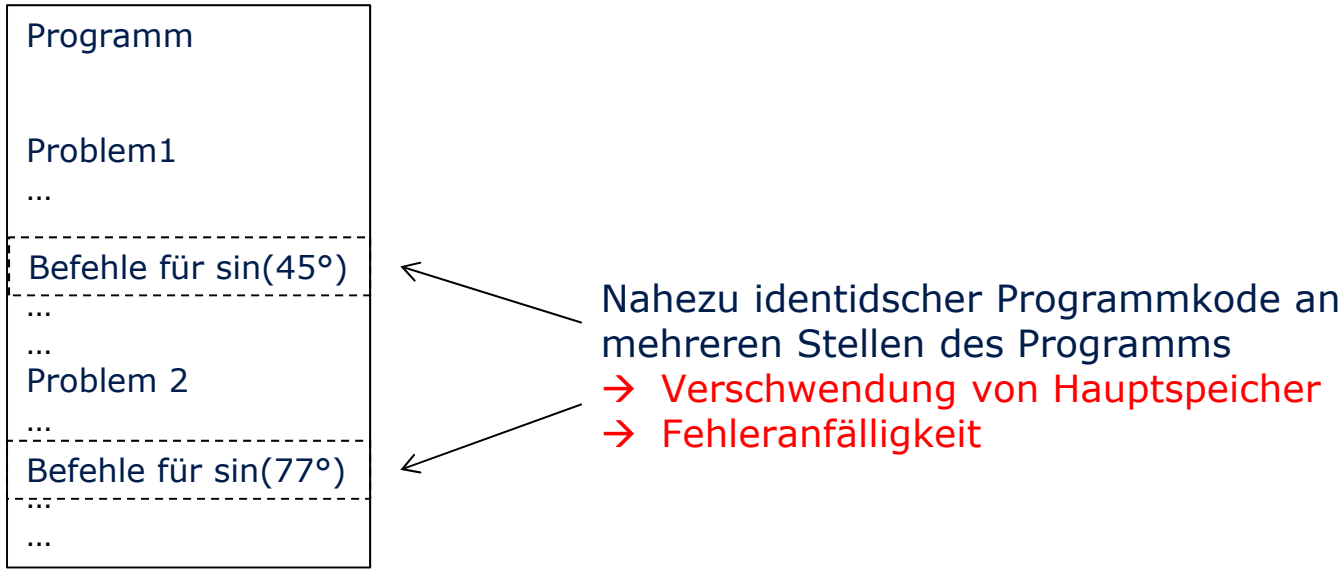
Fazit
- Speichereffizient (7 Befehle)
- flexibel

3 Rechnerarchitektur – Unterprogramme

Häufig benötigt man in einem Programm

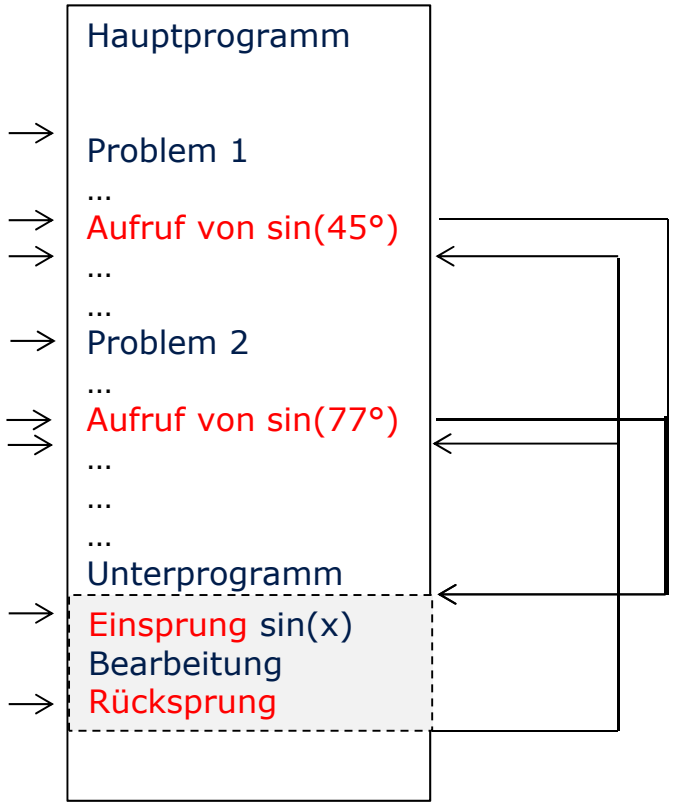
- mehrmals die gleichen Befehlsfolgen
- aber in unterschiedlichen Zusammenhängen

z.B. für Berechnungen wie sin(x), log(x), …



3 Rechnerarchitektur – Unterprogramme

Hauptprogramm ruft an mehreren Stellen das Unterprogramm auf.



Vorteile

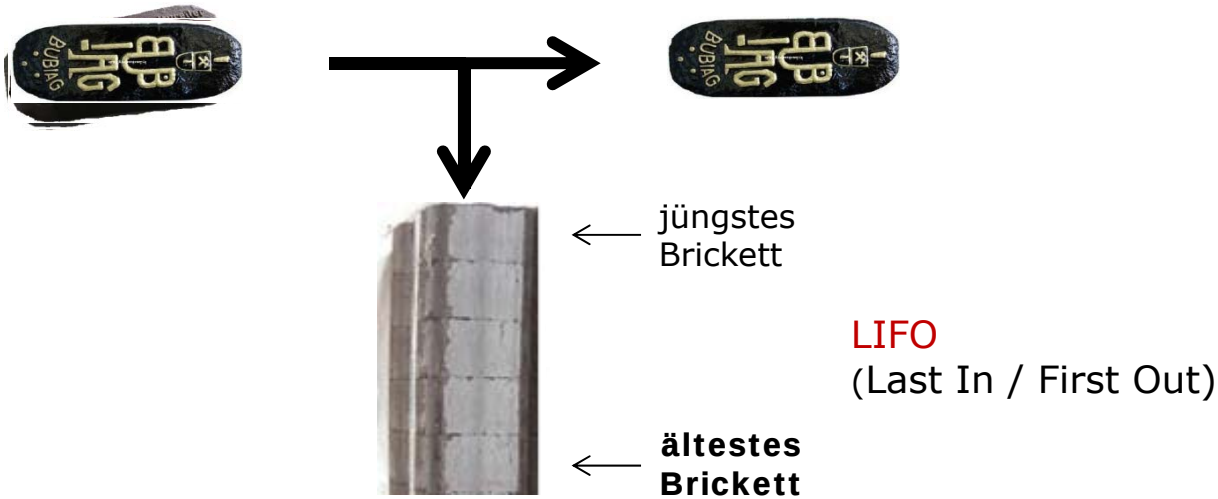
- Unterprogramm nur einmal im Hauptspeicher – weniger Redundanz
- Flexibler Code
- Wiederverwendbar
- Verbesserte Wartbarkeit

