

Wolfgang E. Nagel  
Center for Information Services and High Performance Computing (ZIH)

# Rechnerarchitektur II

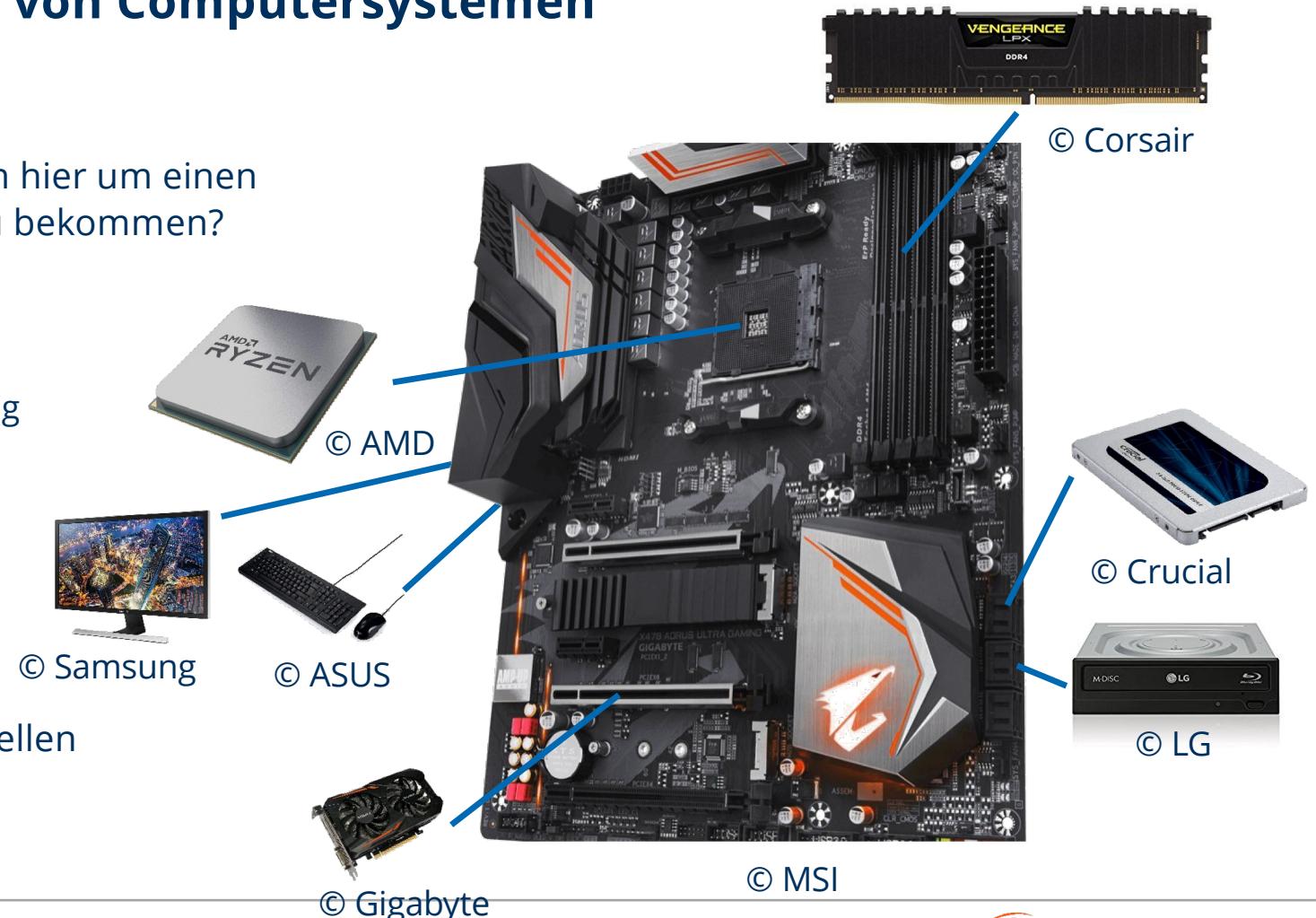
Vorlesung 3: Aufbau und Entwicklung von Prozessoren am Beispiel x86



# Prinzipieller Aufbau von Computersystemen

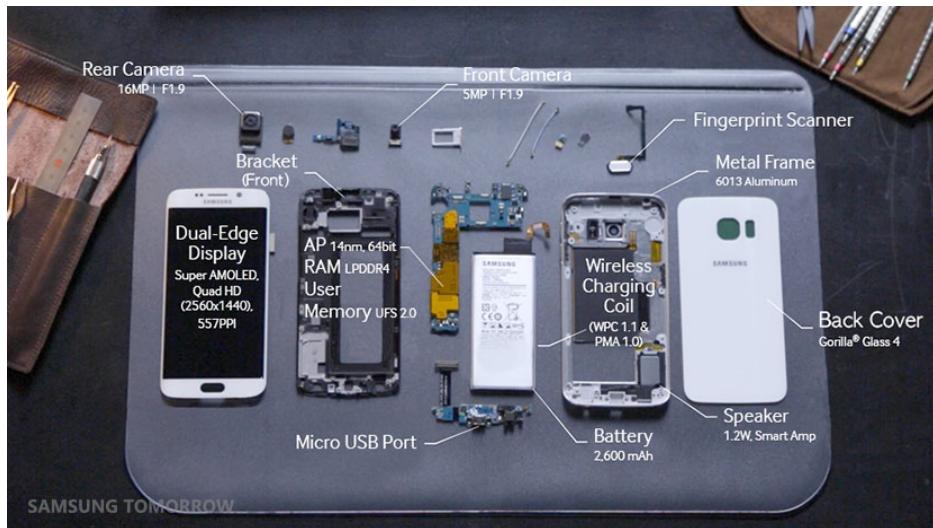
Welche Komponenten fehlen hier um einen funktionsfähigen Rechner zu bekommen?

- Prozessor
  - Diese und nächste Vorlesung
- RAM
- Festplatten
- Optische Laufwerke
- I/O-Geräte
- Netzwerke und Schnittstellen
- Spätere Vorlesungen



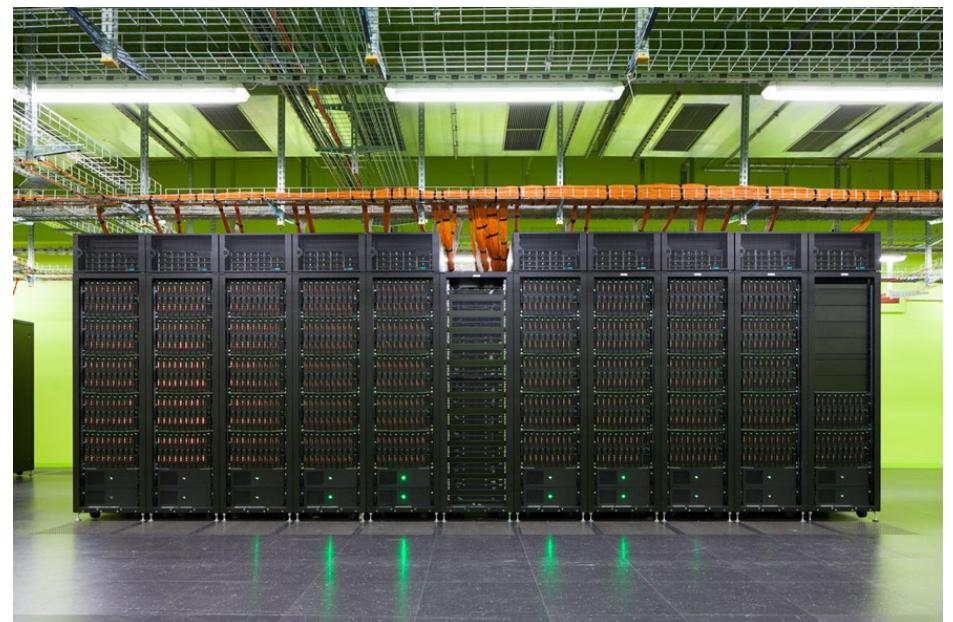
# Rechnersysteme sind ...

.. kleiner (z.B. Samsung S6)



© Samsung

... größer (z.B. HPC System @ ZIH)

