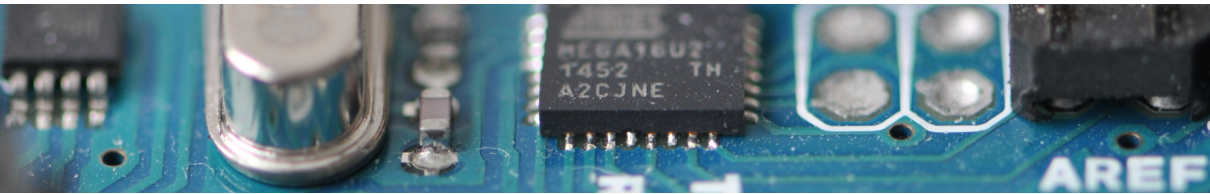
universität
innsbruck



Rechnerarchitektur

Leistung

Univ.-Prof. Dr.-Ing. Rainer Böhme

Wintersemester 2020/21 · 20. Jänner 2021

Gliederung

1. **Leistungsmessung**
2. Caches: Optimierung des Speicherzugriffs
3. Pipelines: Optimierung der CPU-Nutzung

Leistung

Allgemeine Definition aus der **Physik:**

$$\text{Leistung} = \frac{\text{Arbeit}}{\text{Zeit}}$$

Arbeit in der **Informatik:** i. d. R. Rechenoperationen

- Instruktionen allgemein $\rightarrow$ MIPS (*million instructions per second*)
- Integer-Operationen $\rightarrow$ MOPS (*million operations per second*)
- Gleitkommaoperationen $\rightarrow$ FLOPS (*floating point operations . . .*)
- . . .

Die Einheiten lassen sich nur sehr begrenzt ineinander umrechnen.

Leistungsverhältnis

Bei **konstanter Arbeit:**

$$\text{Beschleunigung } S = \frac{\text{Ausführungszeit unter Referenzbedingungen}}{\text{Ausführungszeit unter Testbedingungen}}$$

Beispiele

1. Referenzbedingung: Smartphone aus dem Jahr 2018
   Testbedingung: Smartphone von heute

   $S = 2$ bedeutet, das neuere Smartphone ist doppelt so leistungsfähig.

2. Referenzbedingung: Bubble-Sort-Algorithmus auf Intel 8086
   Testbedingung: Quicksort-Algorithmus auf Intel 8086

   $S = 4$ bedeutet, Quicksort ist (für die gewählten Daten, insb. Datenmenge) viermal schneller. **Grund:** weniger Instruktionen

Amdahlsches Gesetz

Annahme Anteil α (z. B. in %) eines Programms lässt sich um Faktor c beschleunigen

Es gilt:

$$S = \frac{t}{(1-\alpha)\cdot t + \alpha \cdot \frac{t}{c}} = \frac{1}{(1-\alpha) + \frac{\alpha}{c}}$$

Beispiel

- 20 % der Ausführungszeit wird für Divisionen benötigt.
- Spezielle Hardware (z. B. Koprozessor) beschleunigt die Division um den Faktor $c = 10$.
- Rechnerische Beschleunigung $S = 1.219512$

Amdahl 1967 (für Parallelisierung)

Messung der Ausführungszeit

Verwendung von **Hardwarezählern** zur Messung der CPU-Zyklen

- Zugriff bei ARM über Koprozessor 15, Mnemonics `MRC` und `MCR`*
- Berechnung der Ausführungszeit über Zykluszeit Δt bzw. Taktfrequenz $1/\Delta t$.
- Unterscheidung **Gesamtzeit** (inkl. Interrupts, Betriebssystem, Festplattenzugriffe) und **CPU**-Zeit (ohne Wartezeit auf E/A-Geräte)

Einflussfaktoren auf die Leistung eines Programms

Implementierung	Ausführungsumgebung
Algorithmus	Befehlssatzarchitektur
Programmiersprache	Mikroarchitektur
Compiler	Betriebssystem
Optimierungsstufe	sonstige Hardware

* in dieser Vorlesung nicht behandelt

Leistung und Leistungsabruf

- **Maximale Leistung** (*peak performance*)
  Theoretisches Maximum, unter Idealbedingungen
  kurzfristig erreichbar

- **Durchschnittliche Leistung** (*average performance*)
  Erwartungswert bei typischen Aufgaben