Problem

- Einsprungadresse eindeutig, aber Rücksprungadresse nicht eindeutig
  → muss beim Aufruf dem Unterprogramm übergeben werden
- Organisation: oftmals mittels Stack

Stack: Stapelspeicher, Kellerspeicher

(Begriff abgeleitet von Kohlelagerung im Keller)



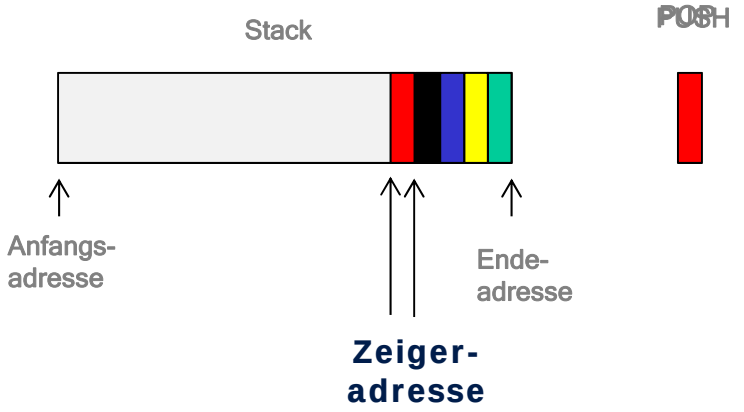
LIFO
(Last In / First Out)

Nur **2 Zugriffsfunktionen** (keine Speicheradressierung):

- Legen auf Stapel — sequentielles Stapeln
- Holen vom Stapel — des zuletzt gestapelten Objektes

Höhe des Stapels abhängig von der Menge der gestapelten Objekte

3 Rechnerarchitektur – Stack



Zugriffsbefehle

PUSH Register

1. Zeigeradresse wird verringert um Speicherwortlänge
2. Schreibvorgang: Registerinhalt → Stack an Position der Zeigeradresse

POP Register

1. Lesevorgang: Stackinhalt an Position der Zeigeradresse → Register
2. Zeigeradresse wird erhöht um Speicherwortlänge

Wenn Stack zu klein dimensioniert, evtl. Stacküberlauf!!!

Vorbereitung des Aufrufs:

Schreiben der Eingabeparameter für das Unterprogramm (UP) in Prozessorregister

CALL UP-Adresse (Aufruf des Unterprogramms)

1. Schreibvorgang: Rücksprungadresse → Stack
   (= Inhalt IP-Register* + Länge des Befehls „CALL“)
2. Schreibvorgang: UP-Einsprungadresse → IP-Register*

(*IP = Instruction Pointer = Befehlszähler)

RETURN (Rücksprungbefehl)

1. Lesevorgang: Rückkehradresse vom Stack holen
2. Schreibvorgang: Rückkehradresse → IP-Register

3 Rechnerarchitektur – Privilegien

Privilegierter Status für die Betriebssystemroutinen

- Voller Zugriff auf den gesamten Hauptspeicher (u.a. Ressourcen)

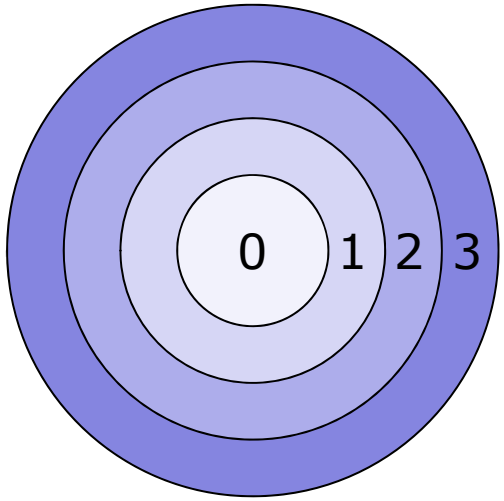
Unprivilegierter Status für die Anwendungsprogramme

- Einschränkung des Hauptspeicherzugriffs auf „eigenen" Bereich (hardwaremäßig überwacht), nicht alle Maschinenbefehle nutzbar

- Status: Flag im Maschinenstatuswort (Register)

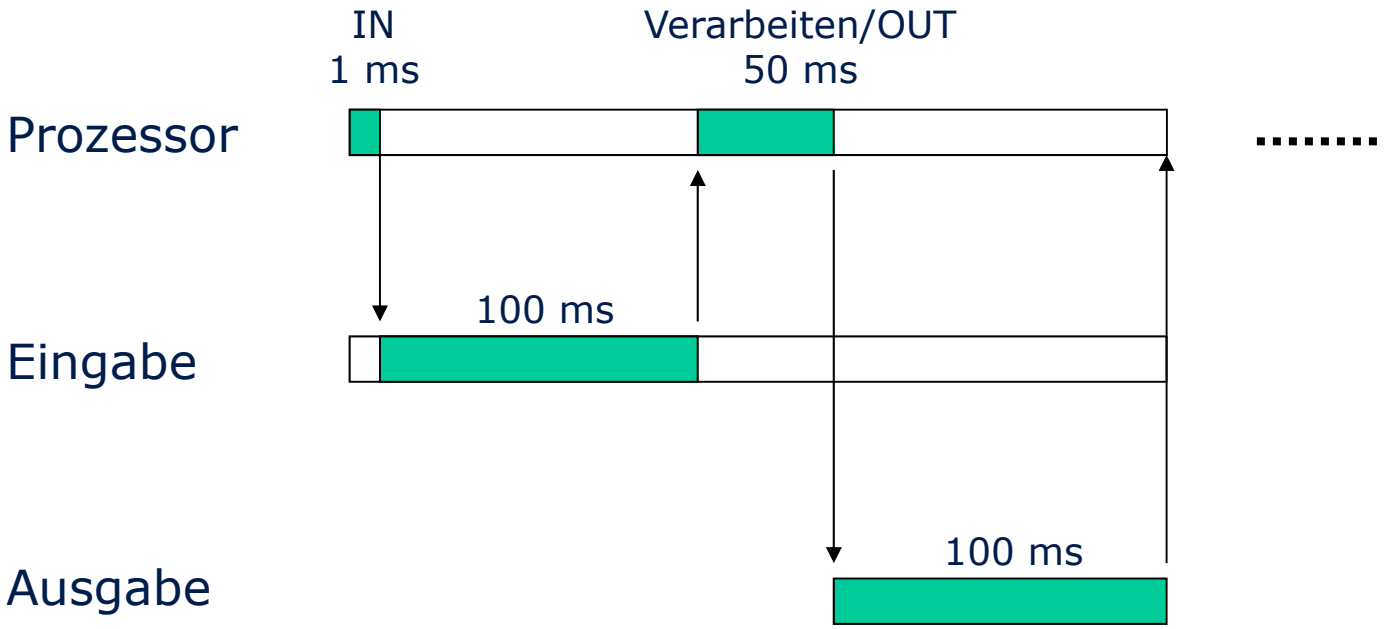
Vorteile der Privilegierung

- Beim Laufen eines fehlerhaften Programms kann das Überschreiben der Speicherbereiche anderer Prozesse verhindert werden
- Verringerte Absturzgefahr, erleichterte Fehleranalyse

