

# Kurs „Rechnerarchitektur“ (RA) im SS 2011

Peter Marwedel  
Informatik 12  
peter.marwedel@tu..  
Tel.: 0231 755 6111  
Sprechstunde: Mo, 13-14

Ramin Yahyapour  
ITMC  
ramin.yahyapour@tu..

2011/03/30

*Willkommen!*

Diese Folien enthalten Graphiken mit  
Nutzungseinschränkungen. Das Kopieren der  
Graphiken ist im Allgemeinen nicht erlaubt.

---

# Vortragende

---

Gemeinsames Kursangebot von

- P. Marwedel (1. Hälfte)
- R. Yahyapour (2. Hälfte)

Vorteile

- Betrachtung der Architektur eines einzelnen Rechners (PM)
- Globale Sicht auf Rechenzentren, Netze;  
Berufsperspektiven von Informatiker(innen), die große  
Rechensysteme nutzen (RY)

---

# 1.1 Gegenstand des Kurses RA

---

Fortgeschrittene Konzepte von Rechensystemen,

- d.h. der Architektur von Rechnern
- und deren Einbettung in Systeme:  
Rechner in Netzen, im Verbund, ..

Bezug zum Kurs Rechnerstrukturen (RS):

- RA baut auf RS auf.
- Stoff aus RS wird im Wesentlichen als bekannt vorausgesetzt (Gatter, Binäre Logik, Zahlenrepräsentation, ...).
- Stoff aus RS wird vertieft (*multi-cores, grid, cloud*, Netze, ...).

---

# Gegenstand des Kurses RA

## - Definitionen von „Rechnerarchitektur“ -

---

**Def.** (nach Stone): *The study of computer architecture is the study of the **organization and interconnection of components** of computer systems. Computer architects construct computers from **basic building blocks** such as memories, arithmetic units and buses.*

*From these building blocks the computer architect can construct **anyone of a number of different types of computers**, ranging from the smallest hand-held pocket calculator to the largest ultra-fast super computer. The functional behaviour of the components of one computer are similar to that of any other computer, whether it be ultra-small or ultra-fast.*

---

# Gegenstand des Kurses RA

## - Definitionen von „Rechnerarchitektur“ -

---

Nach Stone ..

*By this we mean that a memory performs the storage function, an adder does addition, and an input/output interface passes data from a processor to the outside world, regardless of the nature of the computer in which they are embedded.*

*The major differences between computers lie in the way the modules are connected together, and the way the computer system is controlled by the programs. **In short, computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks.***

---

# Gegenstand des Kurses RA

## - Definitionen von „Rechnerarchitektur“ -

---

**Def.** (nach Amdahl, Blaauw, Brooks):

*The term architecture is used here to describe the attributes of a system as seen **by the programmer**, i.e., the conceptual structure and functional behaviour, as distinct from the organization and data flow and control, the logical and the physical implementation.*

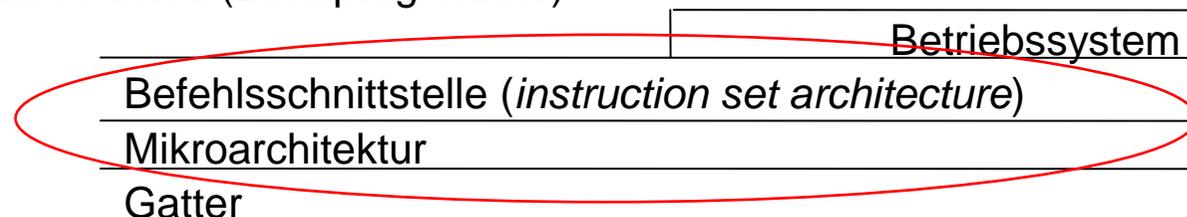
# Gegenüberstellung der Definitionen

Programmierschnittstelle	Interner Aufbau
Externe Rechnerarchitektur	Interne Rechnerarchitektur
Architektur	Mikroarchitektur
Rechnerarchitektur	Rechnerorganisation

Die externe Rechnerarchitektur definiert

- Programmier- oder Befehlssatzschnittstelle
- engl. *instruction set architecture, (ISA)*
- eine (reale) Rechenmaschine bzw.
- ein *application program interface (API)*.