- **„Nachhaltige“ Leistung** (*sustained performance*)
  Unterschranke für die über den gesamten Lebenszyklus
  versprochene Leistung eines Gesamtsystems
  oder von Komponenten mit Verschleißeffekten (z. B. Flash-Speicher)

Methodische Schwierigkeiten

bei der Leistungsmessung

Beim **Vergleich von Implementierungen** auf definierter Architektur:
(d. h. Hardware und Systemsoftware)

- Granulare Messung einzelner Programmteile
  → Messfehler durch künstliche Aufrufbedingungen
  (Rückwirkungsabweichung)
- Grobe Messung ganzer Programmläufe
  → Messfehler durch Umgebungseinflüsse
  (z. B. Unterbrechungsanforderungen)

Beim **Vergleich von Architekturen:**

- Wahl vergleichbarer Aufgaben und Implementierungen
- Bei Datenabhängigkeit: Wahl der Testdaten

Weitere Unschärfe durch die Verbreitung von **Virtualisierungstechniken**

Benchmarks

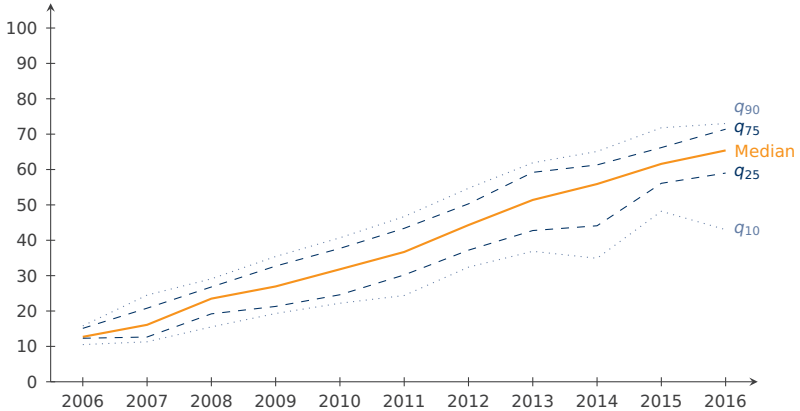
Spezialsoftware zur Leistungsmessung, simuliert typische Arbeitslast

- **Synthetische Benchmarks** messen Instruktionen pro Sekunde: z. B. Whetstone für Gleitkomma-, Dhrystone für Ganzzahloperationen
- **Kernels** sind ausgewählte Algorithmen, die sich als Benchmark etabliert haben: z. B. LINPACK
- **Mikrobenchmarks** sind auf die Messung einzelner Komponenten (z. B. Speicheranbindung, Grafik) spezialisiert
- **Benchmark-Suiten** enthalten verschiedene, gut portierbare Programme aus unterschiedlichen Anwendungsgebieten: z. B. Standard Performance Evaluation Corporation (SPEC) misst Leistungsverhältnis (aktuell in der Version 2017)

Leistungsentwicklung

SPEC 2006 Integer-Benchmarks

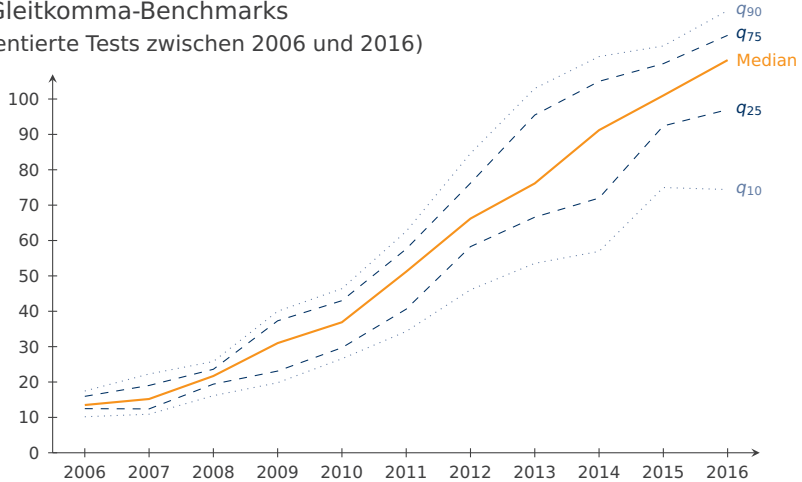
(8131 dokumentierte Tests zwischen 2006 und 2016)



Daten von www.spec.org/cpu2006/results

Leistungsentwicklung

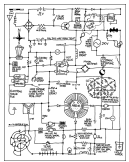
SPEC 2006 Gleitkomma-Benchmarks
(7972 dokumentierte Tests zwischen 2006 und 2016)