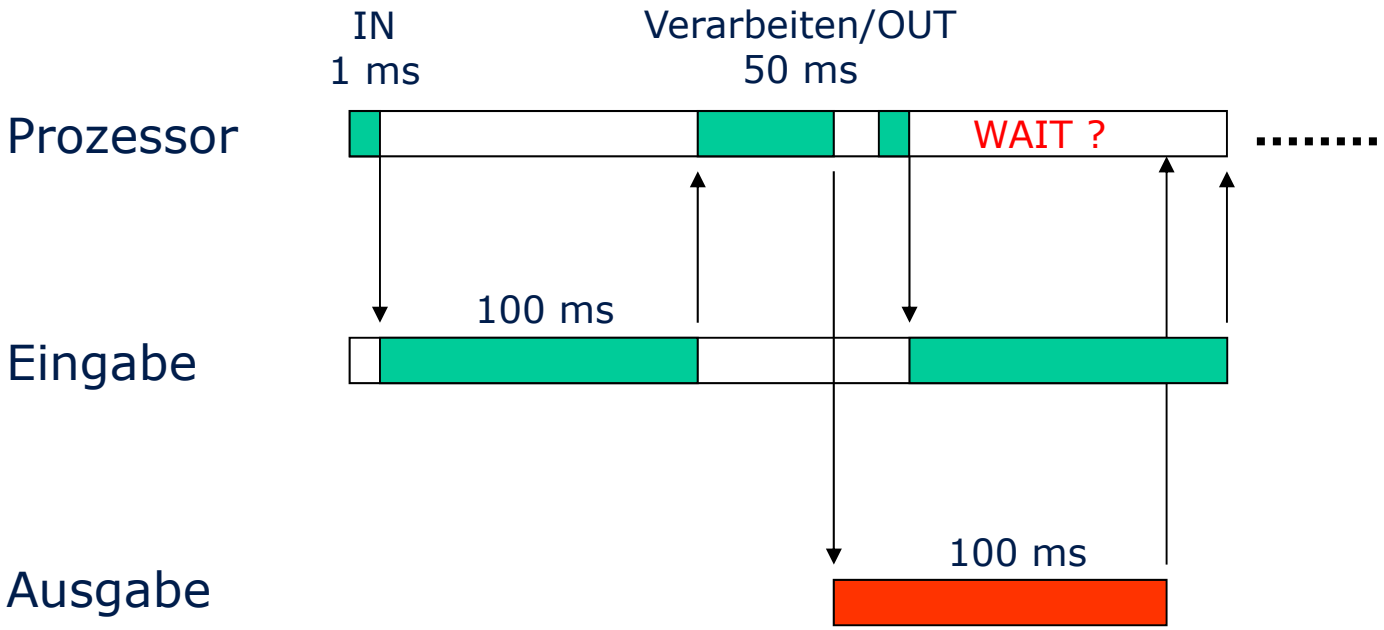


Beispiel: Domains (Ringe)
0: Kernel Mode
1-3: User Mode (Anwendungen)