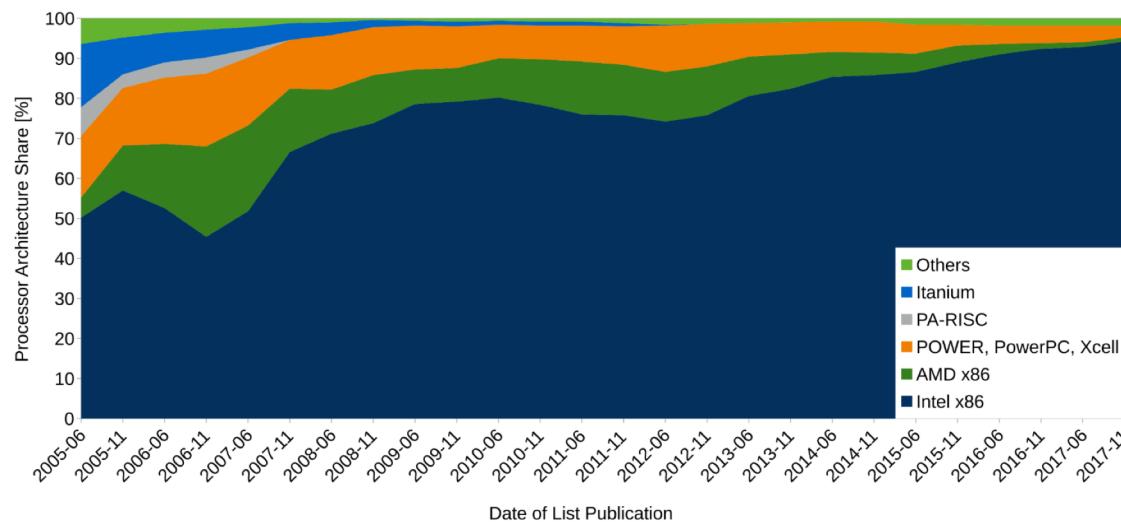


© Robert Gommlich

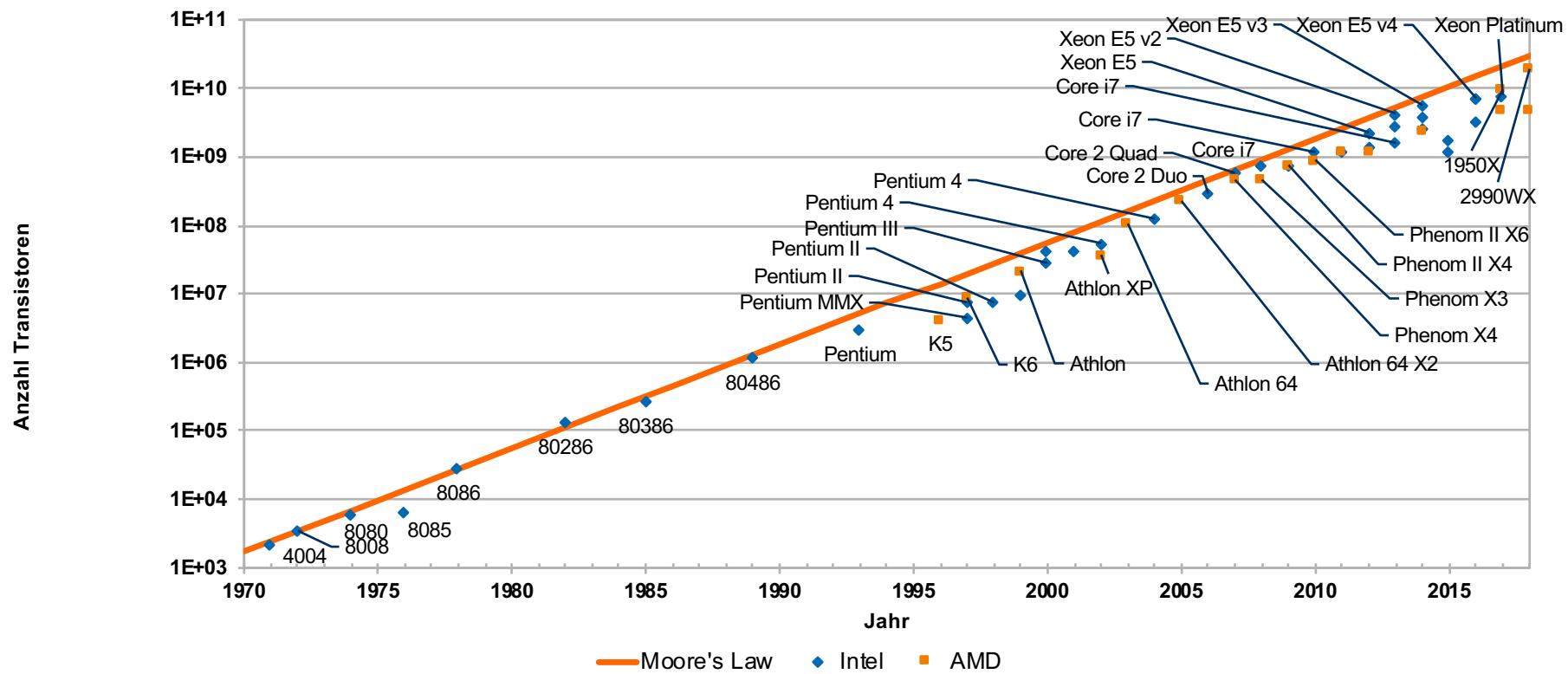
# Evolution der x86-Prozessoren

# Warum x86 Prozessoren?

- Nachvollziehbare Entwicklung über mehrere Jahrzehnte
- Implementieren viele Konzepte aus der Rechnerarchitektur
- Sind weit verbreitet (Laptop, Desktop, Server, HPC Systeme)
- HPC, Top500:



# Was treibt die Entwicklung: Moore's Law



- Verdopplung der Transistorzahl alle 18 - 24 Monate
- Herausforderung: Transistoren zur Erhöhung der Rechenleistung nutzen

**x86 ist eine CISC Architektur.  
Klassifikation von Befehlssätzen**

# RISC und CISC

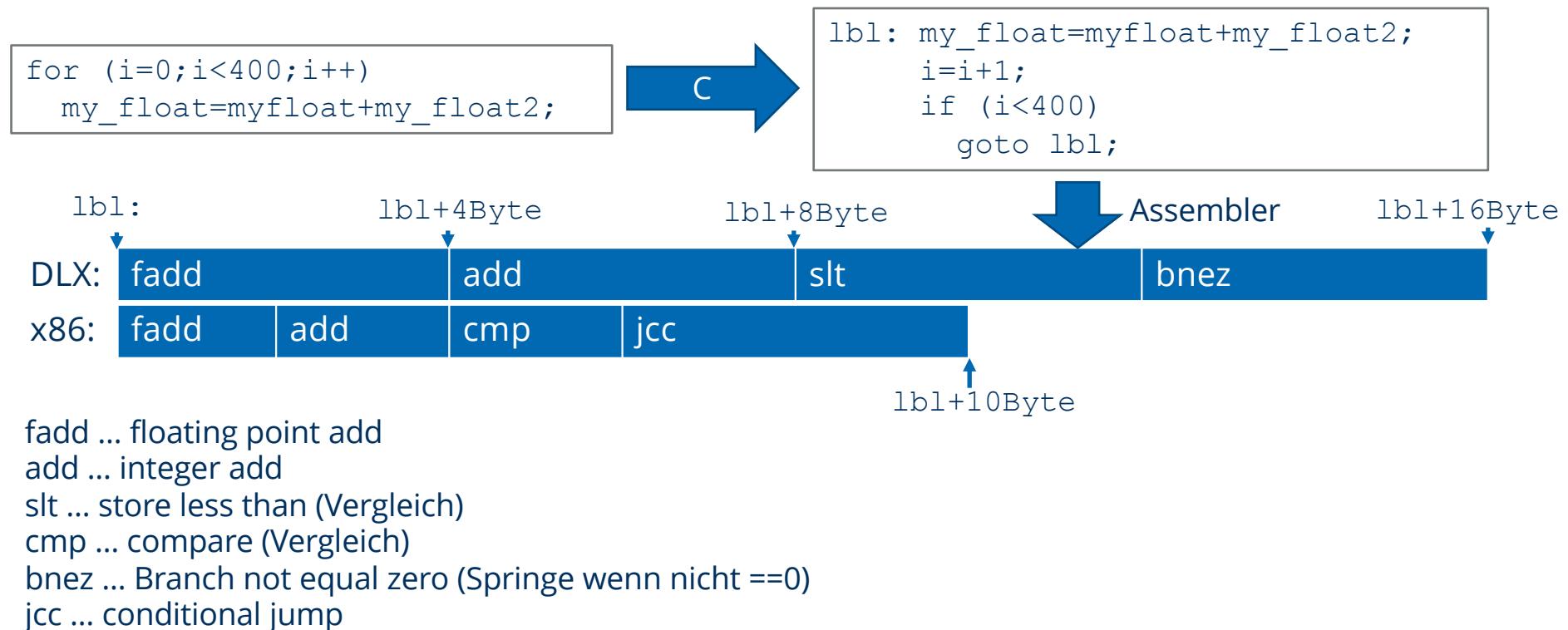
RISC = *Reduced Instruction Set Computing*

- Feste Befehlslänge
- Einfacher zu dekodieren
- Adresse des nächsten Befehls bekannt → gut für Pipelining
- Wenige Befehle/Befehlsformate
- Wenige, einfache Adressierungsarten
- Load/Store-Architektur
  - Jedes Datum muss explizit in Register geladen werden bevor damit gerechnet werden kann
  - Braucht mehr Register

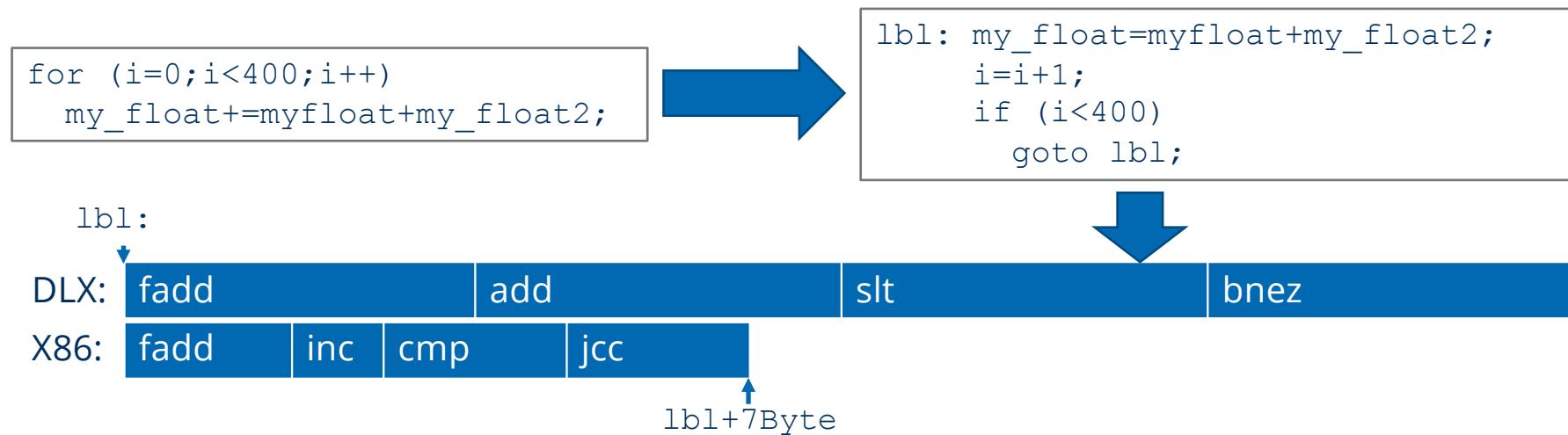
CISC = *Complex Instruction Set Computing*