Daten von www.spec.org/cpu2006/results

Grundsätzliche Schwierigkeiten

bei der Leistungsmessung komplexer Systeme mit einfachen Kennzahlen



Projektion

42

Reduzierung von Komplexität

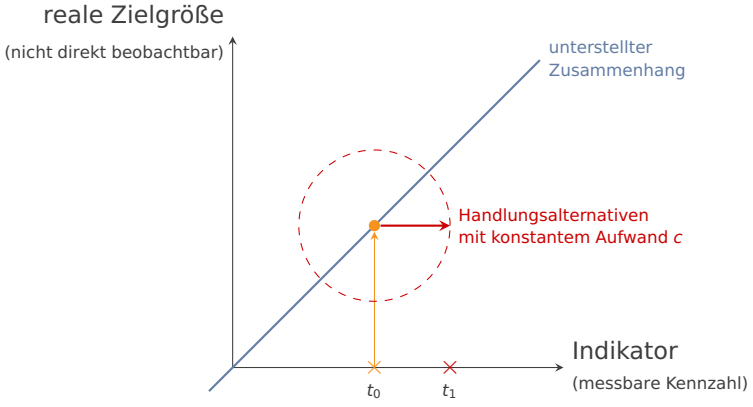
Realität **Kennzahl**

Reaktion

rationales Verhalten mit Kenntnis des Messverfahrens

vgl. z. B. Campbell 1976, Goodhart 1981; Illustration: xkcd.com

Rationale Reaktion: Überanpassung



Folge: Kennzahlen verlieren Aussagekraft, Verhaltensanreize verfehlen Ziel

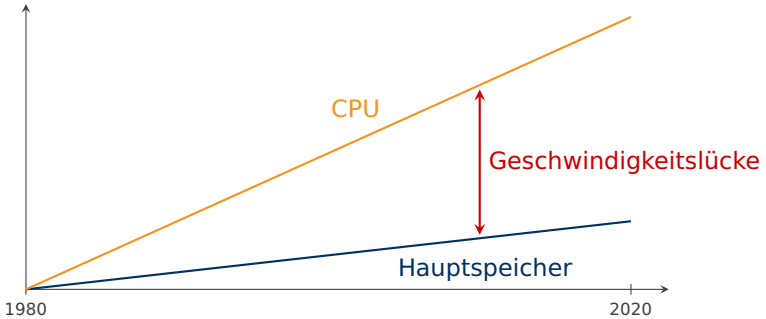
Gliederung

1. Leistungsmessung
2. **Caches: Optimierung des Speicherzugriffs**
3. Pipelines: Optimierung der CPU-Nutzung

Geschwindigkeitslücke (W)

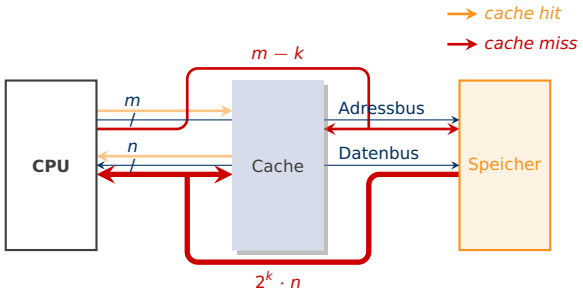
Seit 1980 wächst die Geschwindigkeit des . . .

- Hauptspeichers um 7 % pro Jahr
- Prozessors um 50 % pro Jahr



Anbindung des Cache an CPU und Speicher

Der Cache ist ein kleiner, schneller, mit SRAM realisierter Pufferspeicher.

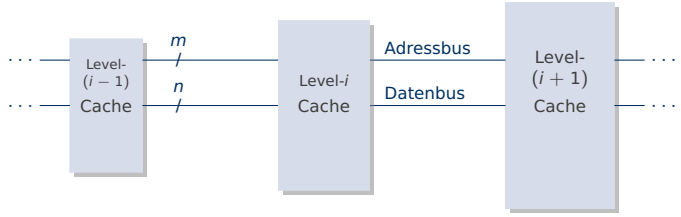


Vision

- Vorhersage, auf welche Speicherzelle die CPU als nächstes zugreift
- Daten schon im Vorfeld im Cache ablegen (*read ahead*)

Cache-Hierarchie

Ein oder mehrere Cache-Ebenen bleiben für die CPU **transparent.**



Lokalitätsprinzip

Ziel: Vorhersage der Adressen von wahrscheinlichen Lesezugriffen

- **Zeitliche Lokalität:** Hohe Wahrscheinlichkeit, dass eine Speicheradresse wiederholt verwendet wird (z. B. bei Schleifen)

  → Verwendete Adressen und Inhalte puffern.