Synchrone Arbeit



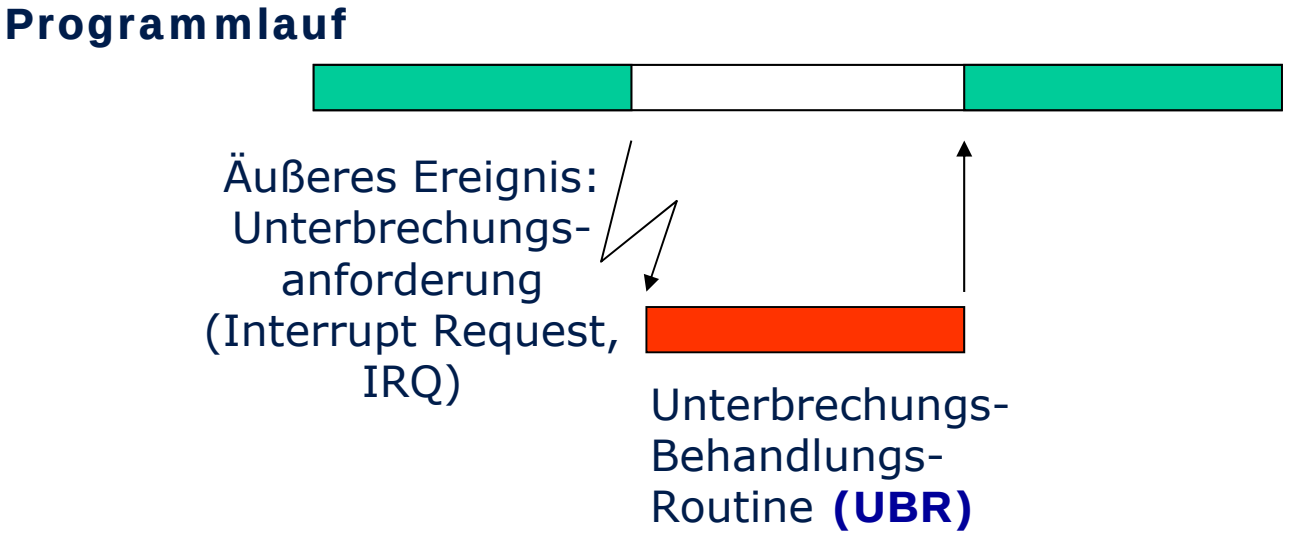
Asynchrone Arbeit



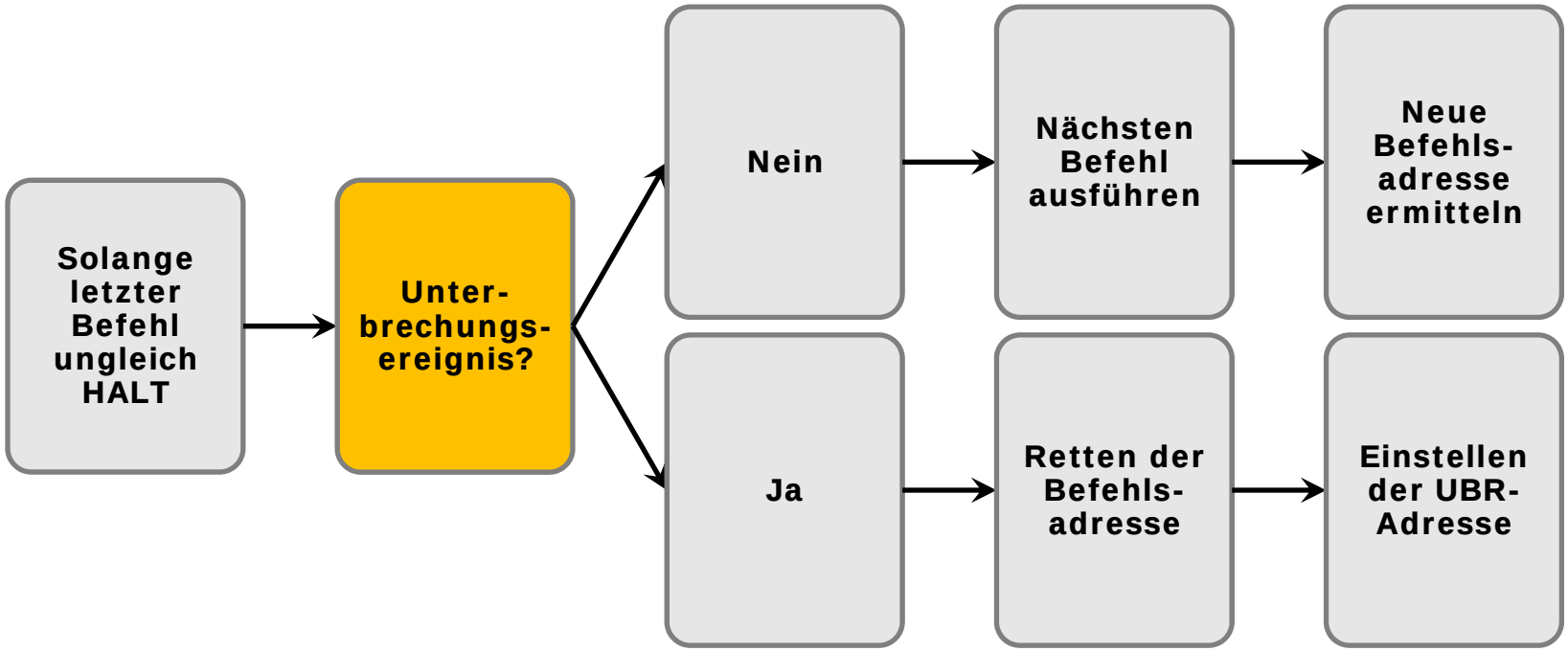
Unterbrechungen (Interrupts)

Polling: periodisches Abfragen aller E/A-Gerätes
nicht sehr zweckmäßig, weil hohe Systembelastung

Interruptkonzept erlaubt dem Prozessor Reaktion auf (asynchrone) äußere Ereignisse



3 Rechnerarchitektur – Interrupts



- Nach Befehlsausführung Überprüfung auf Unterbrechung
- Bei Unterbrechung Speichern der aktuellen Befehlsadresse und Aufruf der Unterbrechungsroutine (UBR)
- Ansonsten Ausführen des nächsten Befehls (Normalbetrieb)

Unterbrechungssystem Intel-Architektur

- prinzipiell 256 verschiedene Unterbrechungen
- durch eine 8 Bit lange Interruptnummer bezeichnet
- Adressen der UBR am Anfang des Hauptspeichers (jeweils 4 Byte)

Die ersten fünf Interrupts sind hardwaremäßig festgelegt (interne UB)

INT 0	bei Division durch Null
INT 1	nach jedem Befehl bei Einzelschrittbetrieb (für Testzwecke)
INT 2	bei NMI-Signal (Nichtmaskierbarer Interrupt) am Prozessor für externe Unterbrechungen höchster Priorität
INT 3	bei Erreichen eines Haltepunktes im Testbetrieb
INT 4	bei Datenüberlauf nach Rechenoperationen

3 Rechnerarchitektur – Interrupts Intel-Architektur

Hardwareinterrupts

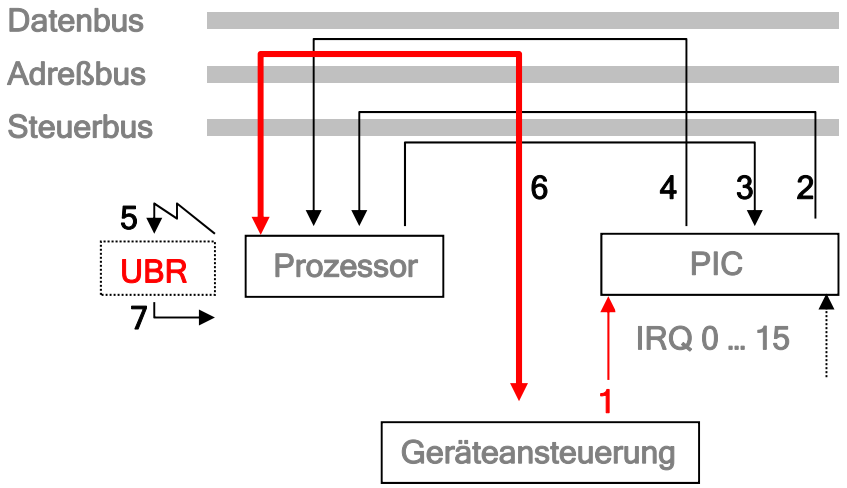
- Interne Unterbrechungen bei Ausnahmeereignissen
  - Division durch Null
  - Spannungsausfall
- Externe Unterbrechungen bei asynchroner Arbeit mit Geräten
  - Initialisierung einer Unterbrechungsbehandlungsroutine
  - Starten E/A-Operation
  - Warten auf Unterbrechung (E/A-Ende-Meldung vom Gerät)
  - Ende E/A anzeigen

Softwareinterrupts (Traps)

- Hilfsmittel zur Kommunikation von Programmen und Betriebssystemfunktionen

Externe Unterbrechungen in Intel-Architektur

1. Gerät meldet UB (IRQ 0 bis IRQ 15) an beim Interrupt-Controller PIC (Programmable Interrupt Controller)
2. PIC gibt Signal über Steuerbus zum Prozessor
   Unterbrechung der Befehlsabarbeitung im Prozessor
3. Prozessor quittiert
4. liest anschließend die zum Ereignis gehörende UB-Nummer vom PIC
5. Aufruf Unterbrechungsbehandlungroutine UBR
6. Innerhalb von UBR Datenaustausch mit Gerät
7. Nach UBR-Ende Fortsetzung des unterbrochenen Programms



Innerhalb UBR
Retten/Wiederherstellen
aller Parameter des
unterbrochenen
Programmes