- Variable Befehlslänge
- Kurze *durchschnittliche* Befehlslänge (seltene Befehle sind länger, vgl. Kompression)
- Einfach erweiterbar
- Viele Befehle/Befehlsformate
- Komplexe Adressierungsarten
- Arithmetische Befehle mit Speicheroperanden
- Operanden können direkt aus Speicher genutzt werden

## Beispiel feste vs. variable Befehlslänge (C → Assembler)



## Beispiel CISC unterstützt viele Befehle (kürzere Befehle möglich)



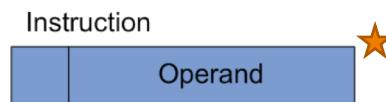
Neu:

*Increment statt Addition 2 Byte → 1 Byte*

*Jump short (1 Byte Befehlscode + 1 Byte Adresse) statt Jump near*

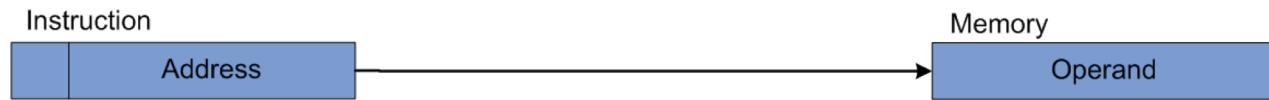
# Beispiele Adressierungsarten von CISC (x86) und RISC (MIPS)

- Immediate Operand
- z.B. add **\$400**, %r10

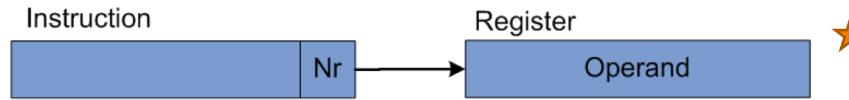


★ Verfügbar auf RISC (MIPS)

- Immediate Adresse
- z.B. add **(%rsp)**, %r10



- Register Zugriff
- z.B. add **%r2**, %r10

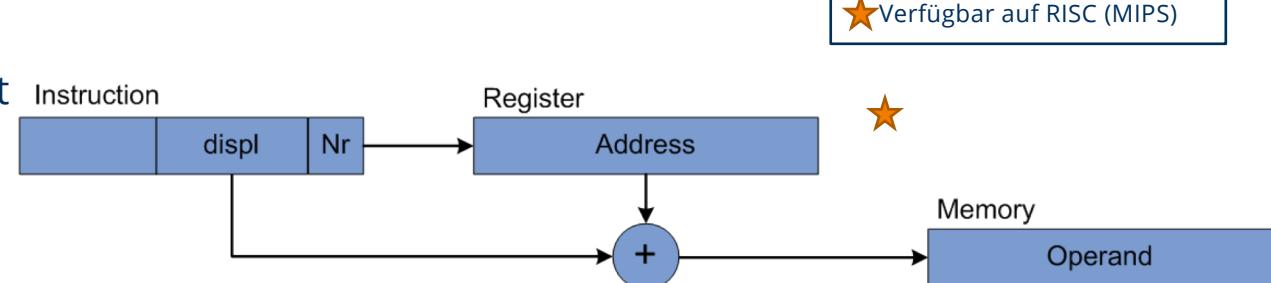


- Register indirekt
- z.B. add **(%r2)**, %r10

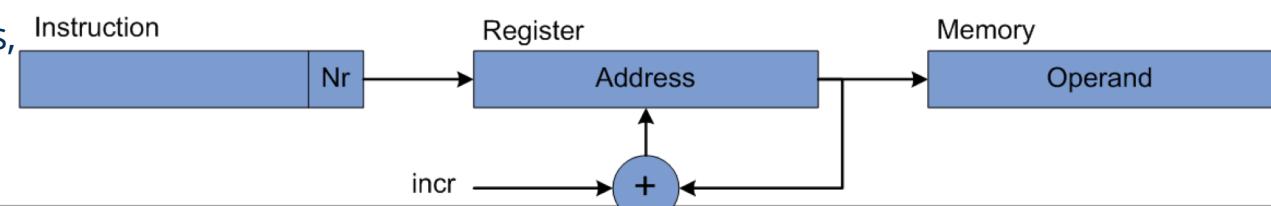
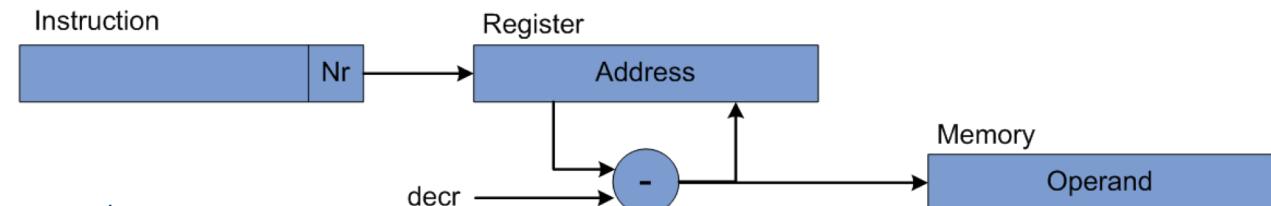


# Beispiele Adressierungsarten von CISC (x86) und RISC (MIPS)

- Register indirekt mit displacement
  - z.B. add **16(%rsp)**, %r10

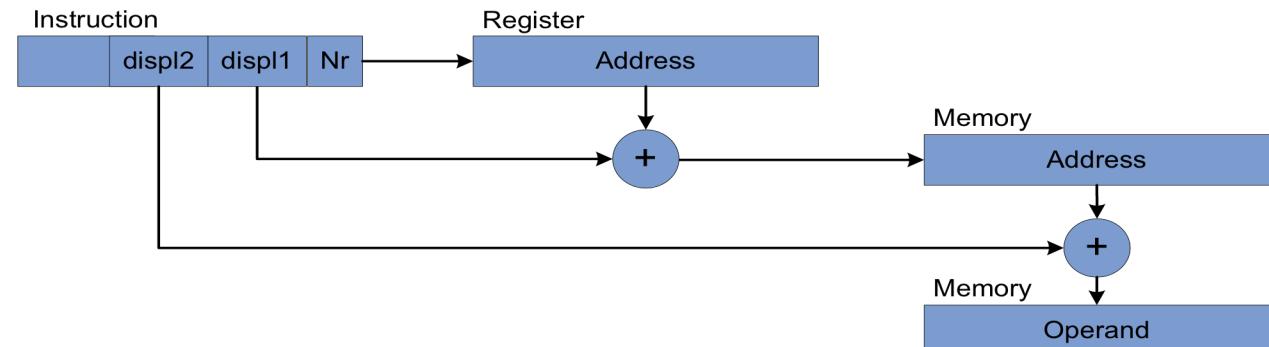


- Zugriffe auf Stack, z.B. push/pop
- Register indirekt mit preautodecrement
  - z.B. loop:  
Dekrementiere Counter, wenn nicht 0  
dann führe Sprung aus
- Register indirekt mit postautoincrement
  - Z.B. rep Prefix:  
So lange Counter > 0: führe Befehl aus,  
dekrementiere Counter



# Beispiele Adressierungsarten von CISC (x86)

- Implizite Adressierung
  - Feste Quell- oder Zielregister für Befehle z.B.:
    - Ein Operand in EAX (z.B. *cbw* (*Convert Byte to Word*) als 1 Byte Befehl)
    - Loop Counter in ECX (z.B. *loop* Befehl: Dekrementiert ECX und führt ggf. Sprung aus)
    - ESP/EBP/ESI/EDI für Adressberechnung
  - Spart Bits im Befehl
- Speicher indirekt