- **Räumliche Lokalität:** Hohe Wahrscheinlichkeit, dass auf benachbarte Elemente zugegriffen wird (z. B. bei Arrays)

  → Alle benachbarten Speicherworte lesen und puffern.

  (Übertragung von Blöcken aus 2^k Worten im DRAM-Burst-Modus.)

Leistungssteigerung durch Cache-Einsatz

Trefferrate

$$h = \frac{\text{Anzahl } \textit{cache hits}}{\text{Anzahl } \textit{cache hits} + \text{Anzahl } \textit{cache misses}} \tag{1}$$

Sei

- c die Zugriffszeit auf den Cache — z. B. 5 ns
- r die Zugriffszeit auf den Hauptspeicher — z. B. 50 ns

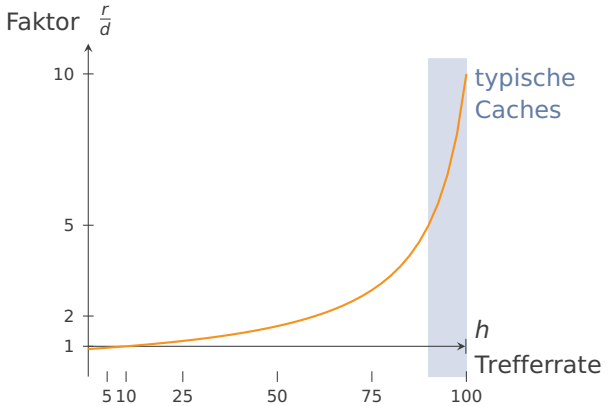
dann ist die **mittlere Speicherzugriffszeit**

$$d = c + (1 - h) \cdot r\,. \tag{2}$$

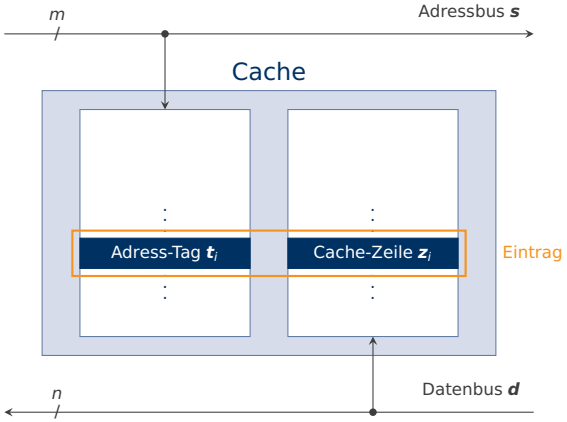
Nachteil: Schlechtere Vorhersagbarkeit des exakten Zeitverhaltens

Grafische Darstellung

der Leistungssteigerung durch Cache-Einsatz



Aufbau eines Caches



Cache-Verwaltung

Reinform 1: vollassoziativ

- Jeder Block des Hauptspeichers kann in jeder beliebigen Cache-Zeile abgespeichert werden.

$$\boldsymbol{z} \in \{0,1\}^{n \cdot 2^k}, k \in \mathbb{N} \quad \Rightarrow \quad \boldsymbol{t} \in \{0,1\}^{(m-k)}$$

- Verschiedene Verdrängungsstrategien beim Schreibzugriff
- Lesezugriff erfordert Vergleich mit allen Tags $\rightarrow$ aufwändig

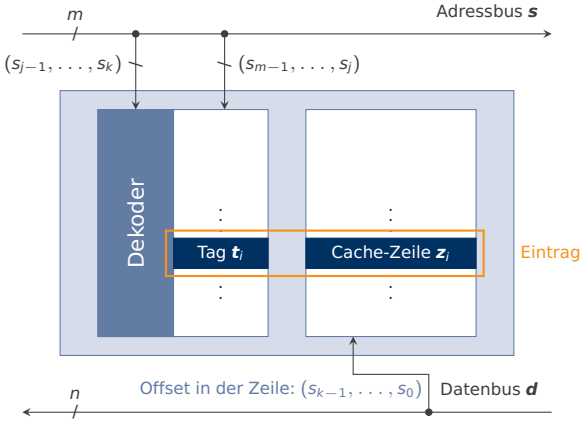
Reinform 2: direkt abgebildet (engl. *direct mapped*, DM)