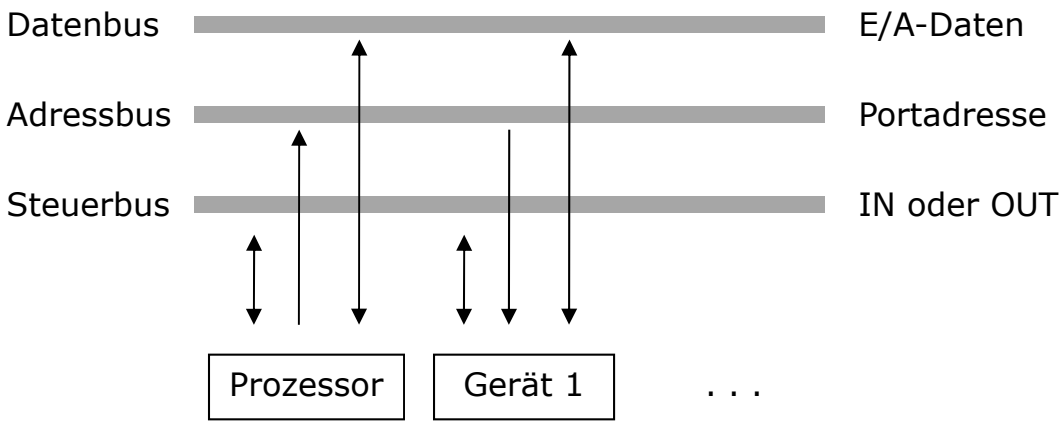
32 Rechnerarchitektur – Zugriff auf E/A-Geräte

Zugriff auf E/A-Geräte ähnlich wie Zugriff auf Hauptspeicher

- Initiative geht immer vom Prozessor aus,
- Geräte beobachten Bus und reagieren, wenn sie angesprochen werden.

Prozessor gibt Signale auf **Steuerbus**,

- Entscheidung Hauptspeicherzugriff oder E/A-Operation, Lesen / Schreiben
- am Adressbus liegt Hauptspeicher- bzw. Geräteadresse (Portadresse) an
- über Datenbus werden die Daten vom/zum Ziel übertragen



Caching

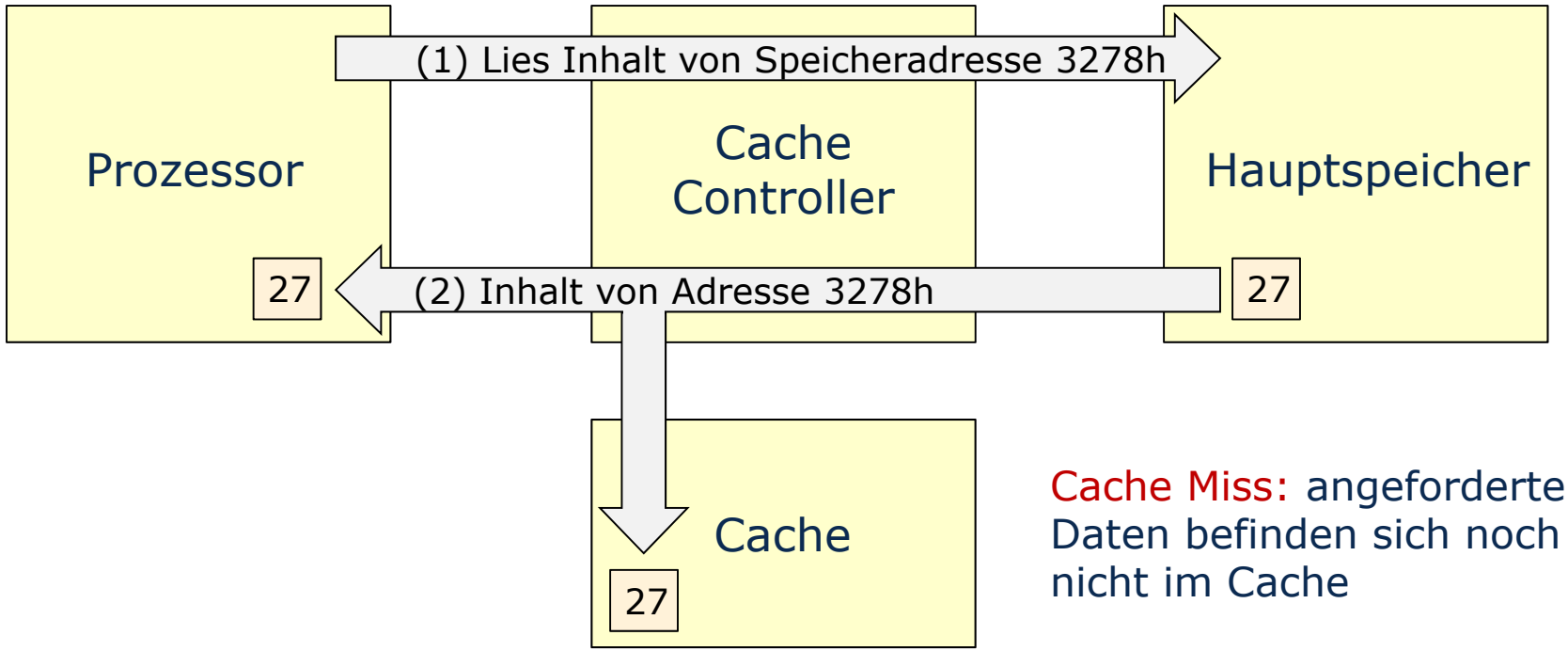
Ziel: Zugriffszeit auf langsamere Medien verkürzen
Cache: kleiner, schneller Zwischenspeicher

Vergleich von Speichermedien:

Medium	Mittlere Zugriffszeit	Erreichbare Daten-transferrate in MB/s
Register	0,1 ns	70000
Cache	2 ns	20000
Hauptspeicher	10 ns	6000
Festplatten	3,5 ms	150
Optische Platten (CD, DVD, …)	25 ms	10

[Gumm, Sommer: Einführung in die Informatik. Oldenbourg Verlag, 2012.]