Executables (Binärprogramme)



---

# Themenüberblick

---

1. Einleitung
2. Programmiermodelle (*instruction set architectures* (ISAs))
  - RISC, CISC,
  - DSP, Netzwerk,
  - Graphikprozessoren, EPICs, ...
3. Mikroarchitektur
  - Realisierung von Arithmetik
  - Performanzsteigerung
4. Speicher
  - Speicherhierarchie, Flashspeicher
5. Mehrprozessorsysteme
  - Typen von Parallelrechnern, Synchronisation,
  - Caching, .....

5% Überlappung mit  
„Eingebettete Systeme“

---

# Wieso ist Verständnis von Rechnerarchitektur wichtig?

---

**Zentral:** Möglichkeiten und Grenzen des „Handwerkszeugs“ eines Informatikers einschätzen können!

Grundverständnis wird u.a. benötigt bei:

- bei der Geräteauswahl,
- bei der Fehlersuche,
- bei der Leistungsoptimierung / Benchmarkentwürfen,
- bei Zuverlässigkeitsanalysen,
- beim Neuentwurf von Systemen,
- bei der Codeoptimierung im Compilerbau,
- bei Sicherheitsfragen.

Keine groben Wissenslücken in zentralen Bereichen der IT!

---

# Gliederung (heute)

---

- Gegenstand des Kurses:  
*Was ist Rechnerarchitektur?*
- Bewertung von Rechnern
- Organisatorisches  
Materialien zum Kurs, Übungen,  
Leistungsnachweis
- Befehlssätze: RISC und CISC



---

## 1.2 Bewertung von Rechnern

---

### Mehrere Kriterien

- Funktional: Befehlssatz, Speichermodell, Interruptmodell
- Preis
- Energieeffizienz (geringe elektrische Leistung)
- „Performanz“: (durchschnittliche) „Rechenleistung“ zur Abgrenzung von der benötigten elektrischen Leistung hier Bevorzugung von „Performanz“ oder *Performance*
- Realzeitverhalten (*timing predictability*)
- Erweiterbarkeit
- Größe/Gewicht
- Zuverlässigkeit
- Sicherheit, ....



Standardmäßig betont.  
**Wir** wollen die anderen  
Kriterien nicht ignorieren.