- Die j niederwertigsten Adressbits definieren eine Cache-Zeile für jeden Block des Hauptspeichers. $j > k \quad \Rightarrow \quad \boldsymbol{t} \in \{0,1\}^{(m-j)}$
- Beim Lesezugriff Dekodierung und ein Vergleich

Mischformen z. B. zweifach assoziativer Cache

- Jeder Block des Hauptspeichers kann in einer von zwei definierten Cache-Zeilen abgespeichert werden.

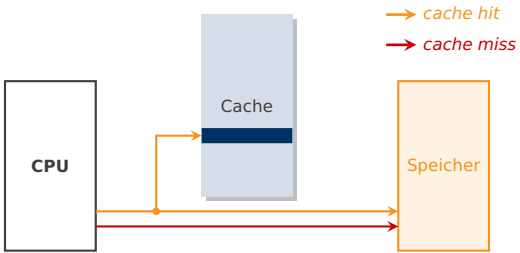
Aufbau eines Caches



Cache-Kohärenz

Sicherung der Konsistenz von Speicher und Cache nach **Schreibzugriff**

Write through-Methode

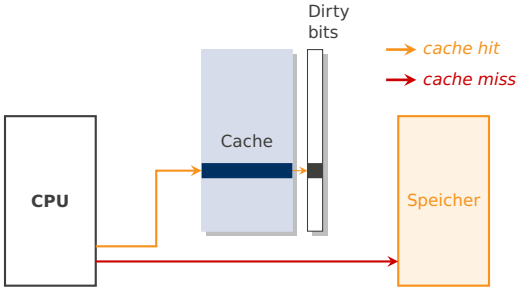


Sofortige Aktualisierung → keine Inkonsistenz

Cache-Kohärenz

Sicherung der Konsistenz von Speicher und Cache nach **Schreibzugriff**

Write back-Methode

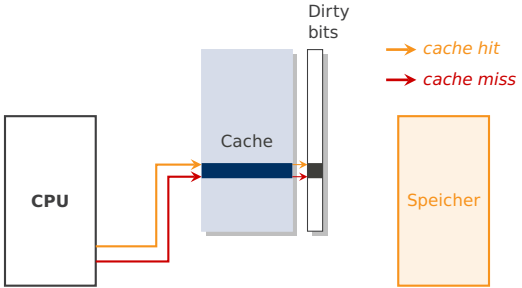


Bei einem Treffer wird nur der Cache aktualisiert. *Dirty bit* zeigt an, dass die Zeile bei Verdrängung zurückgeschrieben werden muss.

Cache-Kohärenz

Sicherung der Konsistenz von Speicher und Cache nach **Schreibzugriff**

Write allocation-Methode



Jeder Schreibzugriff wird zunächst im Cache gepuffert. Es kann zur Verdrängung beim Schreiben kommen.

Hörsaalfrage

24 82 94 16

Welche Cache-Architektur verspricht mehr Leistung?

a. **Cache 1:** 64 KB vollassoziativer, in die CPU integrierter SRAM-Cache mit Trefferrate $h = 92\,\%$ und Zugriffszeit 3 ns. Länge der Cache-Zeile: 16 Byte.
b. **Cache 2:** 4 MB direkt abgebildeter SRAM-Cache zwischen CPU und Hauptspeicher mit Trefferrate $h = 95\,\%$ und Zugriffszeit 12 ns. Länge der Cache-Zeile: 512 Byte.

In beiden Fällen besteht der Hauptspeicher aus schnellem DRAM mit Zugriffszeit 40 ns.

Zugang: `https://arsnova.uibk.ac.at` mit Zugangsschlüssel **24 82 94 16**. Oder scannen Sie den QR-Kode.