**CISC ist leicht erweiterbar →  
x86 verschiebt Grenzen des Machbaren**

# Ursprung und frühe Jahre

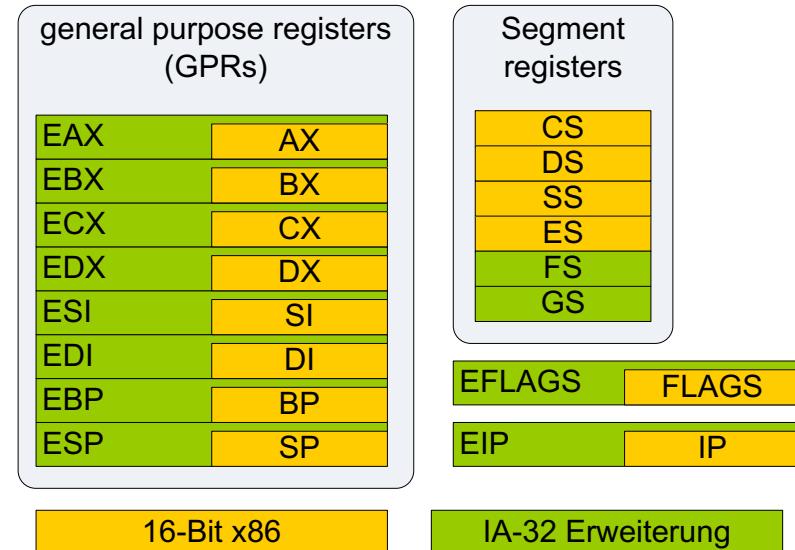
- 1968: Intel Gründung durch ehemalige Fairchild Mitarbeiter
- 1971: erster Mikroprozessor
- 1978: Intel 8086/8088 (29.000 Transistoren)
  - 16-Bit Verarbeitungsbreite
  - Intel 8088 **weit verbreitet in IBM PC** (1981)
    - Viel Software
    - Dadurch de-facto Standard
  - 20-Bit Adressraum (1 MB), reicht nicht für „neue“ Software
  - Keine Memory Management Unit (MMU) → kein OS Multitasking
- 1982: Intel 80286 (130.000 Transistoren)
  - 24-Bit Adressraum (16 MB), aber kaum Systeme mit „viel“ RAM
  - **MMU**, Floating Point Unit (FPU), als Co-prozessor
  - Aber 16 Bit Verarbeitungsbreite zu schmal

Software seiner Zeit verlangt IBM PC,  
Heute: bestimmte Prozessoren

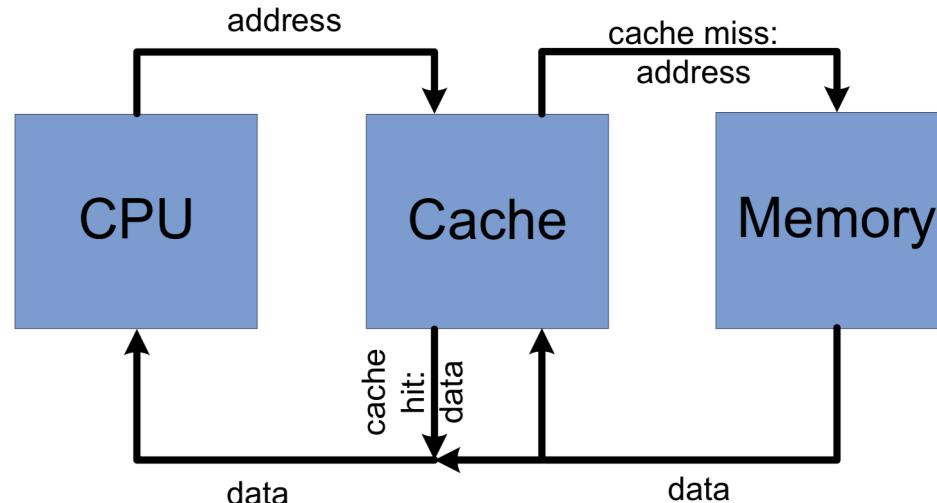


# 32-Bit x86 Prozessoren

- 1985: Intel 80386 (280.000 Transistoren)
- **32 Bit Verarbeitungsbreite**
  - Breitere Register werden über alte Register gemappt
- Adressumsetzung in Hardware erweitert
  - Anforderung für Windows 95
- FPU nur als Co-Prozessor →  
Fließkommaprogramme langsam
- Speicherzugriff langsam
- 1989: Intel 80486 (1.2 M Transistoren)
  - 8 KB **Cache** für Code und Daten
  - **Integrierte FPU**
  - Besseres **Pipelining**



# Cache



- Kleiner, schneller Zwischenspeicher
- Puffer für häufig benötigte Daten und Befehle

# Fokus Pipelining

- Aufteilung der Befehlsverarbeitung in 5 Phasen
  - Instruction Fetch (IF): Befehl laden
  - Decode1 (D1)
  - Decode2 (D2)
  - Execute (EX): Ausführung
  - Write Back (WB): Ergebnis in Ziel-Register schreiben
- Mehrere Befehle gleichzeitig in Bearbeitung

Takt	1	2	3	4	5	6	7	8	9	10
Inst1	IF	D1	D2	EX	WB					
Inst2		IF	D1	D2	EX	WB				
Inst3			IF	D1	D2	EX	WB			
Inst4				IF	D1	D2	EX	WB		
Inst5					IF	D1	D2	EX	WB	
Inst6						IF	D1	D2	EX	WB

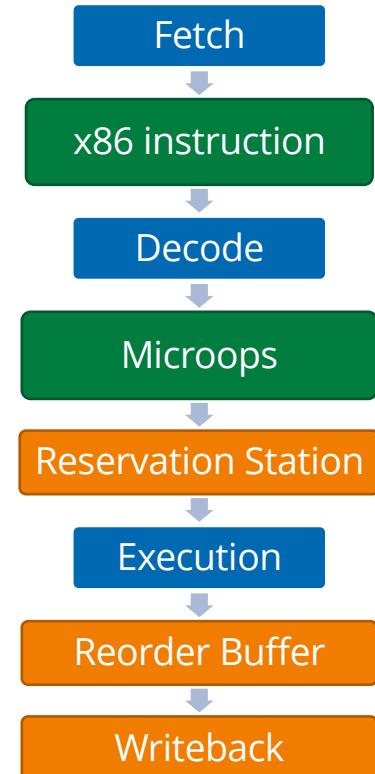
# Zwischenbetrachtung

29.000 Transistoren → 1.200.000 Transistoren

- Zusätzliche Komponenten:
  - MMU
  - Höhere Verarbeitungsbreite
  - Cache
  - FPU
- Parallelverarbeitung via Pipelining
- Höhere Frequenz
- Problem 1: Nur eine Operation pro Takt wird fertig (im besten Fall)
- Problem 2: Zugriff zum Cache immer noch langsam durch von-Neumann-Flaschenhals
- Problem 3: CISC Dekodierung ist komplex, Befehle dauern unterschiedlich lang

# Erweiterungen durch Pentium

- 1993 Pentium (3.1 M Transistoren)
  - Getrennte L1-Caches; 8 KB für Instruktionen; 8 KB Daten
    - **Harvard-Architektur** im Cache, kein von-Neumann-Flaschenhals
  - **Superskalarität**: 2 Befehle werden pro Takt ausgeführt
- 1995: Intel Pentium Pro (5.5 M Transistoren)
  - Umwandlung der CISC Instruktionen in **RISC Mikrooperationen** ( $\mu$ ops)
    - Einfache Instruktionen eine  $\mu$ ops
    - Komplexe Instruktionen mehrere  $\mu$ ops
  - Bessere Nutzung der Pipelines
  - **Out-of-Order Execution**
    - Abarbeitung der  $\mu$ ops abweichend von der Programmreihenfolge
    - Wenn eine Instruktion auf Speicher wartet, kann andere ausgeführt werden



# Fokus Superskalarität

- Mehrere Befehle werden pro Takt ausgeführt
- Braucht mehrere Funktionseinheiten
- Befehle müssen unabhängig voneinander sein

Takt	1	2	3	4	5	6	7	8	9
Inst1	IF	D1	D2	EX	WB				
Inst2	IF	D1	D2	EX	WB				
Inst3		IF	D1	D2	EX	WB			
Inst4		IF	D1	D2	EX	WB			
Inst5			IF	D1	D2	EX	WB		
Inst6			IF	D1	D2	EX	WB		
Inst7				IF	D1	D2	EX	WB	
Inst8				IF	D1	D2	EX	WB	
Inst9					IF	D1	D2	EX	WB
Inst10					IF	D1	D2	EX	WB

# Pentium MMX – Es wird explizit parallel

- Bisher SISD sichtbar, intern parallel (Pipelining, Superskalarität)
- Manche Probleme können sehr gut parallel abgearbeitet werden:
- Beispiel Green Screen/Chroma Keying:



Intel MMX™ Technology Overview, Intel, 1996, Order Number: 243081-002

# Pentium MMX – Es wird explizit parallel

- Bisher SISD sichtbar, intern parallel (Pipelining, Superskalarität)
- Manche Probleme können sehr gut parallel abgearbeitet werden:
- Beispiel Green Screen/Chroma Keying:

**for** pixel **in** pixels:

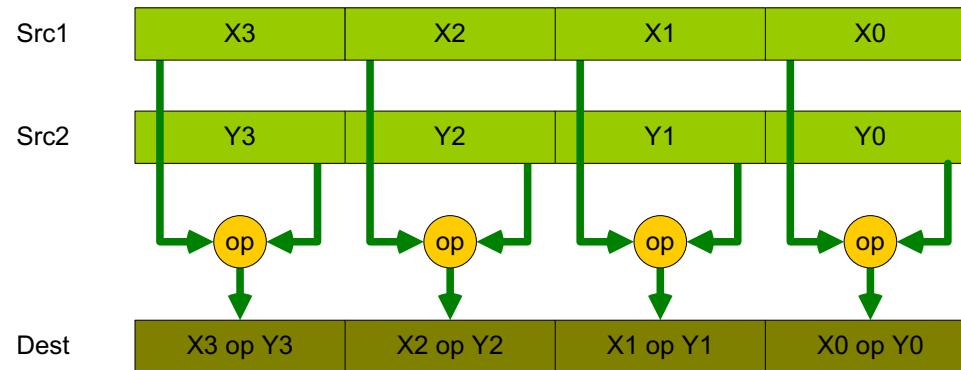
**if** pixel.color == GREEN:

        pixel.color = background\_pixels[pixel.x][pixel.y].color # can be done independently for all pixel

- 1996 SIMD Erweiterung MMX
  - SIMD Klassifikation nach FLYNN
  - Bei MMX: Integer Variablen