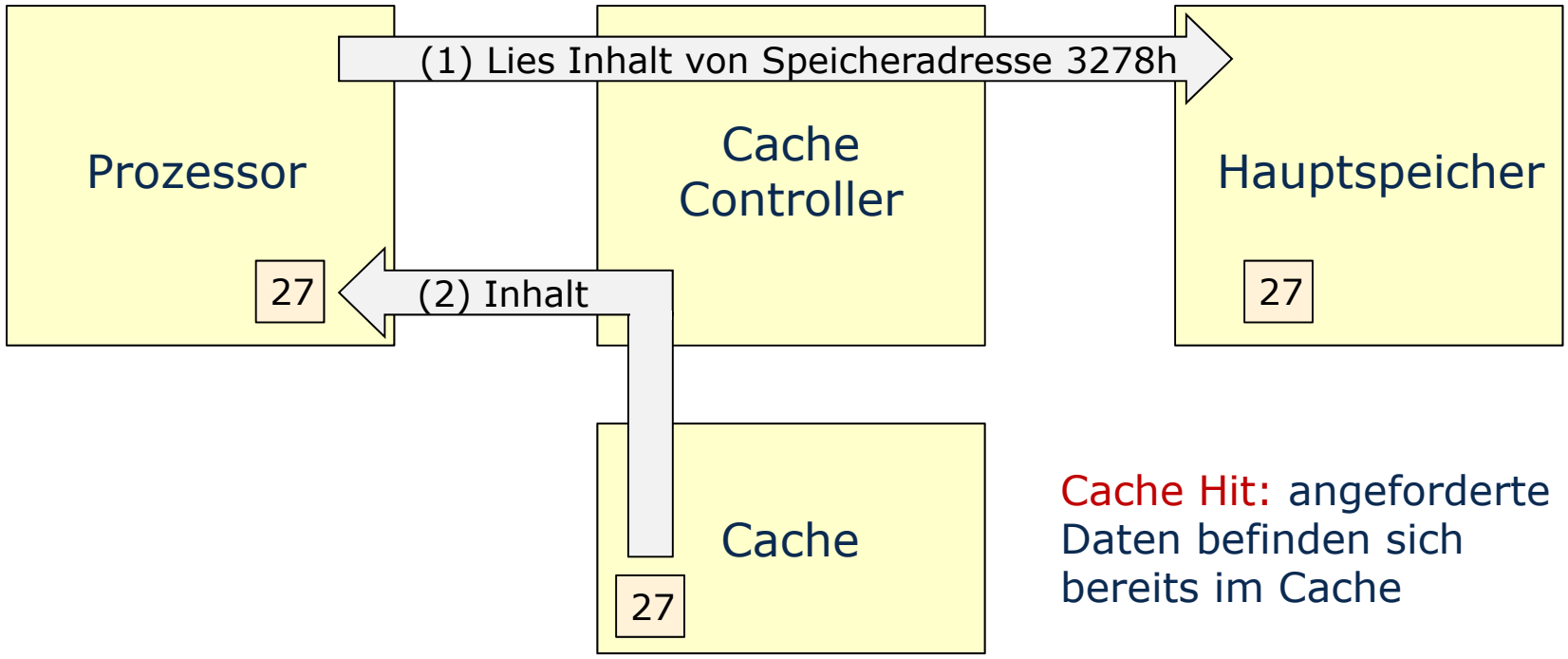
Caching beim Lesen von Daten (1)



Cache Miss: angeforderte Daten befinden sich noch nicht im Cache

Üblicherweise wird nicht nur eine Speicheradresse eingelesen, sondern mehrere (Cache Line) → erstmalige Zugriffe auf benachbarte Adressen werden ebenfalls beschleunigt.

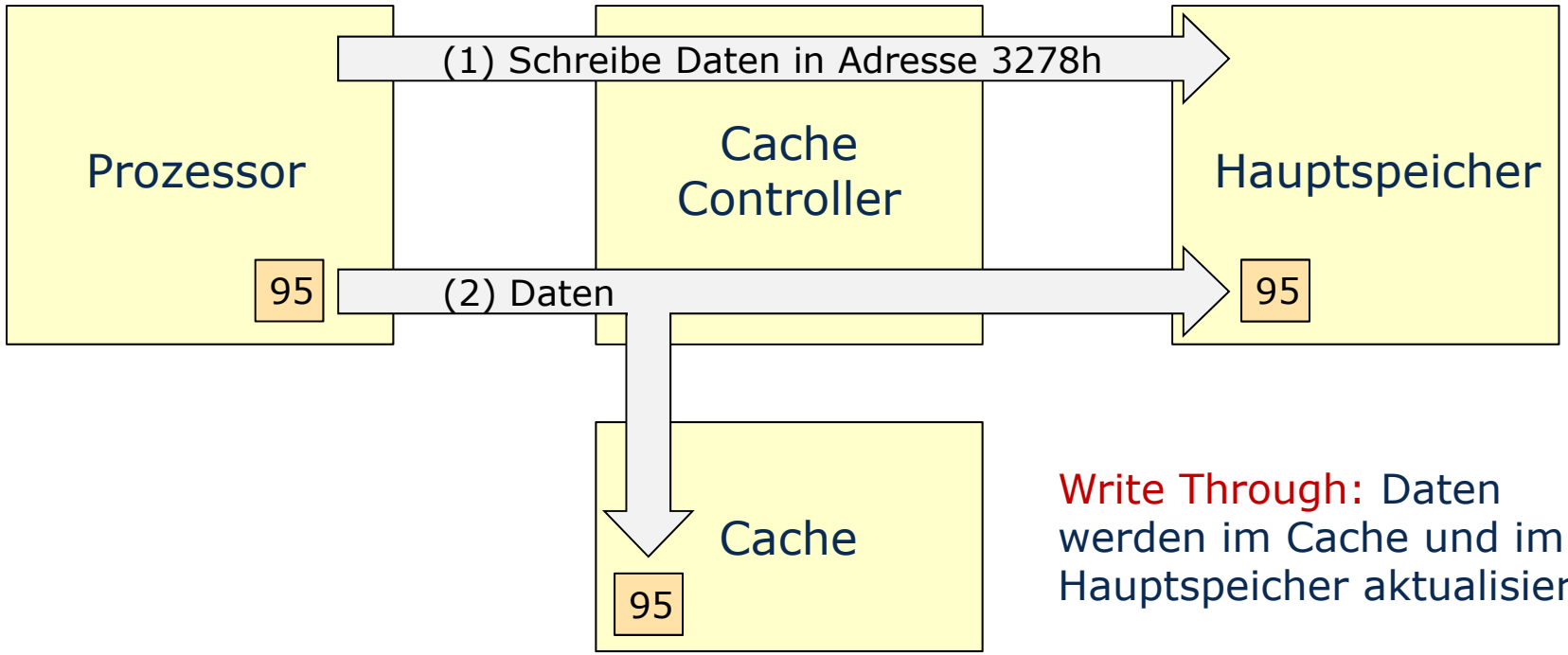
Caching beim Lesen von Daten (2)



Cache Hit: angeforderte Daten befinden sich bereits im Cache

Der Cache ist für den Prozessor transparent, d.h., der Prozessor merkt nicht, ob die Daten aus dem Cache oder dem Hauptspeicher kommen.