---

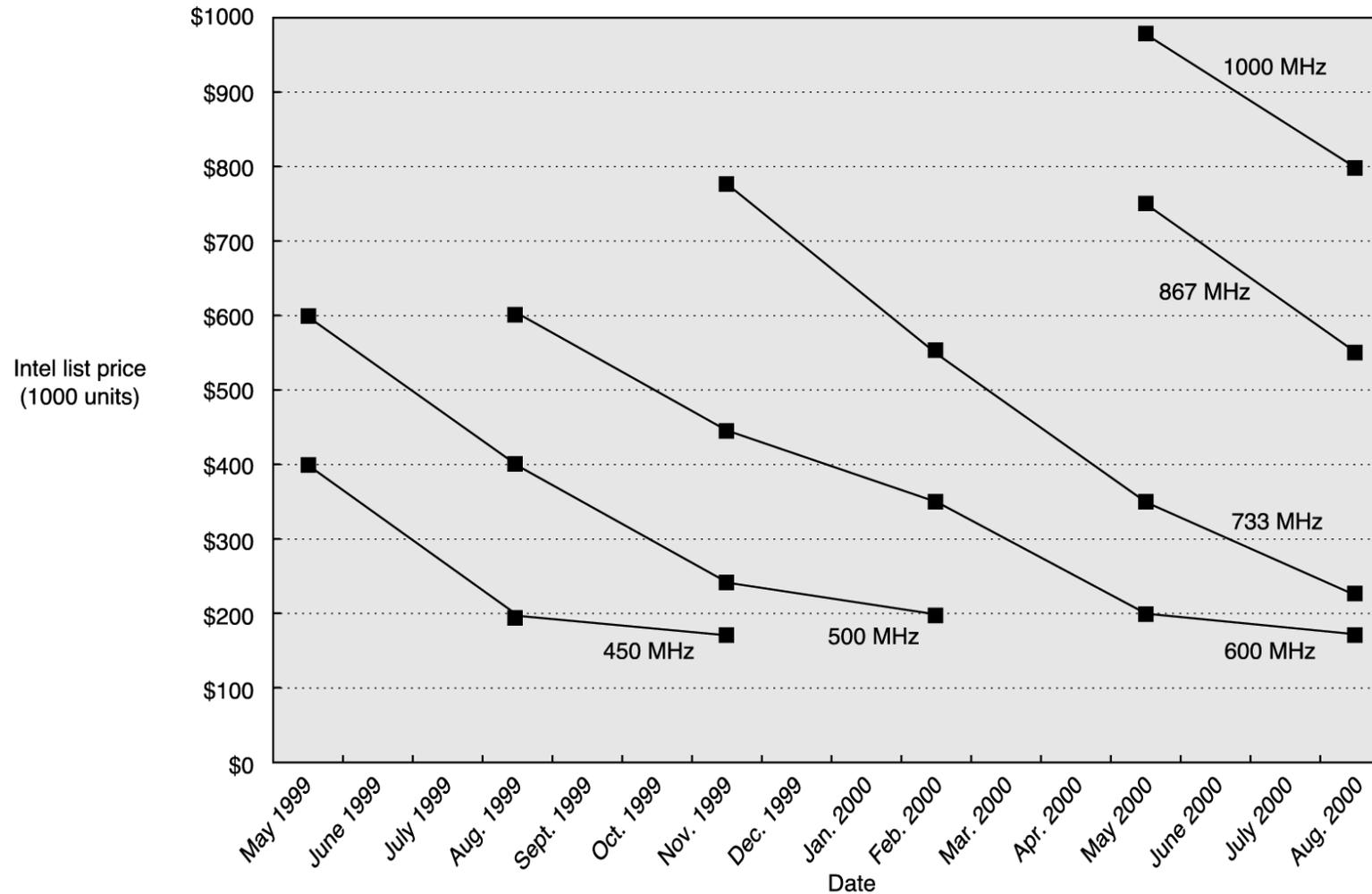
# Funktionale Eigenschaften

---

Die Funktion von Befehlssätzen kann wieder nach mehreren Kriterien bewertet werden:

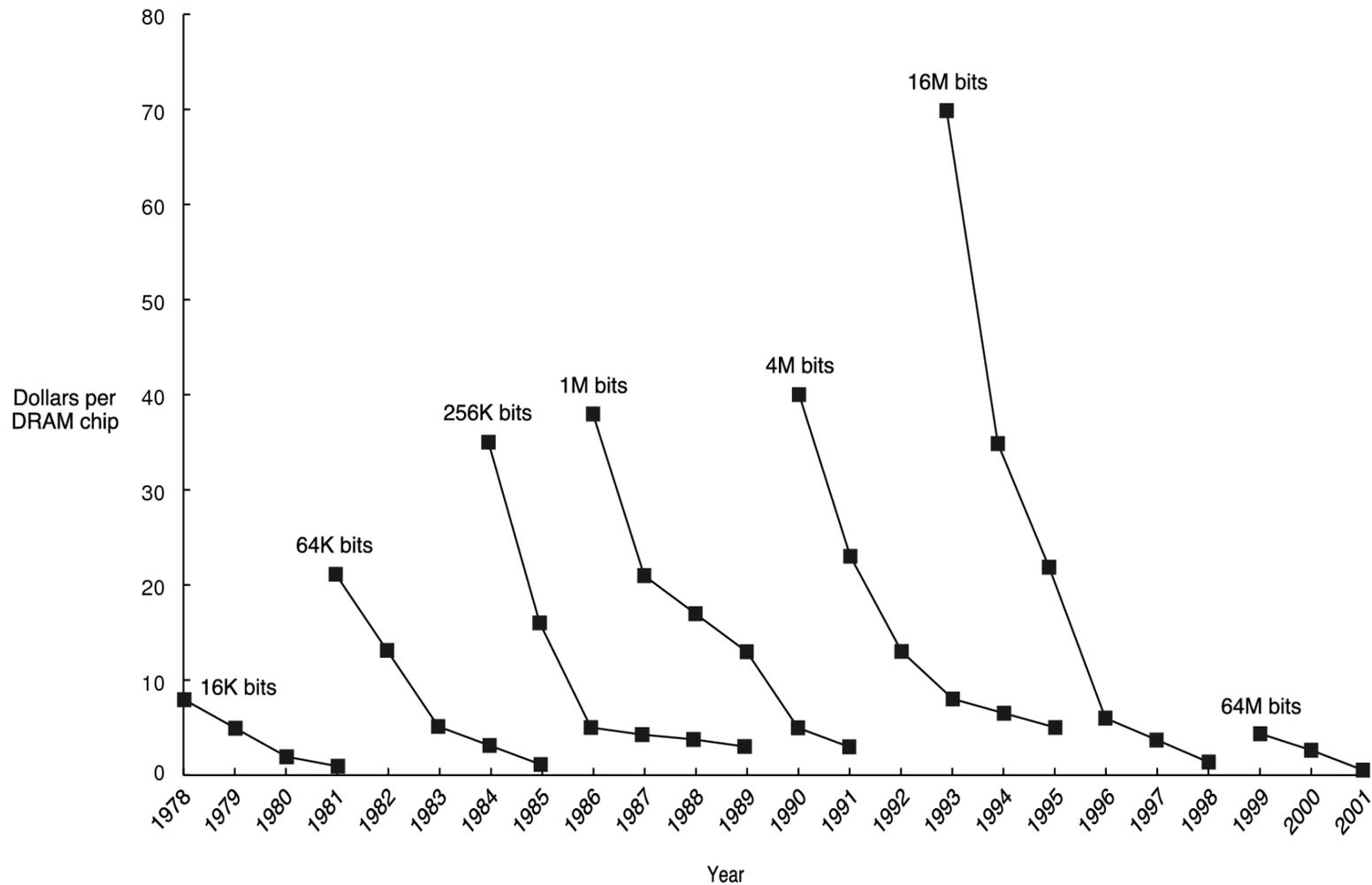
- Operationsprinzip (Von-Neumann, Datenfluss, ...)
- Addressbereiche (4 GB, usw.)
- Byte-Addressierbarkeit
- *Endianness*
- Orthogonalität
- $n$ -Adressmaschine
- ...

