LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Vorlesung Rechnerarchitektur

Sommersemester 2016

Carsten Hahn

14. April 2016

Agenda

- Informationsdarstellung und Adressierung
- Komponenten eines PCs
- Prozessorgrundstruktur und Befehlszyklus
- Compiler, Interpreter, Assembler

Darstellung von Informationen im Rechner

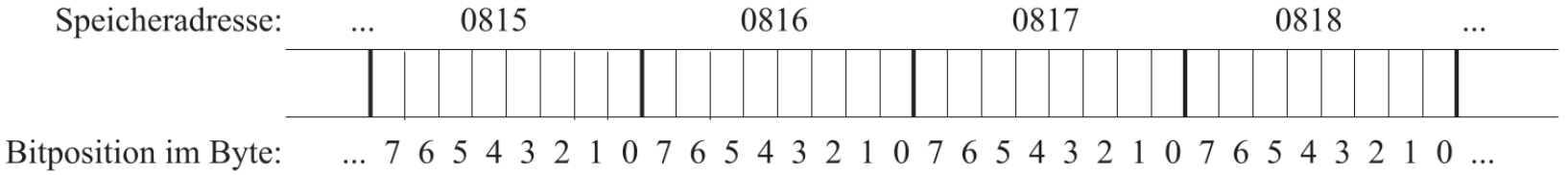
- Informationen sind sogenannte Wahrheitswerte
- Menge der Wahrheitswerte besteht genau aus zwei Elementen {wahr, falsch} bzw. {1, 0}
- Solche Informationen können vom Rechner verarbeitet/gespeichert werden z.B. durch Anliegen eines hohen oder niedrigen Spannungspegels

- Ein Wahrheitswert wird als Bit (Binary Digit) bezeichnet
- 8 Bit werden zu einem Byte zusammengefasst
  - ermöglicht die Bildung von $2^8 = 256$ verschiedenen Bitmustern
  - von 0000 0000 bis 1111 1111
  - zur kürzeren, übersichtlicheren Darstellung existiert hexadezimale Schreibweise

0000 ⇒ 0	0100 ⇒ 4	1000 ⇒ 8	1100 ⇒ C
0001 ⇒ 1	0101 ⇒ 5	1001 ⇒ 9	1101 ⇒ D
0010 ⇒ 2	0110 ⇒ 6	1010 ⇒ A	1110 ⇒ E
0011 ⇒ 3	0111 ⇒ 7	1011 ⇒ B	1111 ⇒ F

Speicheradressierung

- Ein Byte im Normalfall die kleinste adressierbare Speichereinheit
- Bei n-Adressbits können 2^n Byte adressiert werden
  - mit Adressen von 0 bis $2^n - 1$



- Größe eines Speichers wird üblicherweise wie folgt angegeben

Kilobyte	1 KB	1024 Byte	2^{10} Byte
Megabyte	1 MB	1024 KB	2^{20} Byte
Gigabyte	1 GB	1024 MB	2^{30} Byte
Terabyte	1 TB	1024 GB	2^{40} Byte
Petabyte	1 PB	1024 TB	2^{50} Byte

Einschub: Byte-Reihenfolge (Byte Order)

Mehr als ein Byte für die Speicherung einer Zahl:

- Bytes können in
  - aufsteigender
  - absteigender

  Reihenfolge aneinander gehängt werden.

Big-Endian (wörtlich „Großes Ende“):

- Byte mit den höchstwertigen Bits (signifikantesten Stellen) an der kleinsten Speicheradresse.

Little-Endian (wörtlich „Kleines Ende“):

- Byte mit den niederwertigsten Bits (wenigsten signifikanten Stellen) an der kleinsten Speicheradresse

Achtung:

- Der SPIM-Simulator benutzt die Byte-order des Rechners, auf dem er läuft.

Beispiel

Speicherung der Dezimalzahl 1296650323 als 32-Bit-Werte:

- Binär: 01001101 01001001 01010000 01010011
- Hexadezimal: 0x4D 0x49 0x50 0x53

Interpretation des 32-Bit-Wertes als Zeichenkette:

- ASCII (1 Byte/Zeichen): M I P S

Adresse		**...**	**0xA8**	**0xA9**	**0xAA**	**0xAB**	**...**
Big Endian	Binär	...	01001101	01001001	01010000	01010011	...
	Hex	...	0x4D	0x49	0x50	0x53	...
	ASCII	...	M	I	P	S	...
Little Endian	Hex	...	0x53	0x50	0x49	0x4D	...
	Binär	...	01010011	01010000	01001001	01001101	...
	ASCII	...	S	P	I	M	...

Unterschiede bei Konversionen

Beispiel:

- Konversion einer Zwei-Byte- in eine Vier-Byte-Zahl

Little-Endian-Maschine:

- Anfügen von zwei Null Bytes am Ende
- Speicheradresse bleibt gleich

Big-Endian-Maschine:

- Wert muss im Speicher um zwei Byte verschoben werden.

Umgekehrte Umwandlung:

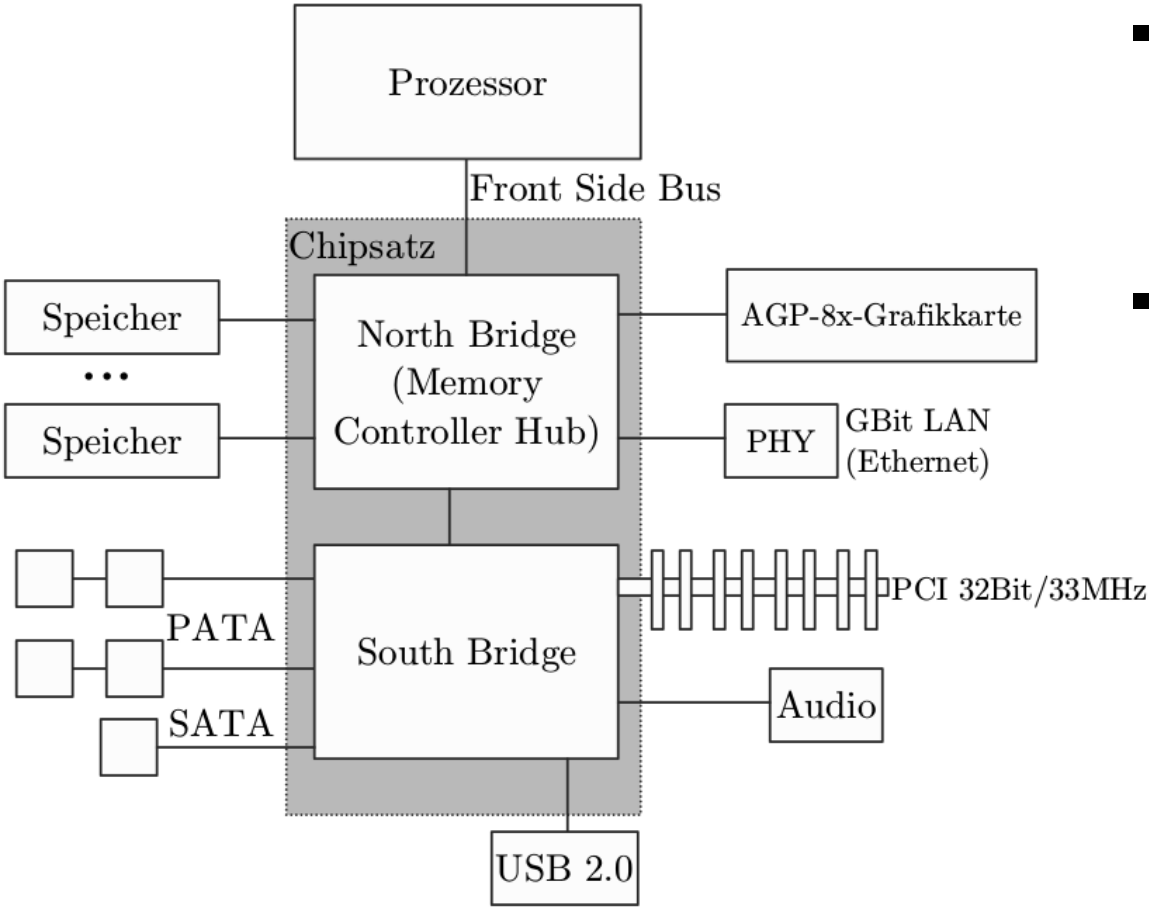
- Einfacher auf Little-Endian-Maschine
- höherwertige Bytes werden verworfen
- Speicheradresse bleibt gleich

Klassifikation von Computern

Eigenschaft	Preis in Dollar	Beispielanwendung
Wegwerfcomputer	0,5	Glückwunschkarten
Mikrocontroller	5	Uhren, Autos, Geräte
Spielkonsolen	50	Heimvideospiele
Personalcomputer	500	Desktop- oder Notebook-Computer
Server	5.000	Netzwerkserver
Verbund von Workstations	50.000 bis 500.000	Abteilungsrechner (Minisupercomputer)
Großrechner (Mainframe)	5 Millionen	Batch-Verarbeitung in einer Bank

Tanenbaum, Andrew S. "Computerarchitektur - Strukturen - Konzepte - Grundlagen" Pearson Studium, (5. Aufl.) (2006)

Mainboard-Layout bzw. Komponenten eines PC



Böttcher, Axel. Rechneraufbau und Rechnerarchitektur. Springer-Verlag, 2007.

- Prozessor: über schnellen Bus mit Komponenten auf der Hauptplatine verbunden
- Chipsatz: verbindet Komponenten, ist in zwei Teile geteilt
  - North Bridge: Koppelt Hauptspeicher, Grafikkarte und physische Schicht (Netzwerk)
  - South Bridge: bedient langsame Busse wie Eingabe/Ausgabe oder Festplatten