# SIMD Erweiterungen am Beispiel MMX

- SIMD Befehle:



- Bei MMX: 64 Bit breite Vektoren, Operanden z.B. 8 Bit breit → 8 Operationen pro Instruktion

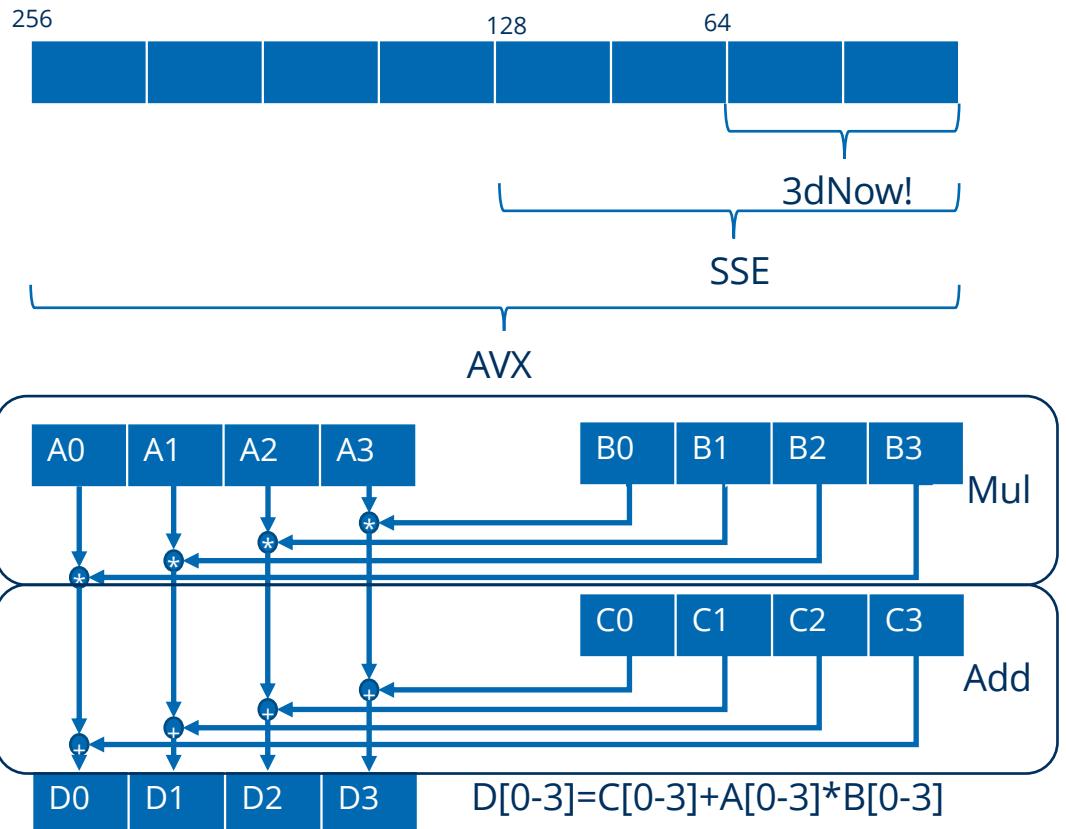
# Fokus SIMD

- Kann verschieden umgesetzt werden
  - Feldrechner (typisch in SIMD Erweiterungen)
  - Vektorrechner (spätere Vorlesung)
- Feldrechnerprinzip:
  - Mehrere Ausführungseinheiten
  - Werden von derselben Instruktion gesteuert (Single Instruction Stream)
  - Bekommen unterschiedliche Daten (Multiple Data Stream)
  - Führen taktsynchron Operation aus
- Beispiel:
  - `movps (%eax), xmm0; // lädt 4 konsekutive single precision Werte in Register xmm0`
  - `mulps xmm1, xmm0; // skaliert die 4 Werte mit den Werten in xmm1`
  - `movps xmm0, (%eax) // schreibt das Ergebnis in den Speicher`

# Fokus SIMD

Kann Performance deutlich steigern, abhängig von

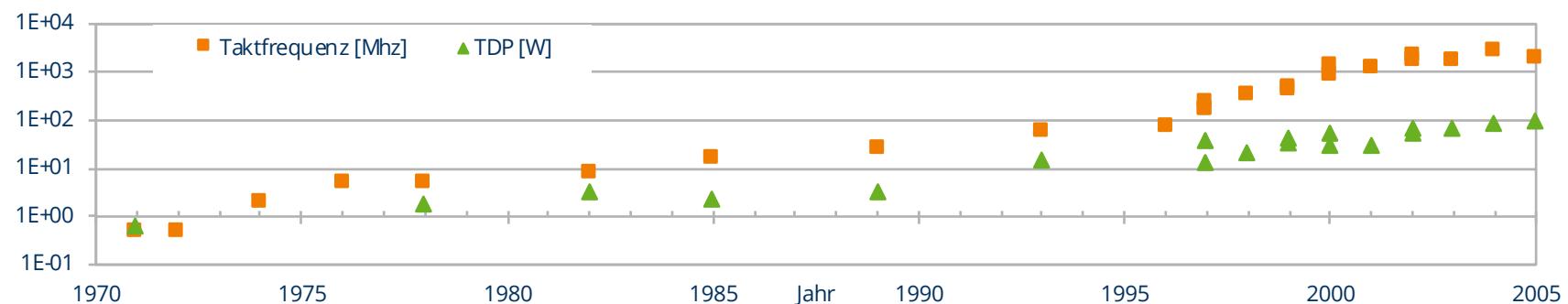
- Breite der verarbeiteten Daten
  - Bestimmt max. Speedup
- Anzahl der Ausführungseinheiten
  - Es muss nicht für jedes Element eine Ausführungseinheit da sein
  - Z.B. Aufteilen einer 256 Bit Operation in 2x128 Bit
- Komplexität der Instruktion
  - Mul/Add: Eine Operation
  - Fused Multiply Accumulate: 2 Operationen  
(Kann auch aufgeteilt werden)



# Inkrementelle Verbesserungen bis 2005 – Pentium 2 bis Pentium 4

Kleinere Strukturen, mehr Transistoren (>100 Millionen)

- Mehr und größere Caches
- Mehr Superskalarität
- Mehr SIMD Erweiterungen (Floating Point, 128 bit – SSE, SSE2, SSE3, ...)
- Höhere Frequenz (bis 3,8 GHz)

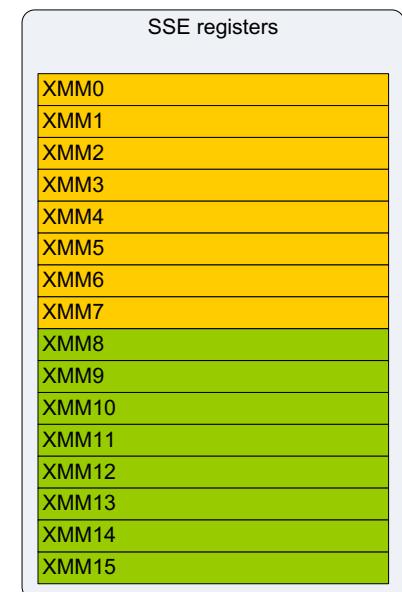
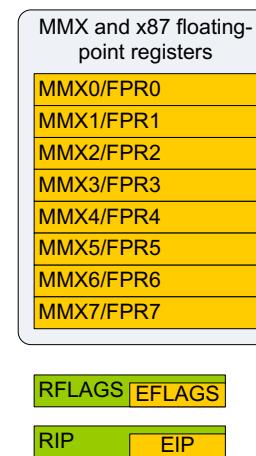
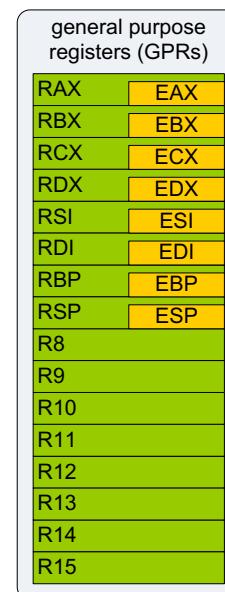


# Herausforderungen zur Zeit des Pentium 4

- Problem 1: 32-Bit reichen nicht aus
  - Intel: neue Architektur (Vorlesung nächste Woche)
  - AMD: Erweiterung der x86 Architektur x86\_64
  - Neue Register, SSE Teil der ISA, ...
- Problem 2: TDP ist erreicht
  - Niedrigere Frequenz
    - Software kann mit neuem Prozessor langsamer werden!
  - Mehr Kerne
    - Braucht Unterstützung durch Betriebssystem und Software
  - Neue Stromsparmöglichkeiten (Schlafzustände, Frequenzwechsel)
    - Braucht Unterstützung durch Betriebssystem
- The Free Lunch Is Over – A Fundamental Turn Toward Concurrency in Software, Herb Sutter, 2005,  
Dr. Dobb's Journal, 30(3)

# Erweiterung auf 64 Bit

- Zuerst von AMD, später Intel
- Mehr Speicher adressierbar
- Native 64 Bit Integerberechnung
  
- Rekapitulation CISC
  - Unterschiedliche Befehlslänge
  - Einfach erweiterbar
- Erweiterung von Befehlen um 1 Byte
  - Dadurch mehr Information encodierbar
  - Dadurch mehr Register adressierbar
- Doppelt so viele GPR und SIMD Register (SSE)



IA32 (mit MMX und SSE)