# Preisentwicklung bei Mikroprozessoren



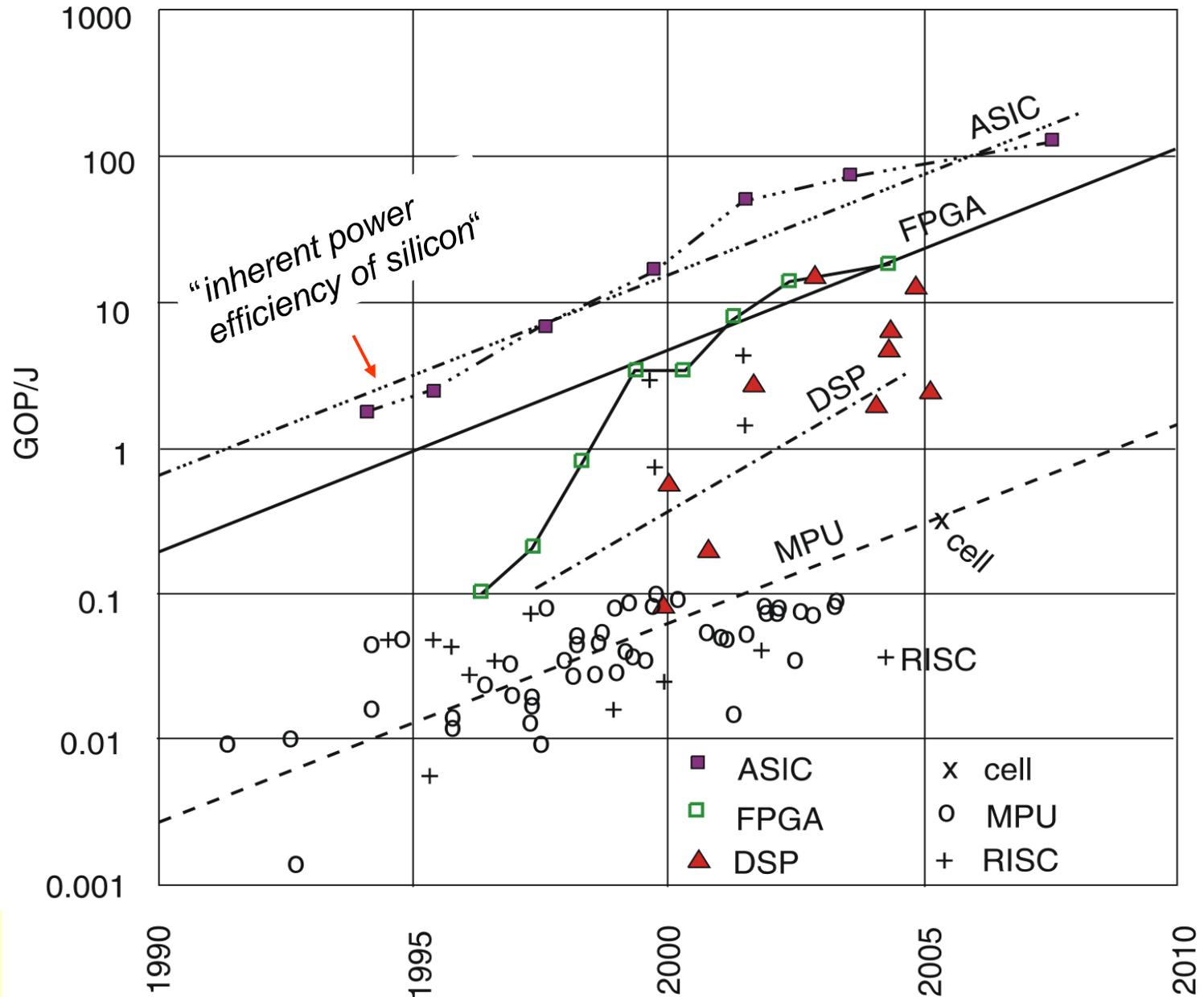
© Elsevier Science (USA). All rights reserved