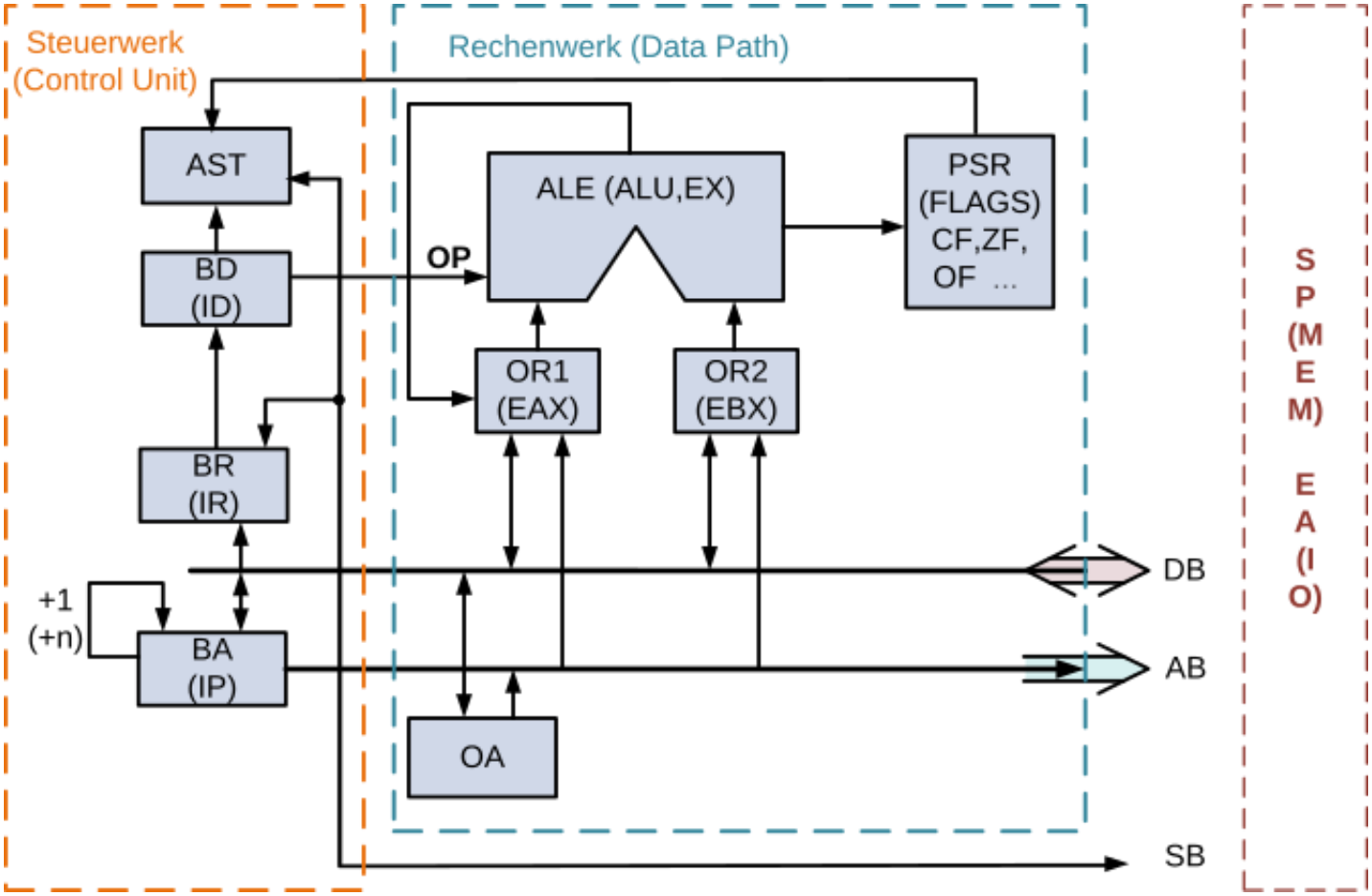
Aktuelles Intel Mainboard-Layout

- Aktuelle Intel-Architektur integriert North Bridge in den Prozessor
- Grafikprozessor optional in der CPU enthalten
- Hauptspeicher und Grafikkarte direkt an den Prozessor angebunden

siehe:

http://www.intel.de/content/www/de/de/chipsets/performance-chipsets/z170-chipset.html

Prozessorgrundstruktur



ALE: Arithmetisch-logische Einheit

AST: Ablaufsteuerung

BA: Befehlsadresse

BD: Befehlsdecoder

BR: Befehlsregister

OA: Operandenadresse

OP: Operation

OR1, OR2: Operandenregister

OA: Operandenadresse

PSR: Prozessorstatus-register

DB: Datenbus

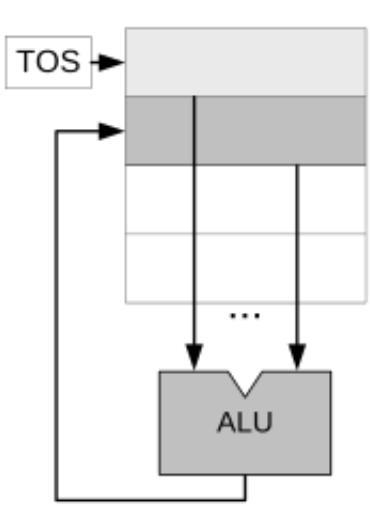
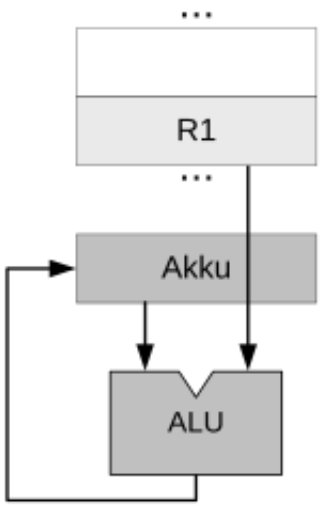
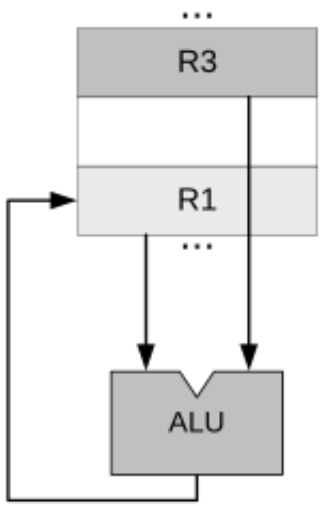
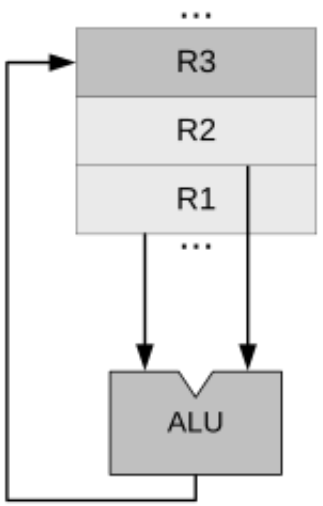
AB: Adressbus