64-Bit Erweiterung

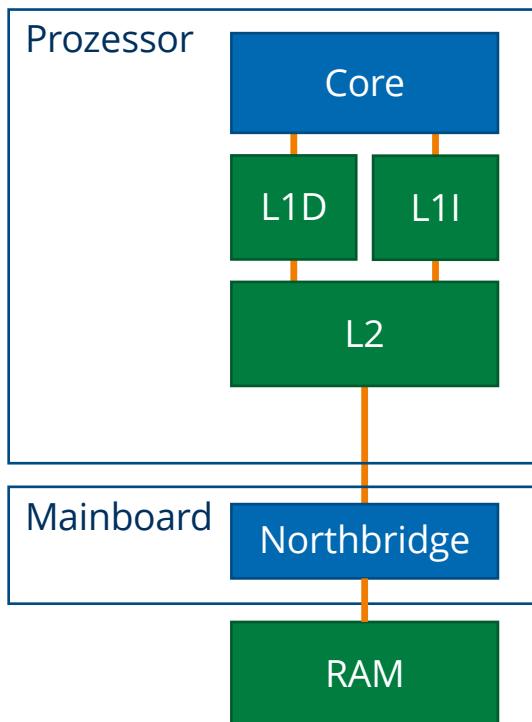
# Herausforderungen zur Zeit des Pentium 4

- Problem 1: 32-Bit reichen nicht aus
  - Intel: neue Architektur (Vorlesung nächste Woche)
  - AMD: Erweiterung der x86 Architektur x86\_64
  - Neue Register, SSE Teil der ISA, ...
- Problem 2: Thermal Design Power (TDP) ist erreicht
  - Niedrigere Frequenz
    - Software kann mit neuem Prozessor langsamer werden!
  - Mehr Kerne
    - Braucht Unterstützung durch Betriebssystem und Software
  - Neue Stromsparmöglichkeiten (Schlafzustände, Frequenzwechsel)
    - Braucht Unterstützung durch Betriebssystem
- **The Free Lunch Is Over – A Fundamental Turn Toward Concurrency in Software**, Herb Sutter, 2005,  
Dr. Dobb's Journal, 30(3)

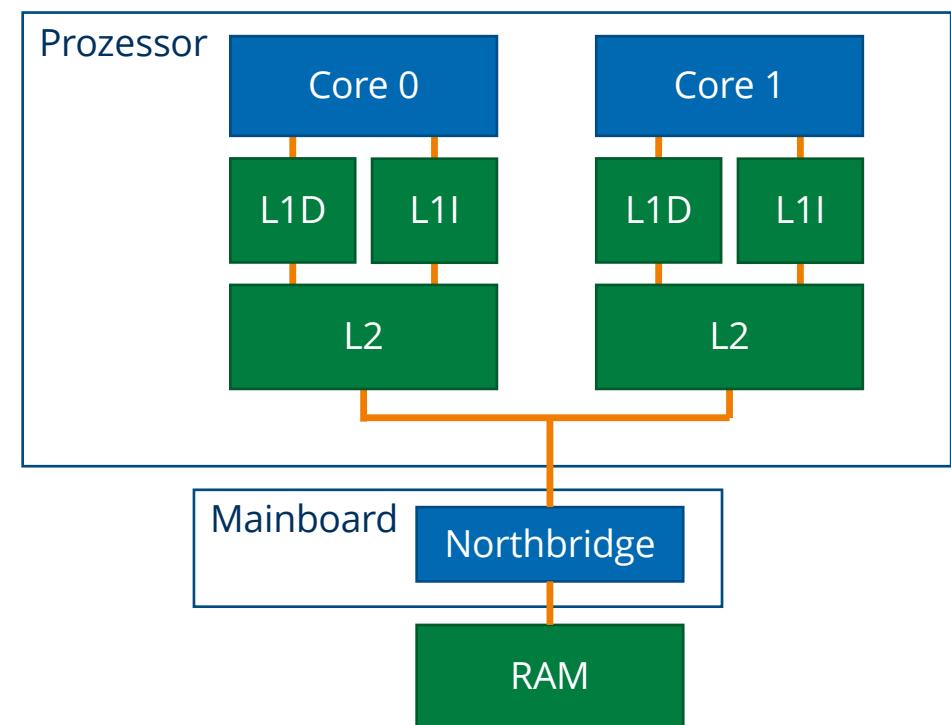
# Parallele x86 Prozessoren

# Multiprocessor und Multicore

Vorher: SISD/SIMD



Nachher: MIMD



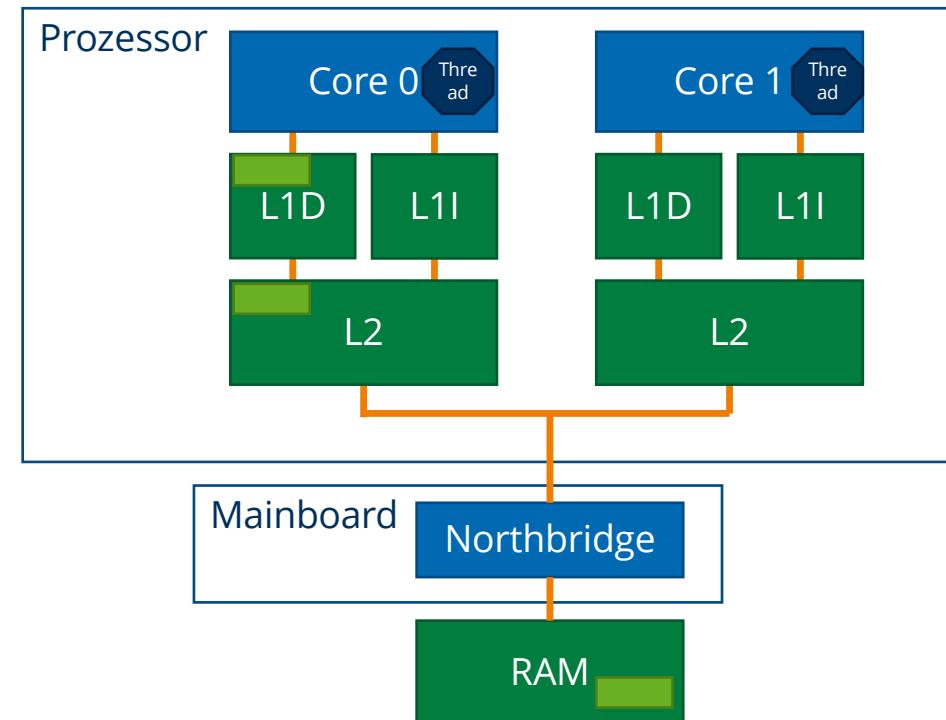
# Interaktion mit anderen Disziplinen

- Betriebssysteme
  - Welcher Kern behandelt Interrupts?
  - Wie werden Prozesse und Threads gescheduled?
  - Kann man Threads und Prozesse an Kerne pinnen?
- Compilerbau
  - Muss Latenzen zwischen Kernen kennen um zu optimieren
  - Muss parallele Programmiermodelle unterstützen (z.B. *OpenMP*, mehr dazu im Wintersemester)
- Softwaretechnik
  - Parallel Programmierung ist jetzt Mainstream
  - Spracherweiterungen für parallele Programmiermodelle

# Herausforderungen für die Rechnerarchitektur

Cache Kohärenz:

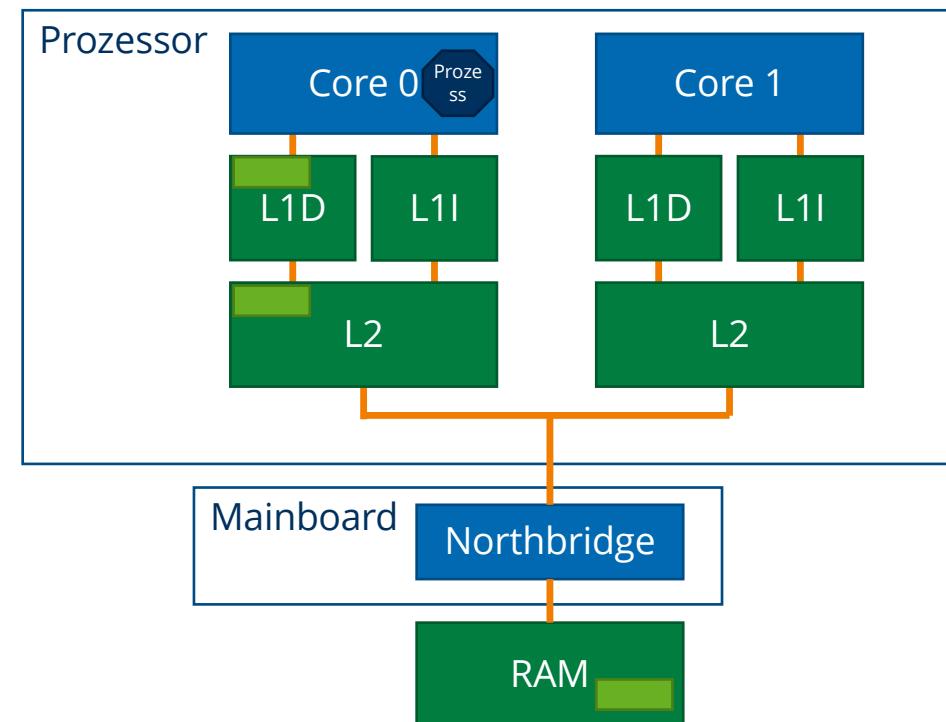
- Threadparallele Software: Kommunikation über gemeinsamen Speicher
- 2 Threads laufen auf 2 Kernen
- Thread 0 lädt Daten
  - Kopie in Caches
- Thread 0 ändert Daten
  - Update in Caches
- Thread 1 liest die gleichen Daten
  - Kommt in späterer Vorlesung