## ... und bei Speichermodulen



© Elsevier Science (USA). All rights reserved

# Energieeffizienz



© Hugo De Man,  
IMEC, Philips, 2007

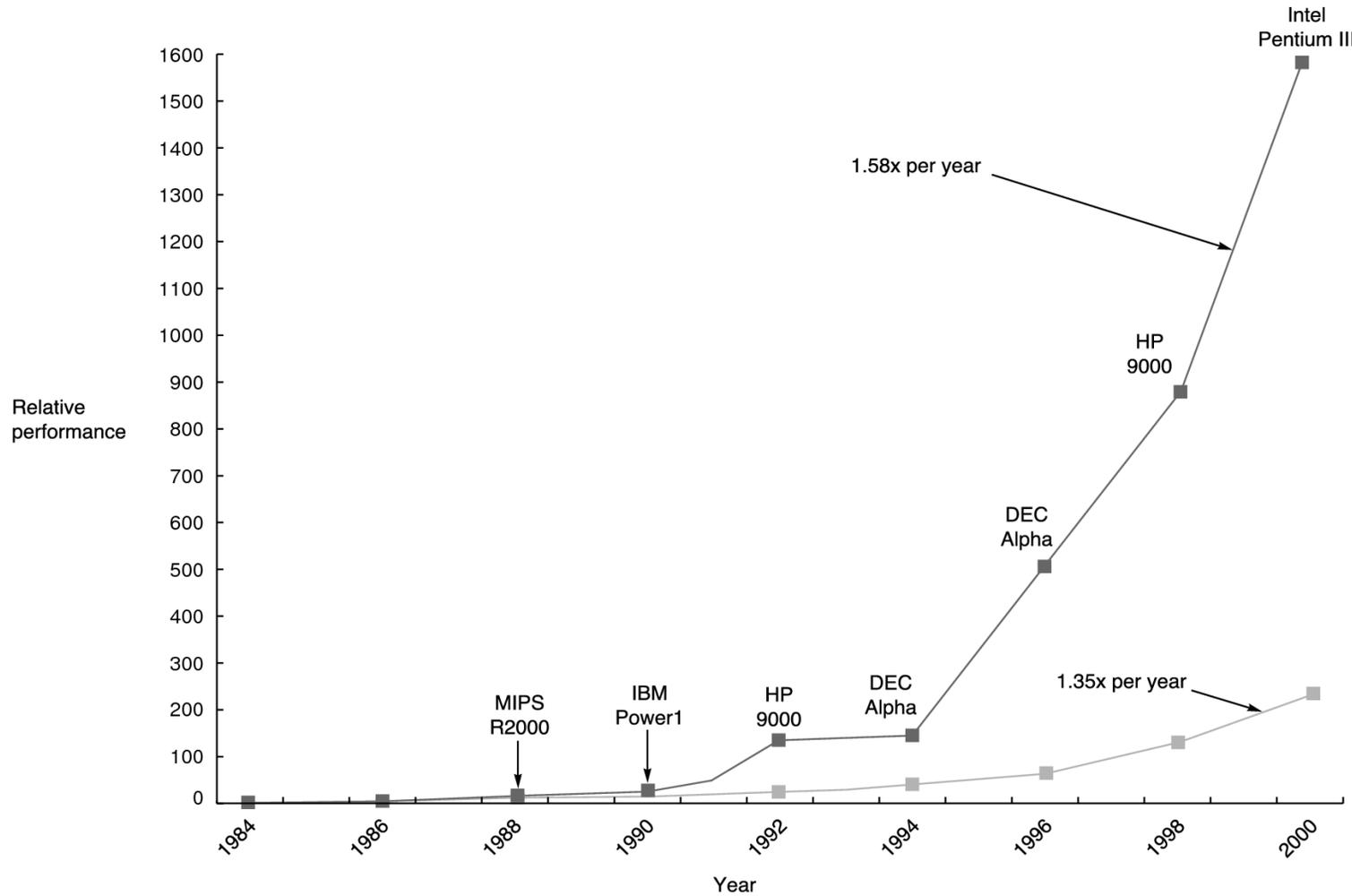
---

# Performanz

---

- Das Mooresche Gesetz (nach Gordon Moore, Mitbegründer von Intel, 1965)  
„Die Anzahl der auf einem Chip integrierten Transistoren verdoppelt sich alle 18 Monate!”
- Anforderungen aus der Software: Nathans erstes Softwaregesetz (nach Nathan Myhrvold, Microsoft)  
„Software ist ein Gas. Es dehnt sich aus und füllt den Behälter, in dem es sich befindet.”
- Anforderungen aus Anwendungen in der Telekommunikations- und Netzwerktechnik, *Video-on-Demand*, *Multi-Media-Messaging*, mobiles Internet

# Performanzentwicklung bei Mikroprozessoren



© Elsevier Science (USA). All rights reserved

---

# Beurteilung der Performanz bzw. „Rechenleistung“

---

Was bedeutet „Rechner  $A$  ist schneller als Rechner  $B$ “?

- Benutzersicht: Antwortzeit  
(bei Bearbeitung einer Aufgabe)
- Serviceanbietersicht: Durchsatz  
(Anzahl Aufgaben, die pro Zeiteinheit bearbeitet werden)

Zentrale Messgröße für **beide Sichten: Ausführungszeit!**

---

# Performance-Maße

---

Verschiedene Definitionen von Ausführungszeit:

- **Laufzeit** (*wall-clock time / elapsed time*)  
= Gesamtlaufzeit inkl. I/O, Speicherzugriffen,  
Betriebssystemoverhead, ggf. weitere Systemlast  
Hinweis: Wartezeiten im Mehrprogrammbetrieb von  
anderen Prozessen nutzbar
- **CPU-Zeit**
  - Wartezeiten / Bearbeitung anderer Prozesse werden  
*nicht* berücksichtigt
- **user CPU-Zeit**
  - Nur Programmabarbeitung, *nicht* Betriebssystemdienste  
z.B. unter Unix: `time <Kommando> ...`  
0.09u 0.07s 0:01.74 9.1%

---

# Performance-Maße II

---

Sinnvolle *Performance*-Definitionen:

- *System-Performance*, d.h. Laufzeit in einem unbelasteten (*unloaded*) System (d.h. kein Mehrprogrammbetrieb)
- *CPU-Performance*, d.h. [*user*] CPU-Zeit (Betrachtung unabhängig von I/O [und Betriebssystem])

Wovon Lauf-/CPU-Zeiten messen?