SB: Steuerbus

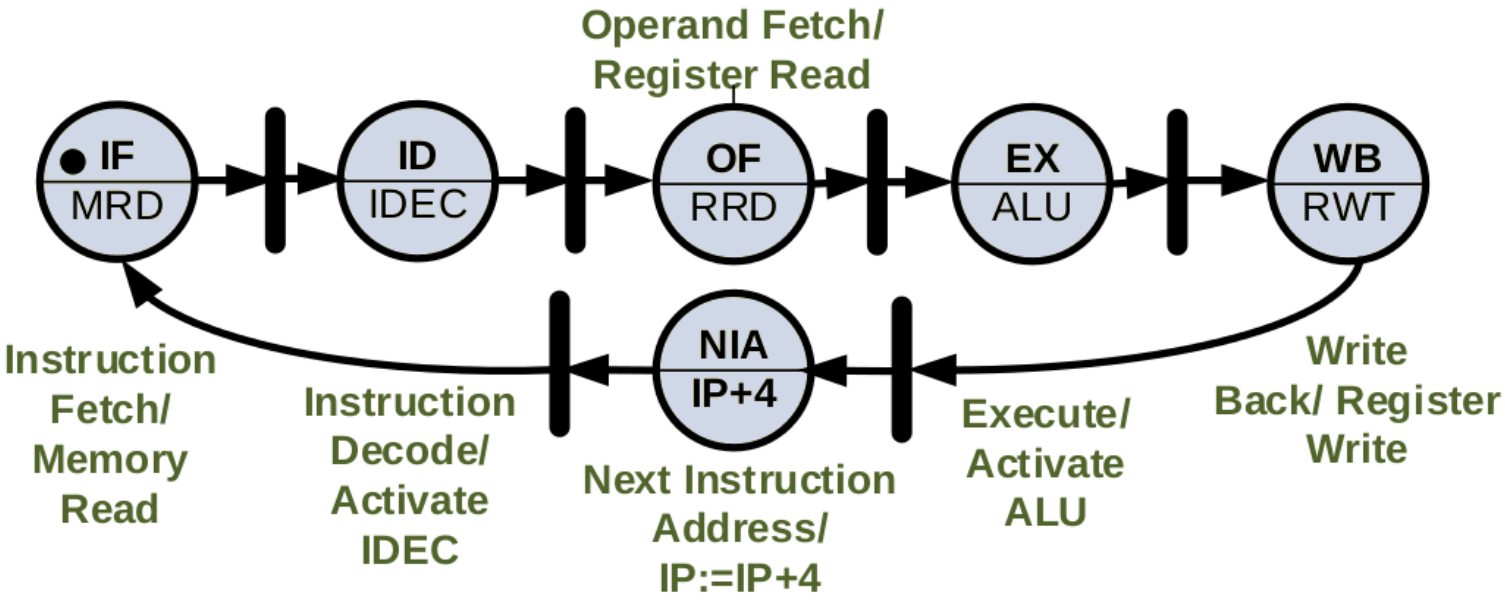
SP: Speicher

Beispiel x86 Assembler: ADD EAX, EBX

Klassifikation nach Operandenadressierung

0-Operand (Stack)	1-Operand (Akkumulator)	2-Operand	3-Operand
			
Add	Add R1	Add R1, R3	Add R3, R1, R2

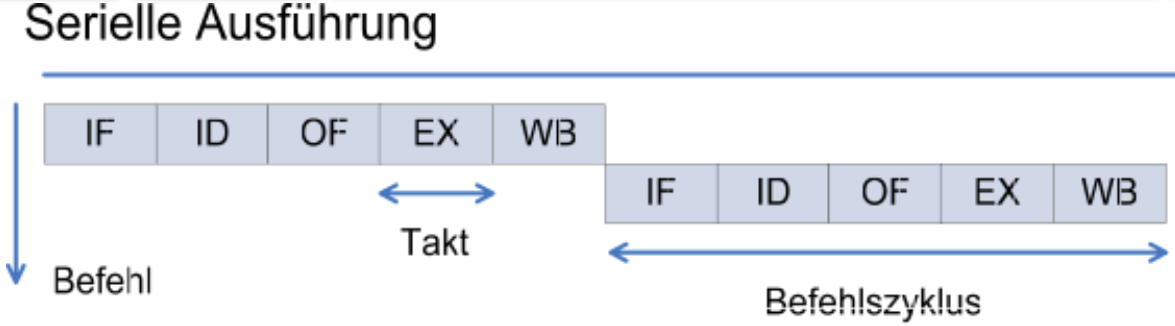
Befehlsabarbeitung



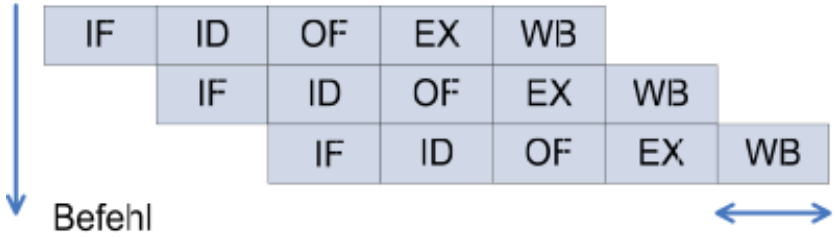
- Arithmetik-Logik-Befehl (Bsp. ADD EAX, EBX)
- Befehl Lesen (IF) – durch Speicher Lesen (MRD) des Maschinencodes
- (Bsp. ADD, 00...00110011b, fiktiv))
- Befehl Dekodieren (ID) – durch Aktivierung Befehlsdecoder IDEC
- 1. Operand: von Register OR1 (RRD) (z.B. EAX)
- 2. Operand: von Register OR2 (RRD) (z.B. EBX)
- ALU-Operation (EX) (Bsp. Addition)
- Ergebnis nach Register OR1 (WB, RWT) (z.B. EAX)
- Erzeugen der nächsten Befehlsadresse (NIA) (z.B. IP:=IP+4)