Ansätze zur Cache-„optimierten“ Programmierung

Ursachen für Cache-Fehlzugriffe (*cache misses*)

- Erstbelegung nach Programmstart → *compulsory*
- Verdrängung benötigter Zeilen mangels Kapazität → *capacity*
- Verdrängung benötigter Zeilen durch Konflikte → *conflict*

Maßnahmen zur Erhöhung der Trefferrate

1. Rechtzeitiges Holen durch explizite **Prefetch-Anweisungen** (durch Compiler oder manuell im Assemblerprogramm)
2. **Vermeidung von Konflikten** durch Füllworte (*padding*, oft durch Compiler) und Wahl der Dimensionen von Feldern in Zweierpotenzen
3. **Erhöhung der Lokalität** beim Zugriff auf Daten

Beispiele in C

Datenlayout

Vorher

```
1 char *name[1000];
2 int  matrikelnr[1000];
```

Nachher

```
3 struct student {
4   char *name;
5   int  matrikelnr;
6 } hoerer[1024];
```

Schleifenaustausch

Vorher

```
7 for (j=0;j<100;j=j+1)
8   for (i=0;i<5000;i=i+1)
9     x[i][j] = 2*x[i][j];
```

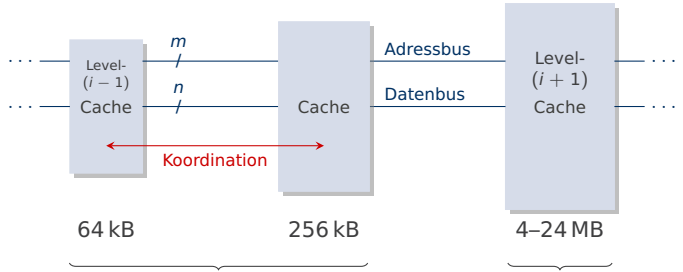
Nachher

```
10 for (i=0;i<5000;i=i+1)
11   for (j=0;j<100;j=j+1)
12     x[i][j] = 2*x[i][j];
```

Cache-Hierarchie

Ein oder mehrere Cache-Ebenen bleiben für die CPU **transparent.**

Beispiel: Intel Nehalem-Mikroarchitektur (2008)

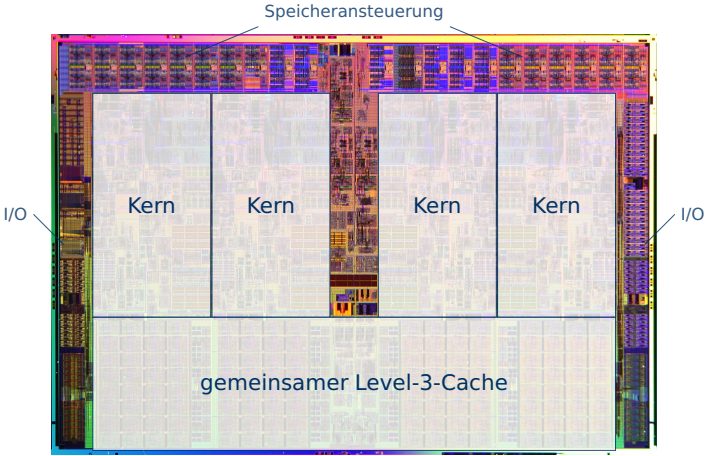


- keine Duplikate
- „Absinken“ bei Verdrängung

- unabhängig

Nutzung der Die-Fläche

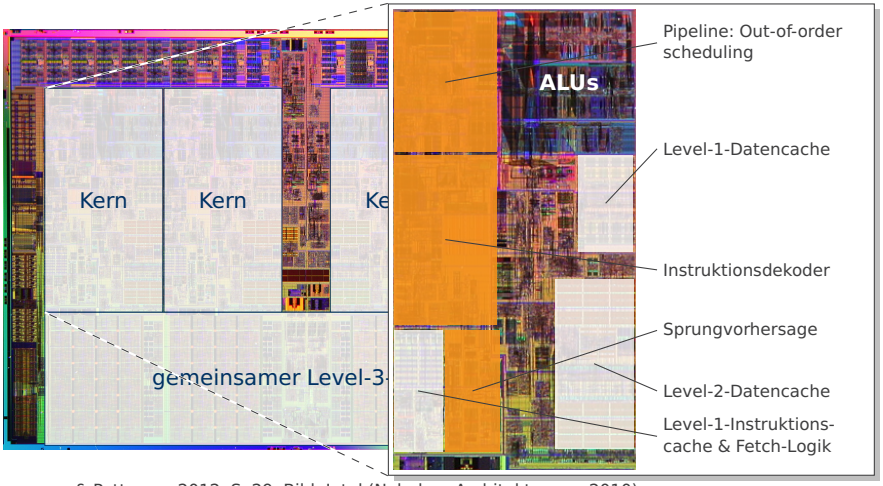
bei einem Mehrkernprozessor



nach Hennessy & Patterson 2012, S. 29; Bild: Intel (Nehalem-Architektur von 2010)