**Immer reale Programme!**

Achtung: Es existieren *Performance*-Definitionen, die nicht auf Zeitmessung bzw. Ausführung realer Programme basieren!

---

# Programmauswahl zur *Performance*-Bewertung

---

Problem: Aufgabe von Rechnern selten eindeutig definiert (i.d.R. nicht: „Ein bestimmtes Programm wird immer ausgeführt“)

☞ Performanz im realen Betrieb muss vorhergesagt/geschätzt werden!

*Dhrystone does not use floating point. Typical programs don't ...*

(R. Richardson, '88)

*This program is the result of extensive research to determine the instruction mix of a typical Fortran program. The results ... on different machines should give a good indication of which machine performs better under a typical load of Fortran programs. The statements are purposely arranged to defeat optimizations by the compiler.*

(H.J.Curnow & B.A.Wichmann, '76)

---

---

# Organisatorisches

---

# Vorlesung

---

- Termin: Di 12-14 und Do 10-12
- Raum: OH 14, E23

---

# Übungen

---

## Termine

Nr.	Tag	Zeit	Ort	
1	Mo	12:30-14:00	GB IV, R. 318	
2	Mo	14:15-15:45	GB IV, R. 318	
3	Fr	12:30-14:00	OH14, E02	
4	Fr	14:15-15:45	OH14, E02	

## Betreuung

- Heiko Falk, Informatik 12, heiko.falk@tu.. (z.Zt. krank)
- Timon Kelter, Informatik 12, timon.kelter@tu..

## Anmeldung:

- Tabelle in der Vorlesung,
- anschließend Aushang im Eingangsbereich OH16

---

# Zeitplan für die Bearbeitung der Übungen

---

- Ausgabe jeweils donnerstags, 10:00,
- Abgabe: mittwochs 12:00,
  - Briefkasten, OH 16, EG
  - Im Einzelfall per Internet nach Rücksprache mit Ü-leiter
- Besprechung: ab freitags nach der Abgabe

---

# Materialien zur Vorlesung

---

## Web:

- Home page: <http://ls12-www.cs.tu-dortmund.de/de/teaching/courses/ss11/ra>

## Folien

- Teil 1 (z. Zt. vorläufig): <http://ls12-www.cs.tu-dortmund.de/de/teaching/courses/ss11/ra/slides/index.html>
- Teil 2: EWS

## Skript

- Teil 1: <http://ls12-www.cs.tu-dortmund.de/teaching/courses/ss11/ra/download/main.pdf>

## Bücher

- Hennessy, John L., Patterson, David A.: *Computer Architecture – A Quantitative Approach*, Morgan Kaufman, 4. Auflage 2006.

---

# Prüfung / Leistungsnachweis

---

- Probeklausur mindestens für Teil 1 geplant
- Prüfung (DPO 2001, Bachelor): Klausur über Inhalte von Vorlesung und Übungen (benotet), 9 Credits:
  - 29.7.2011, ab 8:00,
  - Nachklausur: 12.9.2011, ab 8:00
- Leistungsnachweis (unbenotet, 9 Credits): Bearbeitung der Übungsaufgaben (mindestens **45%** der Gesamtpunktzahl erforderlich) + **Präsentation** ausgewählter Lösungen.
- Sonderregelungen für Nebenfachstudierende: wenn *zwingend* erforderlich nach Absprache

---

# Forum

---

- <http://inpud.cs.uni-dortmund.de/phpbb/>

---

---

## 2. Programmiermodelle (*instruction set architectures (ISAs)*)

---

## 2.1 RISC und CISC

### - *Reduced instruction set computers (RISC) (1)*-

---

Wenige, einfache Befehle wegen folgender Ziele:

- Hohe Ausführungsgeschwindigkeit
  - durch kleine Anzahl interner Zyklen pro Befehl
  - durch Fließbandverarbeitung (siehe Kap. 3)

**Def.:** Unter dem **CPI-Wert** (engl. ***cycles per instruction***) einer Menge von Maschinenbefehlen versteht man die mittlere Anzahl interner Bus-Zyklen pro Maschinenbefehl.