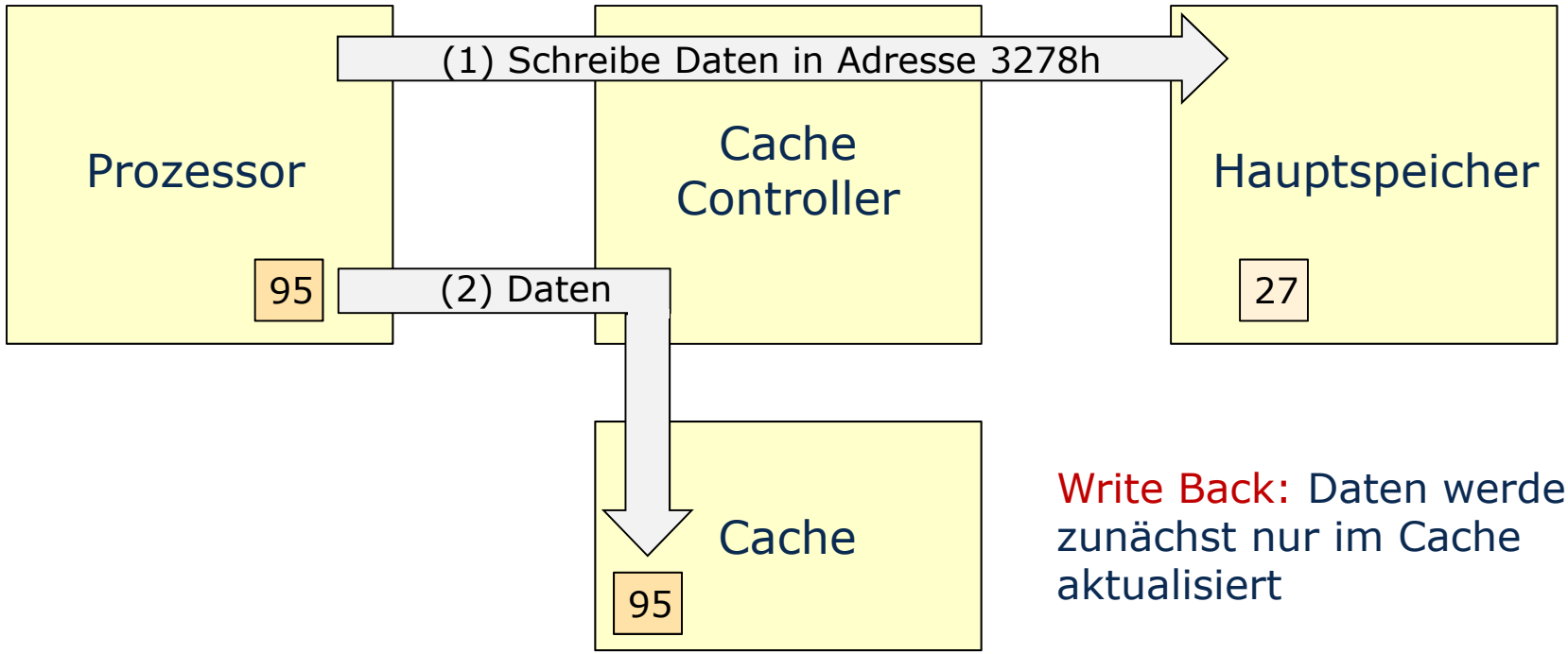
Caching beim Schreiben von Daten: Write Through



Write Through: Daten werden im Cache und im Hauptspeicher aktualisiert

Die Daten in Hauptspeicher und Cache stimmen überein, d.h., sie sind konsistent. Nachteil: Durch den Zugriff auf den Hauptspeicher langsam.

Caching beim Schreiben von Daten: Write Back



Write Back: Daten werden zunächst nur im Cache aktualisiert

Der Schreibvorgang ist schneller, da nur im Cache geschrieben wird. Nachteil: Hauptspeicher und Cache sind nicht mehr konsistent.
Mögliche Strategie: Cash Flush (Rückschreiben in bestimmten Zeitabständen).

- Controller speichert Zuordnung der Cache Lines zu den Teilen des Hauptspeichers sowie die Information, welche Cache Lines geändert wurden
- Cacheable Area: Teil des Hauptspeichers, der mit Cache Lines abgedeckt werden kann

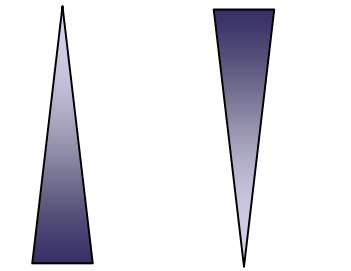
Cache-Hierarchien

- Speicher: in der Regel je schneller, desto teurer

1st Level Cache (L1)

2nd Level Cache (L2)

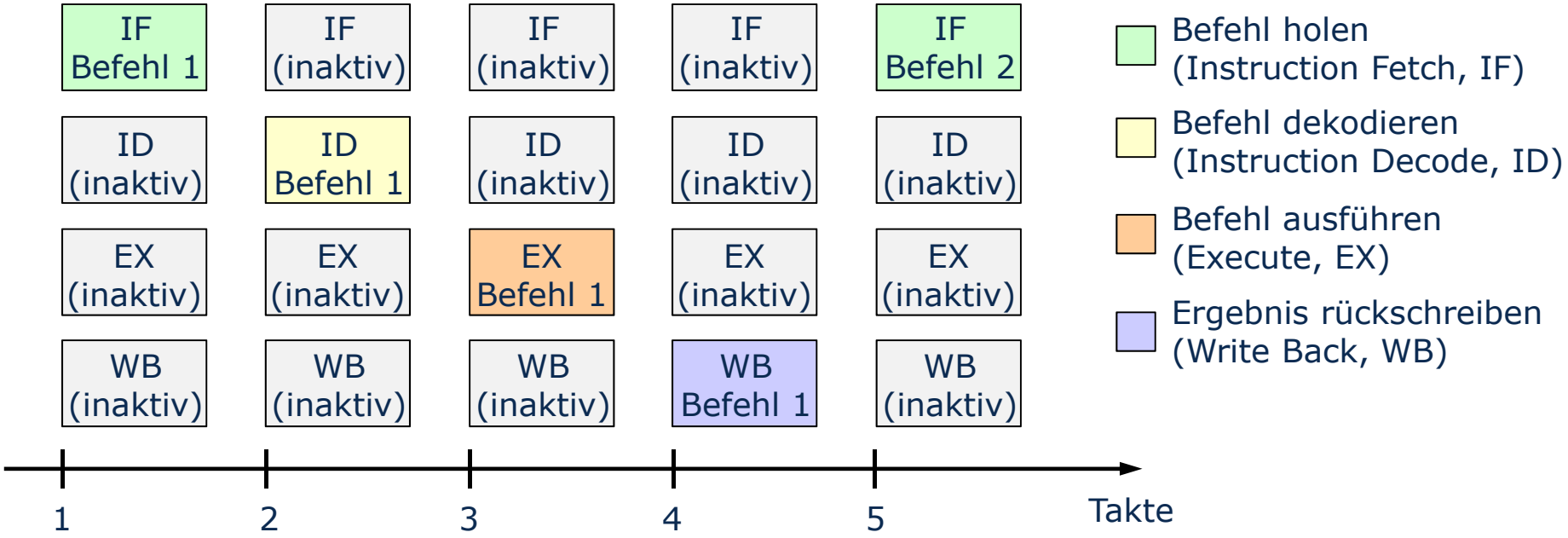
3rd Level Cache (L3)



- Verwendung für Caches z.B. auch für Zugriffe auf Festplatten oder für Webzugriffe (Browser-Cache)