Pipelining

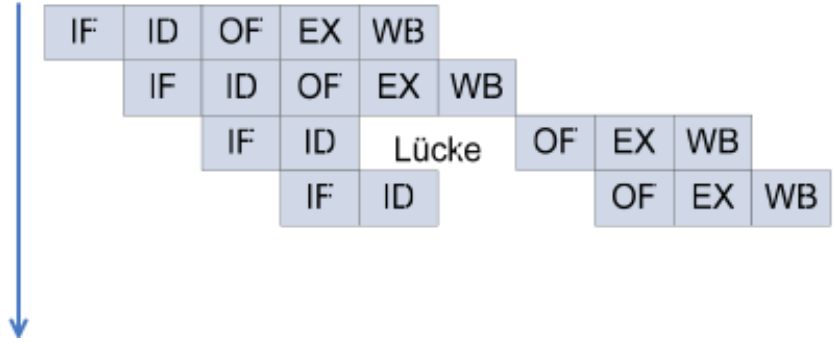
Serielle Ausführung



Ausführung in der Pipeline



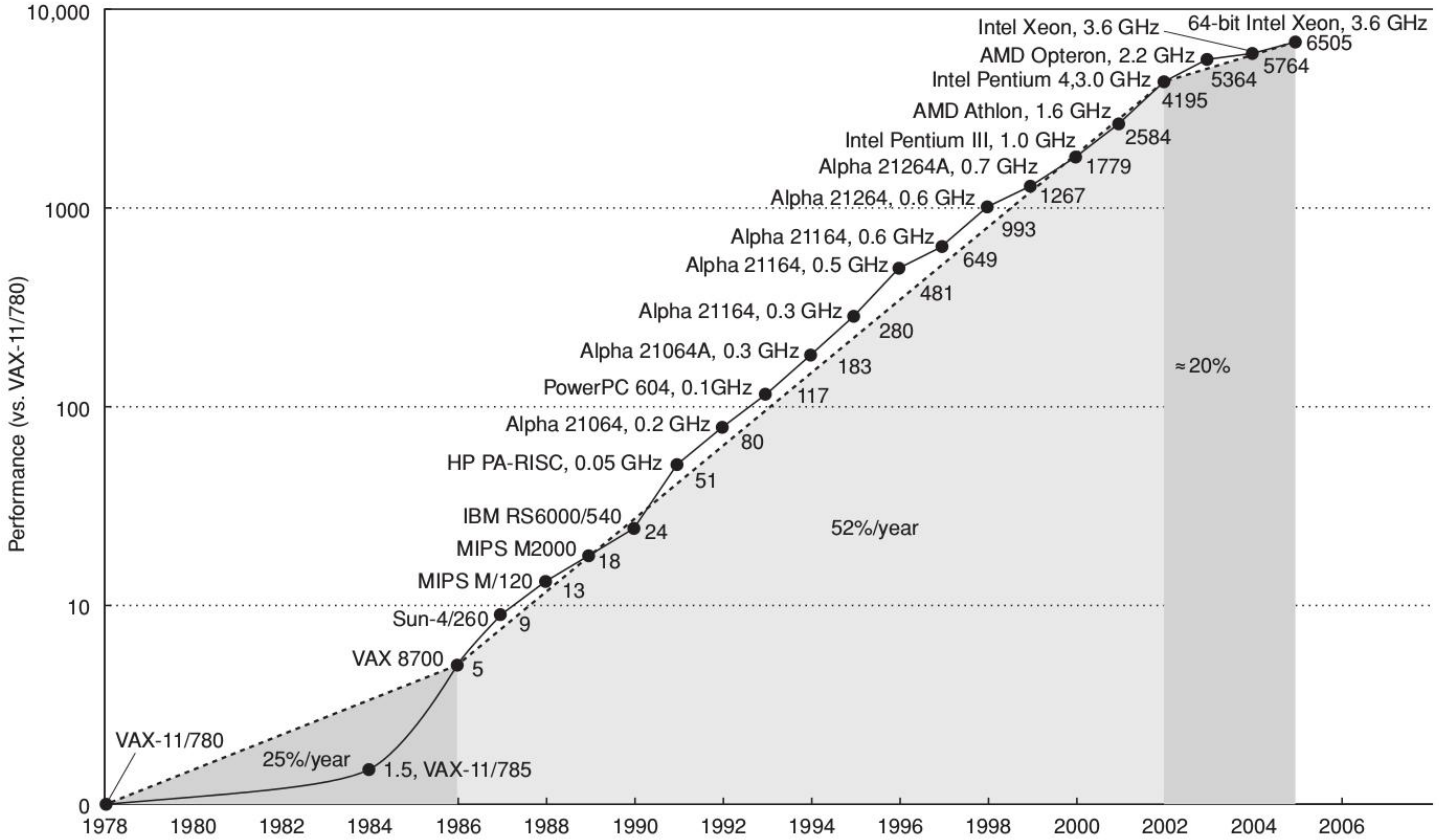
Datenkonflikt RaW




```
ADD R1,R2
ADD R3,R4
ADD R5,R3
```