Nutzung der Die-Fläche

bei einem Mehrkernprozessor



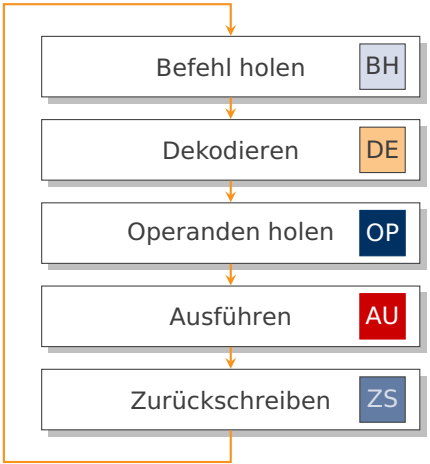
nach Hennessy & Patterson 2012, S. 29; Bild: Intel (Nehalem-Architektur von 2010)

Gliederung

1. Leistungsmessung
2. Caches: Optimierung des Speicherzugriffs
3. **Pipelines: Optimierung der CPU-Nutzung**

Maschinenbefehlszyklus

Modellablauf in Steuer- und Rechenwerk pro Befehl (hier: RISC)



Speicheradresse aus `pc`; Inhalt holen (*fetch*); `pc` erhöhen

Instruktionskode und Operanden ermitteln (Wert, Register, Adresse)

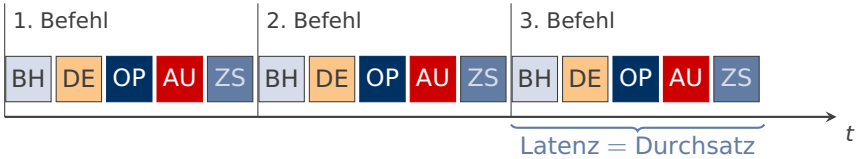
Register oder Speicher auslesen

Rechenwerk führt Instruktion auf den Operanden aus

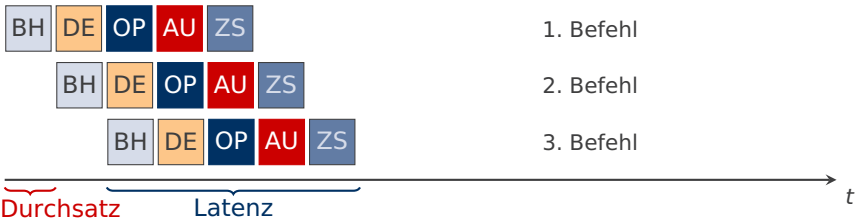
Ergebnis in Speicher oder Register zurückschreiben

Prinzip des Pipelinings

Sequenzielle Abarbeitung der Teilschritte



Überlappendes Abarbeiten



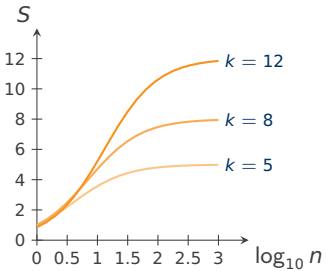
Pro Taktzyklus wird ein Maschinenbefehl abgeschlossen.

Theoretische Leistungssteigerung

Annahmen ideale k-stufige Pipeline, Programm mit n Befehlen, Zykluszeit Δt, d. h. Taktfrequenz = $1/\Delta t$

Berechnung der **Beschleunigung**

$$S(k) = \frac{\text{Ausführungszeit ohne Pipeline}}{\text{Ausführungszeit mit Pipeline}} = \frac{n \cdot k \cdot \Delta t}{[n + (k-1)] \cdot \Delta t} \tag{3}$$



$$\lim_{n \to \infty} S(k) = \frac{n \cdot k}{n + (k-1)} = k \tag{4}$$

Gründe für Abweichungen in der Praxis

Strukturkonflikte *structural hazard*

- Hardware unterstützt Befehlskombination nicht
- **Lösung:** Umordnen (durch Compiler), sonst Leertakte (`NOP`s)

Datenkonflikte *data hazard*

- Operanden noch nicht verfügbar
- **Lösung:** dezidierte *Forwarding*-Logik, sonst wie oben

Steuerkonflikte *control hazard*

- Geholter Befehl ist nicht der benötigte (nach Verzweigungen)
- **Lösung:** Sprungvorhersage (*branch prediction*), Anhalten