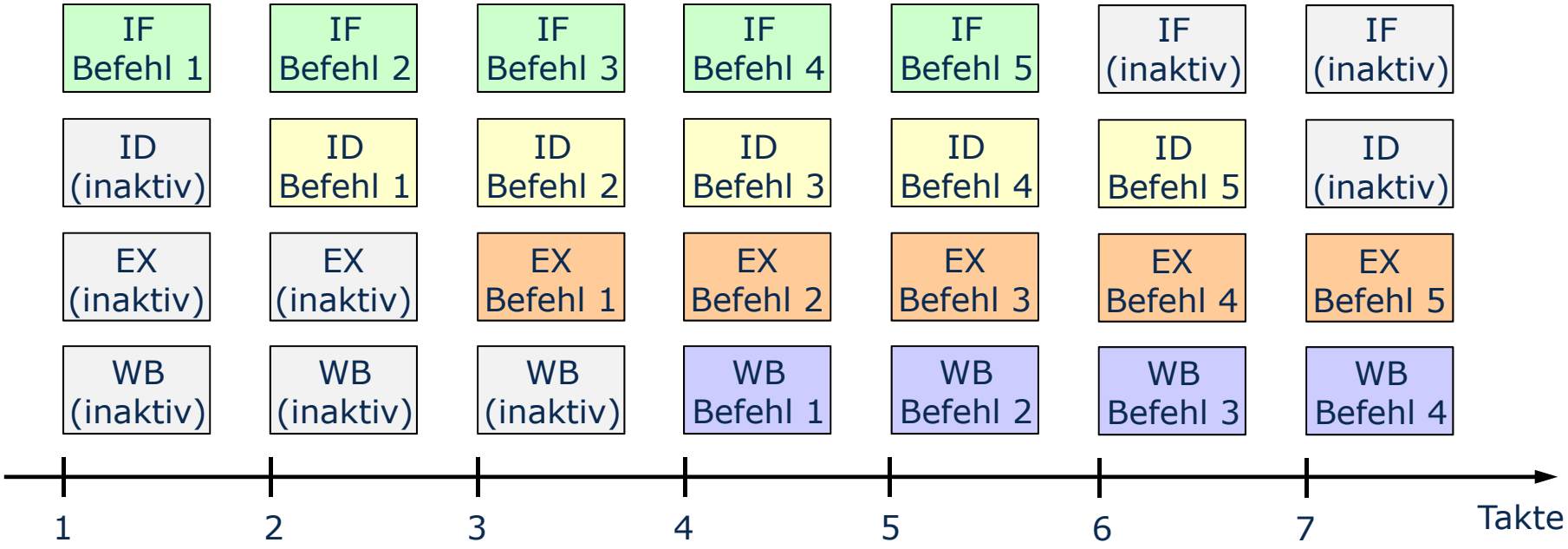
Motivation

- Befehlsabarbeitung erfordert mehrere Takte
- CPU: Hardwarekomponenten für die einzelnen Schritte
  → nur eine Baugruppe aktiv → Zeitverlust
  → Beispiel: 4 Schritte, Annahme: 1 Takt pro Schritt



Pipelining

- Mehrere Befehle gleichzeitig im Prozessor in verschiedenen Bearbeitungsstufen → Parallelisierung der Arbeit im Prozessor
- Voraussetzung: Hardwarekomponenten können unabhängig voneinander arbeiten
- Hardwarekomponenten → Pipeline-Stufen (Beispiel: 4 Stufen)



- Zu Beginn Vorrüstzeit, bis Pipeline gefüllt
- Pipeline sollte nicht leer laufen
- Ausführung eines Schrittes kann mehrere Takte dauern – langsamste Stufe entscheidend für Geschwindigkeit der Pipeline
- Einsatz oft an mehreren Stellen im Prozessor

Probleme mit Programmverzweigungen

- Pipeline funktioniert gut bei sequentieller Abarbeitung von Befehlen
- Änderung der Reihenfolge jedoch möglich

Adr.	Befehl	Erläuterung
8001	MOV AX, 1	Register AX auf 1 setzen
8002	MOV BX, 2	Register BX auf 2 setzen
8003	ADD AX, BX	Addiere Reg. AX und BX
8004	ADD BX, 1	Addiere 1 zu Register B
8005	CMP BX, 101	Vergleich Reg. BX mit 101
8006	JNZ 8003	Wenn BX ≠ 101, springe zurück zu Adresse 8003
8007	...	...

Problem: Erst nach Ausführung des Befehls ist klar, ob es mit dem Befehl aus Adresse 8007 oder dem Befehl aus Adresse 8003 weitergeht.

Strategien bei Programmverzeigungen (Beispiele)

- Abwarten ineffizient – Pipeline läuft leer
- Befehle „auf Verdacht" in die Pipeline holen und ggf. verwerfen

Statische Ansätze (Static Prediction)

- Predict not Taken: Annahme, dass keine Verzweigung erfolgt
- Predict Taken: Annahme, dass Verzweigung erfolgt

Dynamische Ansätze (Dynamic Prediction)

- Branch Prediction: Versuch, vorherzusagen, ob eine Verzweigung erfolgt oder nicht, z.B. mit einer Branch History Table (BHT), in welcher gespeichert ist, ob beim letzten Mal verzweigt wurde
- Speculative Execution: es werden beide Möglichkeiten weiterverfolgt, bis klar ist, ob verzweigt wird oder nicht (zwei Pipelines)

Parallelisierung zur Performancesteigerung

- eine der wichtigsten Ansätze zur Performancesteigerung
- Pipelining: Parallelisierung unterhalb der Maschinenbefehlsebene
- Maschinenbefehlsebene: mehrere Ausführungseinheiten

- Prozessorebene: Multiprozessorsysteme
  - Multiprozessorboards
  - Multicore-CPUs
  - gemeinsamer Arbeitsspeicher
  - Kommunikation: über Bussystem; Network-on-Chip (NoC)

- Rechnerebene: Multirechnersysteme
  - kein gemeinsamer Arbeitsspeicher
  - Kommunikation i. Allg. über Netzwerk

Aufwand für Parallelisierung

Hardware

- Spezielle Anforderungen insbesondere bei Multiprozessorsystemen, z.B. an Boards und Prozessoren; Kommunikation zwischen den Prozessoren muss unterstützt werden; Kühlungsprobleme

Software

- Parallelisierung muss unterstützt werden, sonst keine optimale Nutzung möglich
- Verwaltungsaufgaben (Multiprozessor- und Multirechnersysteme):
  - Aufteilung der Rechenarbeit auf die Knoten (Kerne/Prozessoren/Rechner)
  - Load Balancing (gleichmäßige Verteilung der Rechenlast während der Bearbeitung)
  - Sammeln der Teilergebnisse und Ermitteln des Gesamtergebnisses
  - Zugriffe auf gemeinsame Ressourcen synchronisieren