RISC-Maschinen: CPI möglichst nicht über 1.

CISC-Maschinen (s.u.): Schwierig, unter CPI = 2 zu kommen.

Programmlaufzeit = Dauer eines Buszyklus \* Anzahl der auszuführenden Befehle \* CPI-Wert des Programms

Wiederholung aus RS

---

# Klassifikation von Befehlssätzen

## - *Reduced instruction set computers (RISC) (2)*-

---

Eigenschaften daher:

- **feste Befehlswortlänge**
- **LOAD/STORE-Architektur**  
Operanden von Arithmetik-Befehlen immer in Registern!
- **einfache Adressierungsarten**
- **"semantische Lücke"** zwischen Hochsprachen & einfachen  
Assemblerbefehlen wird  **durch Compiler überbrückt.**
- **Statt** aufwändiger  **Hardware** zur Beseitigung von  
Besonderheiten (z.B. 256 MB-Grenze bei MIPS, 16 Bit  
Konstanten) wird diese Aufgabe der  **Software** übertragen.
- **Rein in Hardware realisierbar** („mit Gattern und Flip-  
Flops“)

Wiederholung aus RS

---

# ***Complex instruction set computers (CISC) (1)***

---



## ***Complex instruction set computers (CISC)***

Entstanden in Zeiten schlechter Compiler & großer  
Geschwindigkeitsunterschiede Speicher / Prozessor

- 👉 Befehle sollten möglichst nahe an den Hochsprachen  
sein (keine semantische Lücke)
- 👉 Mit jedem geholten Befehl sollte der Prozessor viel tun
  - 👉 sehr komplexe Befehle

*Wiederholung aus RS*

# Complex instruction set computers (CISC)

## Beispiel MC680x0 (1)

Beispiel: Motorola 68000 (erster Prozessor der 680x0-Serie)

Format des Kopierbefehls MOVE:



Opcode	Größe	Ziel		Quelle	
"00"	"01"=Byte, "11"=Wort, "10"=Doppelwort (32 Bit)	Register	Modus	Register	Modus

bis zu 4 Erweiterungsworte zu je 16 Bit

Viele komplexe Adressierungsarten schon in den ersten Prozessoren der Serie.

In Form der ColdFire-Prozessoren weiterhin eingesetzt.

Wiederholung  
aus RS

# Complex instruction set computers (CISC)

## Beispiel MC680x0 (2)

Modus	Register feld	Erweit.	Notation	Adressierung
"000"	n	0	Dn	Register-Adressierung
"001"	n	0	An	Adressregister-Adressierung
"010"	n	0	(An)	Adressregister indir.
"011"	n	0	(An)+	Adressreg. indirekt.mit <i>postincrement</i>
"100"	n	0	-(An)	Adressreg. indirekt.mit <i>predecrement</i>
"101"	n	1	d(An)	Relative Adressierung mit 16 Bit Distanz
"110"	n	1	d(An,Xm)	Register-relative Adressierung mit Index
"111"	"000"	1	d	direkte Adressierung (16 Bit)
"111"	"001"	2	d	direkte Adressierung (32 Bit)
"111"	"010"	1	d(*)	Programmzähler-relativ
"111"	"011"	1	d(*,Xn)	Programmzähler-relativ mit Index
"111"	"100"	1-2	#zahl	unmittelbare Adressierung

Wiederholung aus RS

---

# ***Complex instruction set computers (CISC)*** **- Eigenschaften -**

---

Relativ kompakte Codierung von Programmen

Für jeden Befehl wurden mehrere interne Zyklen benötigt

☞ Die Anzahl der Zyklen pro Befehl (der ***cpi***-Wert) war groß  
(Mikro-) Programm zur Interpretation der Befehle nötig  
Compiler konnten viele Befehle gar nicht nutzen

*Wiederholung  
aus RS*

---

# Zusammenfassung

---

- Definitionen zur Rechnerarchitektur
- Bewertung von Rechnern
- Organisatorisches
- ISAs
  - RISC, CISC