Operand nicht aktuell

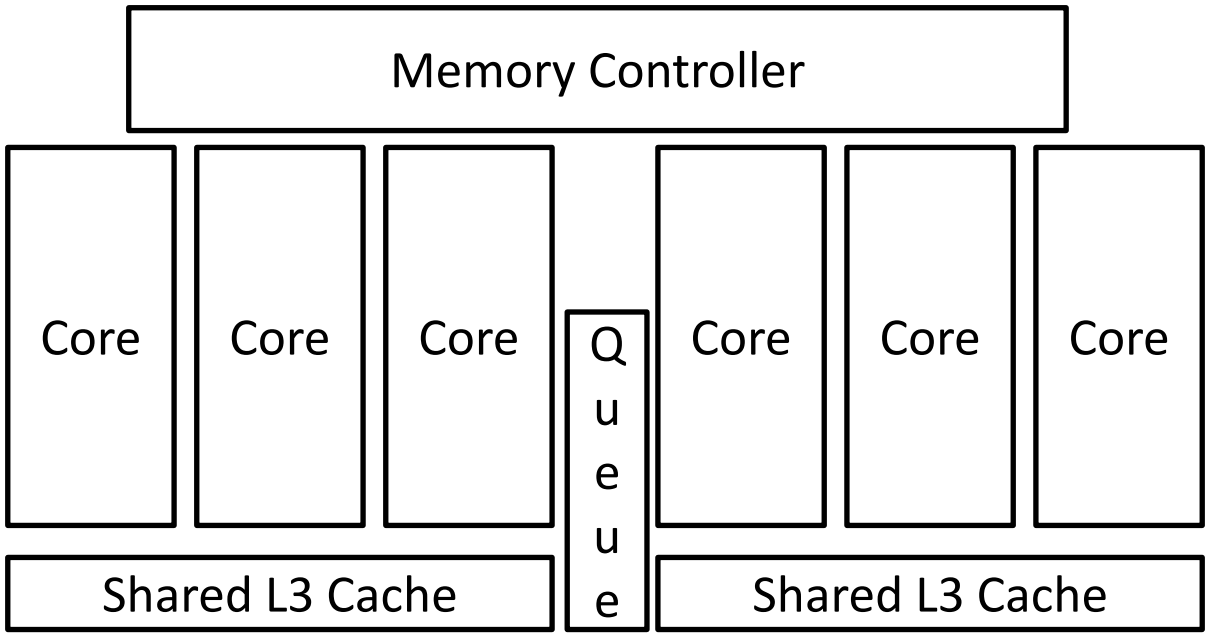
Leistungszuwachs von Prozessoren



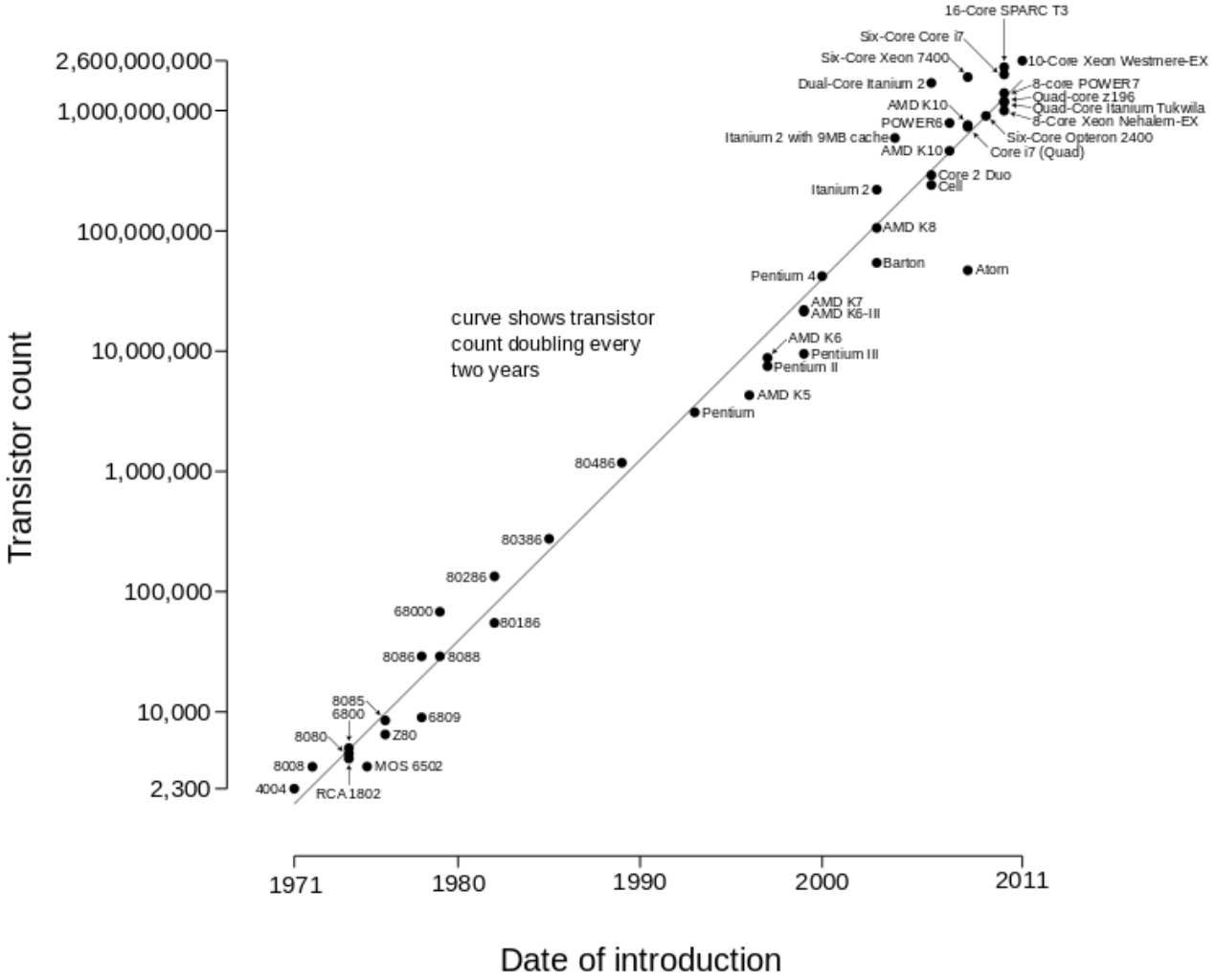
Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

Prozessor Leistungszuwachs seit den 1980er Jahren gemessen mit dem SPECint Benchmark im Vergleich zu einem VAX 11/780

Leistungssteigerung durch multiple Prozessorkerne



Mooresches Gesetz



https://commons.wikimedia.org/wiki/File:Transistor_Count_and_Moore's_Law_-_2011.svg