# Herausforderungen für die Rechnerarchitektur

Cache Kohärenz auch bei sequentiellen Anwendungen wichtig

- Prozess läuft auf Core 0
- Prozess lädt Daten
  - Kopie in Caches
- Prozess ändert Daten
  - Update in Caches
- Prozess wird auf Core 1 verschoben

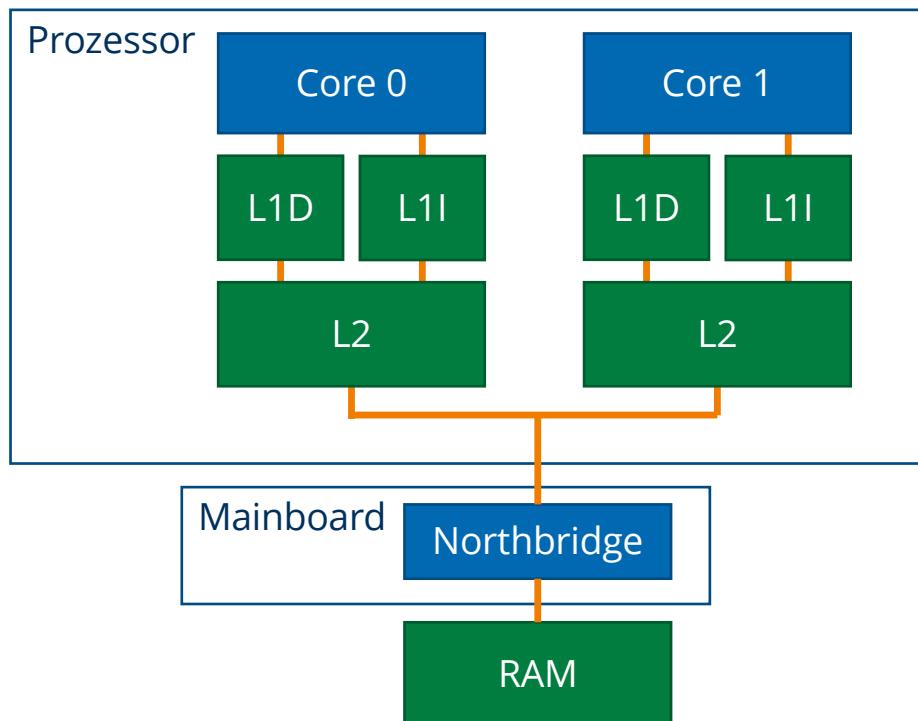


# Sequentielle Software auf parallelen Prozessoren

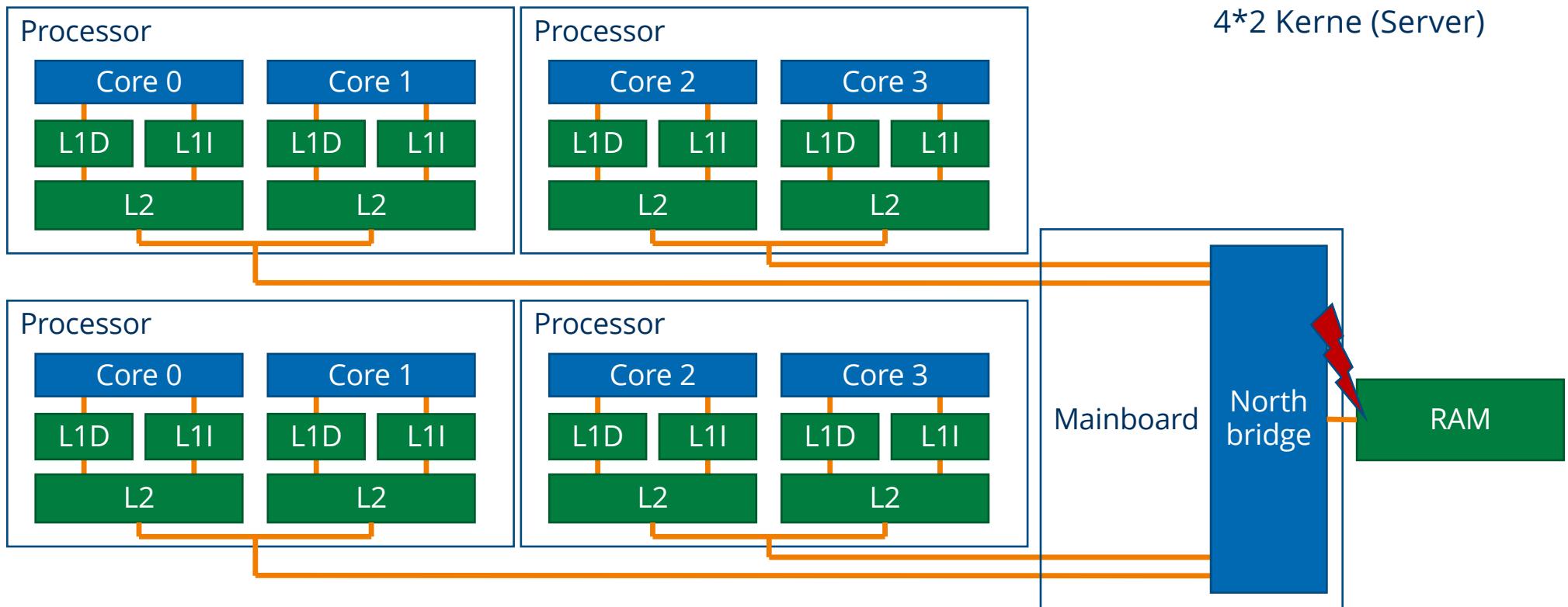
- TDP gilt für gesamten Prozessor
- Sequentielle Programme benötigen nur einen Kern, die inaktiven Kerne brauchen im idle wenig/keine Energie
- Lösung 1: verschiedene Prozessoren für verschiedene Nutzer:
  - Mehr Kerne mit niedriger Frequenz sind effizienter für parallele Anwendungen
  - Wenige Kerne mit hoher Frequenz sind effizienter für sequentielle Anwendungen
- Verschiedene Prozessoren, z.B. 65W Core2: 2x 3.33 oder 4x 2.83 GHz
- Lösung 2: Turbo Boost
  - „Übertakten“ des *aktiven* Prozessorkerns
  - Höhere Frequenz, ggf. Spannung
  - Höherer Verbrauch des aktiven Kerns, aber Prozessor immer noch unterhalb der TDP

# Parallele Prozessoren und gemeinsamer Speicher

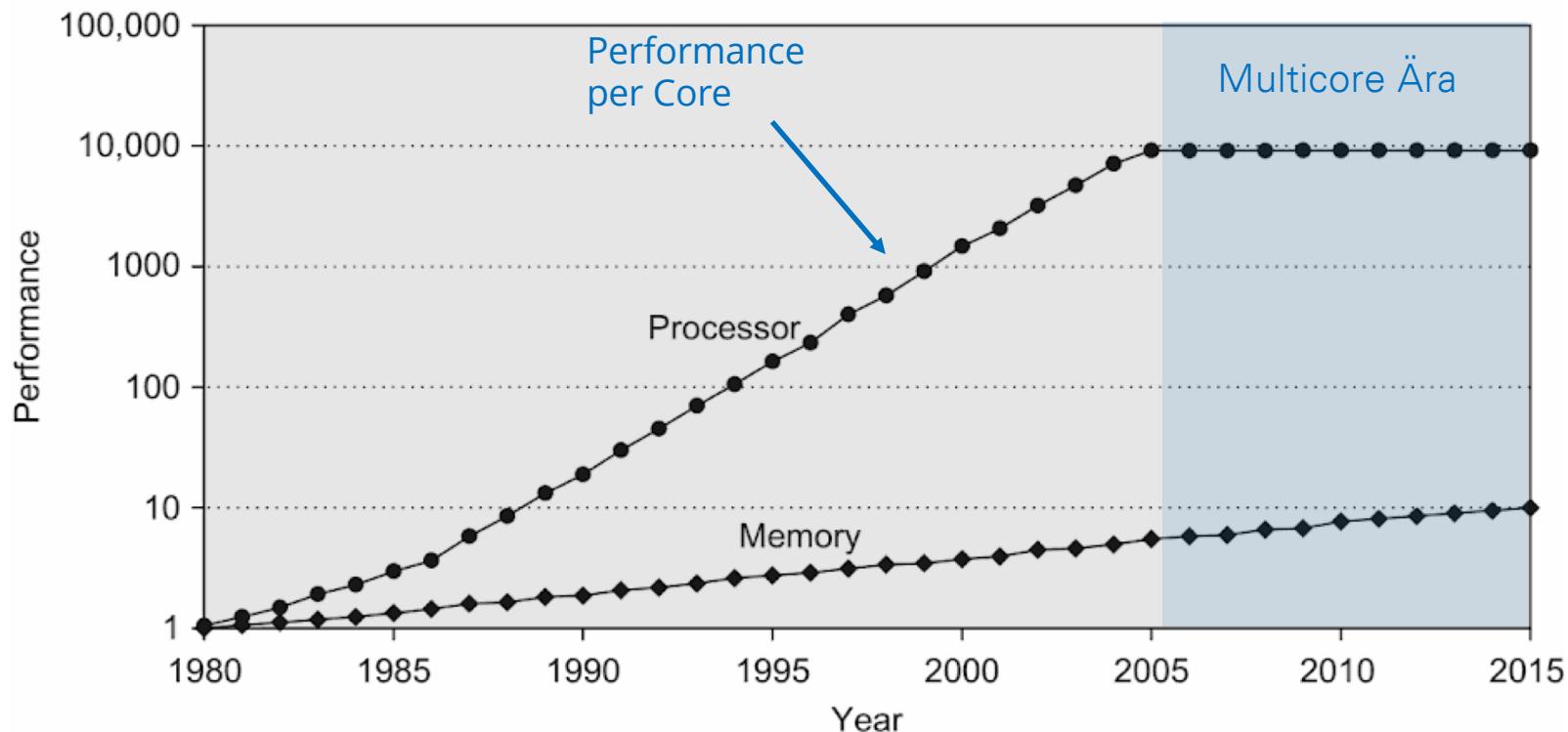
2 Kerne:



# Parallele Prozessoren und gemeinsamer Speicher

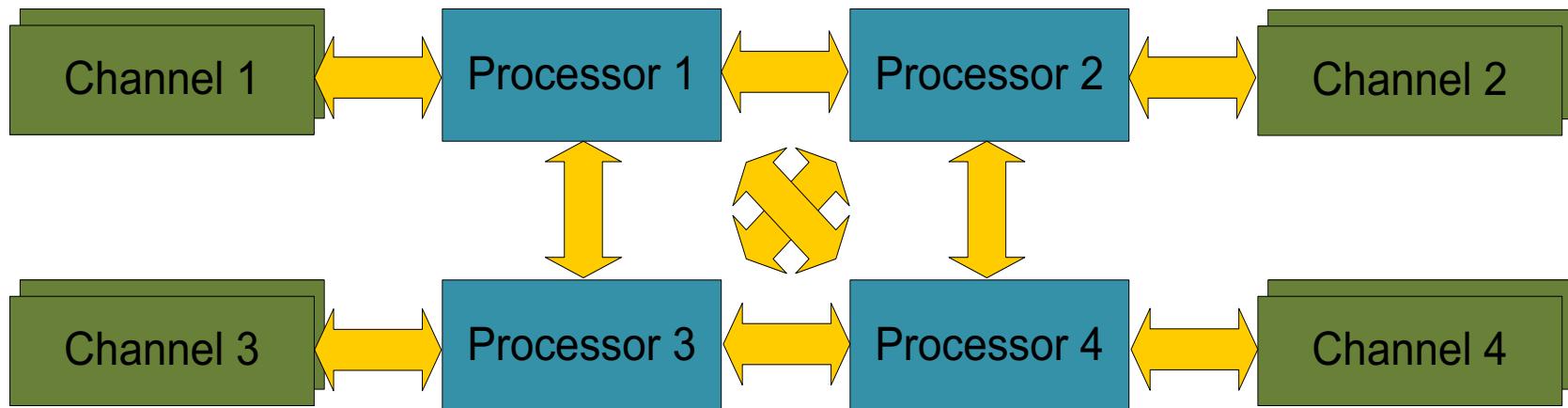


# Processor-Memory Gap



Hennessy, Patterson, *Computer Architecture: A Quantitative Approach 6<sup>th</sup> ed.*, Morgan Kaufmann, ISBN 9780128119051, Figure 2.2

## Gemeinsamer verteilter Speicher – Aus UMA wird NUMA



- Integrierte Speichercontroller in Prozessoren
- Unterschiedliche Latenz/Bandbreite zu lokalem und entferntem Speicher (NUMA: non-uniform memory access) → Sollte von Betriebssystem und Software beachtet werden: Wo sind Prozesse und ihr Speicher?
- Erst Serverprozessoren, seit Ryzen auch Desktopprozessoren

## Weitere inkrementelle Verbesserungen bis 2019

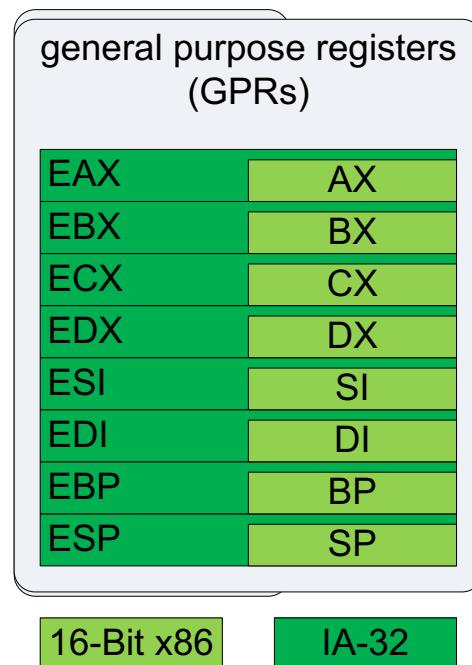
- Erweiterung SIMD auf 256, 512 Bit breite Vektoren
  - Neue Befehle: FMA (Fused Multiply Add), SHA, AES, ...
  - Mehr Kerne
  - EVEX Erweiterung:
    - Verdoppelung der SIMD Registeranzahl (jetzt: 32)
    - Maskenregister
- Transistoranzahl bei Intel nicht bekannt, bei AMD (Threadripper) 9,6 Mrd Transistoren

# Zusammenfassung

# Stetige Transistorerhöhung

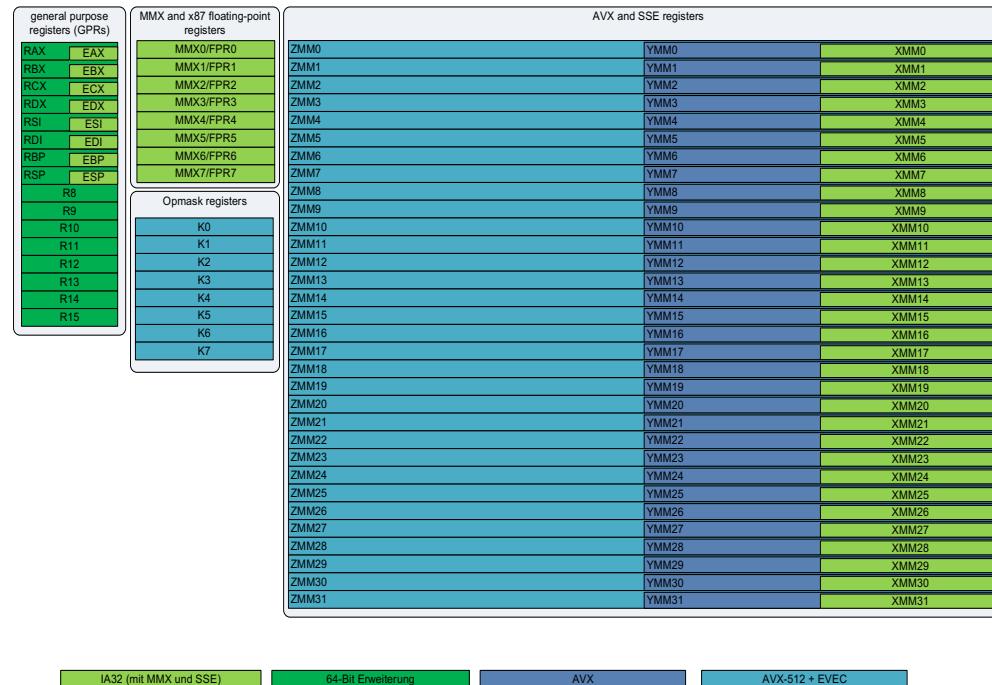
- Von 29.000 auf 9,6 Milliarden Transistoren
- Mehr Cache
- Mehr Funktionseinheiten
  - MMU
  - FPU
  - SIMD
- Mehr spekulative Ausführung
  - Branch Prediction
  - Out-of-Order Execution

# x86 ist eine CISC Architektur



- Unterschiedliche Befehlslänge und Komplexität
- Ermöglicht einfache Erweiterung der Befehle, Adressraums, Registerraums ...

# x86 ist eine CISC Architektur



- Unterschiedliche Befehlslänge und Komplexität
- Ermöglicht einfache Erweiterung der Befehle, Adressraums, Registerraums ...
- Aber: Intern RISC, da einfacher umzusetzen

## Aus SISD wurde MIMD

- Einordnung nach FLYNN
- Software muss SIMD und mehrere Kerne nutzen um maximale Performance zu erreichen
- Betriebssystem muss neue Schnittstellen unterstützen
- Beispiel: 2 \* AVX512 (SIMD) ,FMA, 2 GHz, 8-Kerne, single precision
- $\frac{512 \text{ bit (vector length)}}{32 \text{ bit (operand length)}} = 16$
- $2 \frac{\text{Instructions}}{\text{cycle}} * 16 * 2 \frac{\text{Operations}}{\text{Instruction}} = 64 \frac{\text{Operation}}{\text{cycle}}$
- Jeder Kern kann 64 float Operationen (FLOP) pro Takt berechnen (128 GFLOPS pro Sekunde)
- Bei 8 Kernen sind das 1,024 TFLOPS
- Sollte man nutzen, denn sequentiell (mit Pipelining): 2 GFLOPS → **Faktor 512**

# Auch sequentielle Codes werden parallel ausgeführt

- Pipelining
  - Befehle werden in mehrere Phasen aufgeteilt
  - In jedem Takt wird eine Phase bearbeitet
  - Im nächsten Takt wird die selbe Phase des nächsten Befehls ausgeführt
  - z.B. Decode
- Superskalarität
  - Mehrere Befehle werden gleichzeitig gestartet und parallel abgearbeitet

# Out-of-order und spekulative Ausführung verhindert Wartezeiten

- Prozessoren müssten warten:
  - Auf Speicher (> 100 Takte)
  - Auf Ergebnisse von vorhergehenden Operationen
- Prozessoren erreichen maximale Performance nur selten
  
- Lösung
  - Andere Instruktionen vorziehen
  - Gegebenenfalls Instruktionen wieder zurücknehmen (spekulative Ausführung)