Operationen mit mehreren Zyklen

- Multiplikation, Division, Gleitkommaoperationen, Kontextwechsel
- **Lösung:** separate Gleitkomma-Pipeline, (Geduld)

Beispiel zur Vermeidung von Datenkonflikten

Aufgabe

$$a = b + c$$
$$d = e - f$$

Langsamer Kode

```
LDR   r2, b
LDR   r3, c
ADD   r1, r2, r3
STR   r1, a

LDR   r5, e
LDR   r6, f
SUB   r4, r5, r6
STR   r4, d
```

Schnellerer Kode

```
LDR   r2, b
LDR   r3, c
LDR   r5, e
ADD   r1, r2, r3

LDR   r6, f
STR   r1, a
SUB   r4, r5, r6
STR   r4, d
```

Annahme: Variablen `a`–`f` befinden sich im Speicher relativ zu `sp` oder `pc` adressierbar.

Nächste Woche

1. Klausurtermin **hilfsweise als Online-Prüfung:** 27. Jänner 2021, 12:15–12:45 Uhr

Organisatorisches

- Die Prüfung besteht aus 15 Fragen in der Form der Proseminar-Tests.
- Gesamtbearbeitungszeit: 30 Minuten
- Zum Bestehen benötigen Sie 50 % der erreichbaren Punkte. Raten lohnt sich nicht.
- Die Prüfung findet im OLAT-Kurs der 2020W703063 **VO** Rechnerarchitektur statt.
- Dort wird am Tag vor der Prüfung ein Probetest erscheinen, den alle korrekt angemeldeten KandidatInnen sehen. Bitte probieren Sie das aus.
- Probleme/Fragen ausschließlich an rechnerarchitektur-informatik@uibk.ac.at.
- Lesen Sie die Mitteilungen im OLAT-Kurs der Vorlesung.

Zulässige Hilfsmittel

- Internet-Browser zum Zugriff auf OLAT
- ARM-Befehlsreferenz (verfügbar als arm.pdf im OLAT)

Hinweise zur Präsenzklausur

Gilt nicht für die Online-Prüfung am ersten Termin !

Meine Prüfungsphilosophie

- Verständnis von Konzepten prüfen, nicht die Kapazität Ihres Kurzzeitgedächtnisses
- Ihre Antworten werden von Menschen korrigiert: Falls Sie Annahmen zur Lösung brauchen, schreiben Sie diese deutlich dazu.

- Raten oder Mehrdeutigkeit zahlen sich nicht aus.
- Allein mit Auswendiglernen erreichen Sie ca. 25 % der Punkte.
- Ohne ARM-Kenntnisse verschenken Sie ca. 25 % der Punkte.

Weitere Informationen finden Sie auf unserer **Webseite** unter http://informationsecurity.uibk.ac.at/teaching/ .

Hinweise zur Präsenzklausur (Forts.)

Gilt nicht für die Online-Prüfung am ersten Termin !

Organisatorisches

- Warten Sie vor dem Hörsaal bis Sie aufgerufen werden.
- Gemeinsamer Beginn, Bearbeitungszeit 75 Minuten
- Identitätsprüfung mit gültigem **Studentenausweis und weiterem amtlichen Lichtbildausweis**
- Alle zu bewertenden Antworten ausschließlich auf Klausurpapier
- Ergebnisbekanntgabe im OLAT anhand von Klausur-ID

Zulässige Hilfsmittel

- Stift, Geodreieck, nicht-programmierbarer Taschenrechner
- ARM-Referenz wird bereitgestellt (identisch wie im OLAT)

Sonst keine elektronischen Geräte am Platz !
(Ausschalten reicht nicht.)

Syllabus – Wintersemester 2020/21

07.10.20	1.	Einführung
14.10.20	2.	Kombinatorische Logik I
21.10.20	3.	Kombinatorische Logik II
28.10.20	4.	Sequenzielle Logik I
04.11.20	5.	Sequenzielle Logik II
11.11.20	6.	Arithmetik I
18.11.20	7.	Arithmetik II
25.11.20	8.	Befehlssatzarchitektur (ARM) I
02.12.20	9.	Befehlssatzarchitektur (ARM) II
09.12.20	10.	Prozessorarchitekturen
16.12.20	11.	Ein-/Ausgabe
13.01.21	12.	Speicher
20.01.21	13.	Leistung
27.01.21		**Klausur (1. Termin)**