- Gordon Moore (* 3. Januar 1929 in San Francisco)
- Intel-Mitbegründer
- Mooresches Gesetz (1965) besagt, dass sich die Anzahl der Transistoren eines integrierten Schaltkreises etwa alle zwei Jahre verdoppelt

Amdahls Law:

S_n: Speedup bei n gleichartigen Prozessoren

n: Anzahl an Prozessoren

A: relativer Anteil an der Ausfühungszeit

$o(n)$: Overhead durch die Kommunikation und Synchronisation bei n Prozessoren

$$S_n = \frac{1}{A_{seriell} + o(n) + \frac{A_{parallel}}{n}}$$

Vom Quellcode zum Maschinencode

Grundproblem:

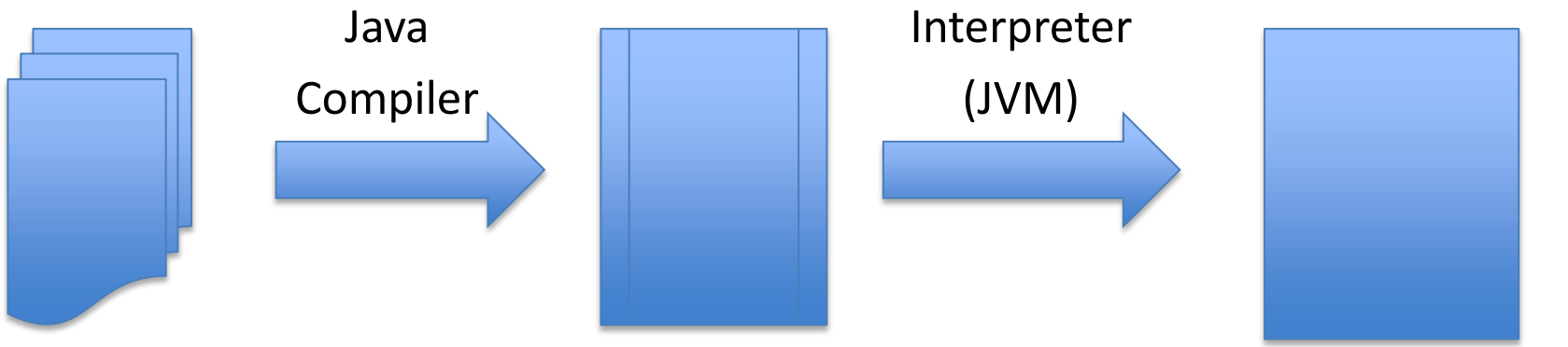
- Maschinensprache für den Menschen höchst umständlich
- Maschinen können aber nur Maschinensprache (primitive Befehle) verstehen

Lösung:

- Menschen nutzen einen anderen Befehlssatz als Sprache L1 (Bsp.: C++)
- Maschinen verarbeiten eine Übersetzung (Compiler) bzw. Interpretation (Interpreter) von L1, hier als L0 bezeichnet.
- **Compiler**:
  - Vollständige Übersetzung des Programms in L1 zu L0
  - Quellprogramm in L1 wird verworfen
  - Zielprogramm in L0 wird in Speicher geladen und ausgeführt
- **Interpreter**:
  - Jede L1 Anweisung wird analysiert, dekodiert und unmittelbar in L0 ausgeführt

Beispiel: Java

Auch hybride Ansätze möglich: Beispiel Java

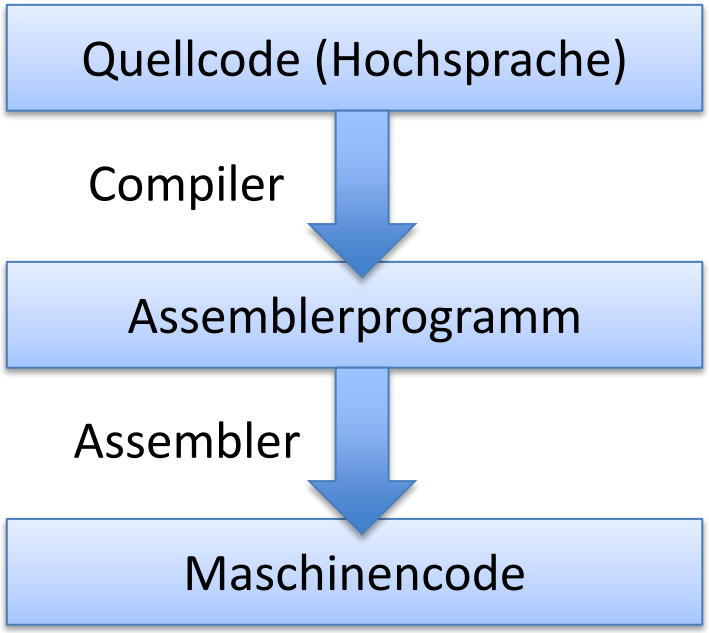


- Quellcode wird von Java Compiler in Bytecode übersetzt
- JVM interpretiert den Bytestream als nativen Maschinencode, der vom OS ausgeführt werden kann

Vom Programm zum Maschinenprogramm

Vereinfachter, abstrakter Ablauf:

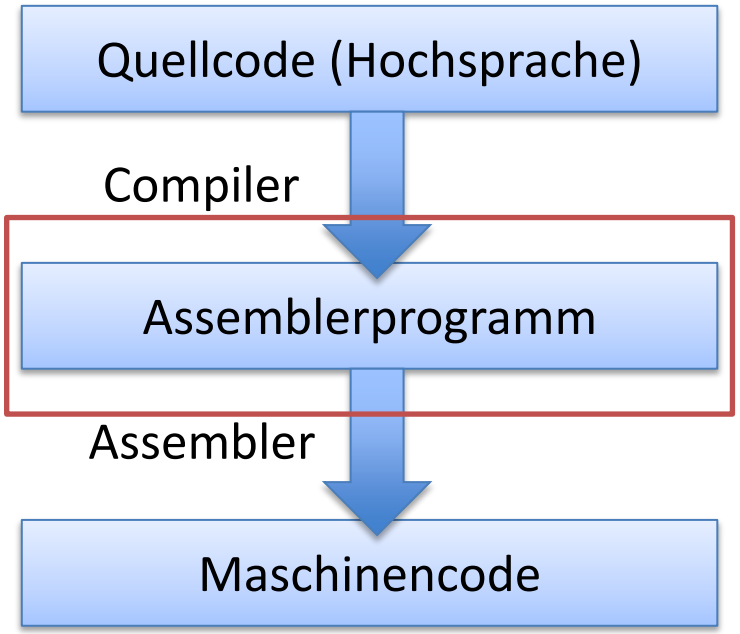
- Hochsprache (Bsp.: C++) wird mittels Compiler in eine Assemblersprache übersetzt
- Compiler analysiert das Programm und erzeugt Assemblercode
  - Für Menschen verständlicher Maschinencode
- Assembler übersetz Assemblercode in Maschinencode
  - Maschinensprache bestehend aus 0 und 1
  - Für Menschen schwer verständlich



Vom Programm zum Maschinenprogramm

Vereinfachter Ablauf:

- Hochsprache (C++, Java,…) wird mittels Compiler in eine Assemblersprache übersetzt
- Compiler analysiert das Programm und erzeugt Assemblercode
  - Für Menschen verständlicher Maschinencode
- Assembler übersetz Assemblercode in Maschinencode
  - Maschinensprache bestehen aus 0 und 1



Fokus nun auf Maschinenebene & Assemblerprogrammierung!

Assemblersprache:

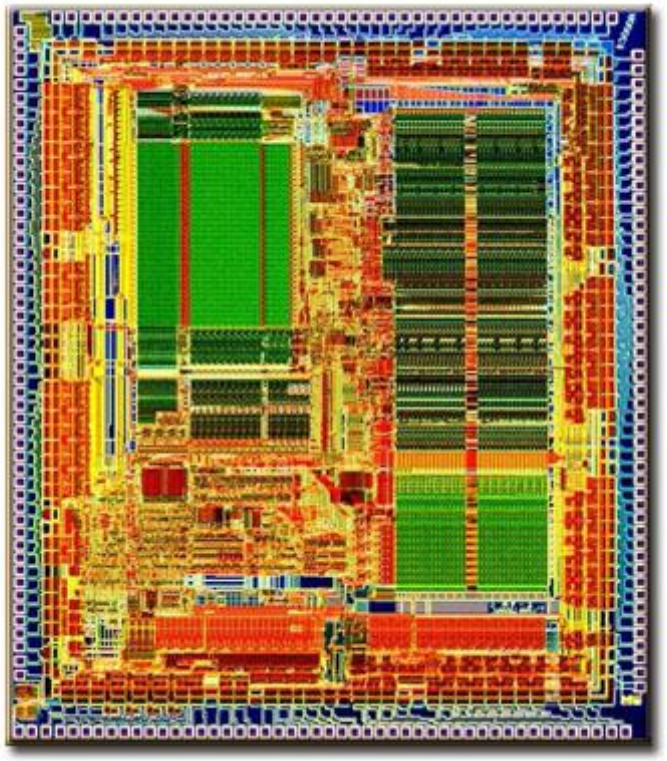
- Hardwarenahe Programmiersprache
- Wird von Assembler direkt in ausführbaren Maschinencode umgewandelt
- Alle Verarbeitungsmöglichkeiten des Mikrokontrollers werden genutzt
- Hardwarekomponenten können direkt angesteuert werden
- Erlauben Namen für Instruktionen, Speicherplätze, Sprungmarken, etc.
- I.d.R. effizient, geringer Speicherplatzbedarf
- Anwendung:
  - Gerätetreiber
  - Eingebettete Systeme
  - Echtzeitsysteme
  - Neue Hardware (Keine Bibliotheken vorhanden)
  - Programmierung von Mikroprozessoren (Bsp.: MIPS)

```
lw    $t0,($a0)
add   $t0, $t1,$t2
sw    $t0,($a0)
jr    $ra
```

Beispiel: Assemblercode

MIPS-Architektur (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages)

- Mikroprozessor ohne Pipeline-Sperren
- RISC-Prozessorarchitektur
- 1981: von John Hennessy entwickelt (Standford-Universität)
- 1984: Weiterentwicklung durch MIPS Computer Systems Inc., heute MIPS Technologies.
- 1991: Mit R4000 auf 64 Bit erweitert (ursprünglich 32-Bit)
- Früher: vor allem in klassischen Workstations und Servern
- Heute: Haupteinsatzbereich im Bereich Eingebettete Systeme



MIPS R3000

RISC / CISC

Unterscheidung zwischen:

- **RISC** = Reduced Instruction Set Computer
- **CISC** = Complex Instruction Set Computer

Historie:

- Beginn: Mikroprozessoren waren alle RISC Prozessoren.
- Dann: Integration immer weiterer Funktionen
  => Immer komplexere Prozessor Designs
- Analyse zeigte:
  - ca. 95% aller Maschinenbefehle in Programmen sind einfache Befehle
- Also:
  - RISC: Entfernen komplexer Befehle (durch Software realisiert)
    - schnellere Ausführung von Befehlen (keine Interpretation nötig)
  - CISC: Prozessoren haben oft einen RISC Kern
    - Komplexe CISC-Instruktionen werden in Folge von RISC-Instruktionen übersetzt.

CISC vs. RISC

CISC CPUs:

- Motorola 68000, Zilog Z80, Intel x86 Familie
- ab Pentium 486 allerdings mit RISC Kern und vorgeschaltetem Übersetzer in RISC Befehle.

Systeme mit RISC CPU:

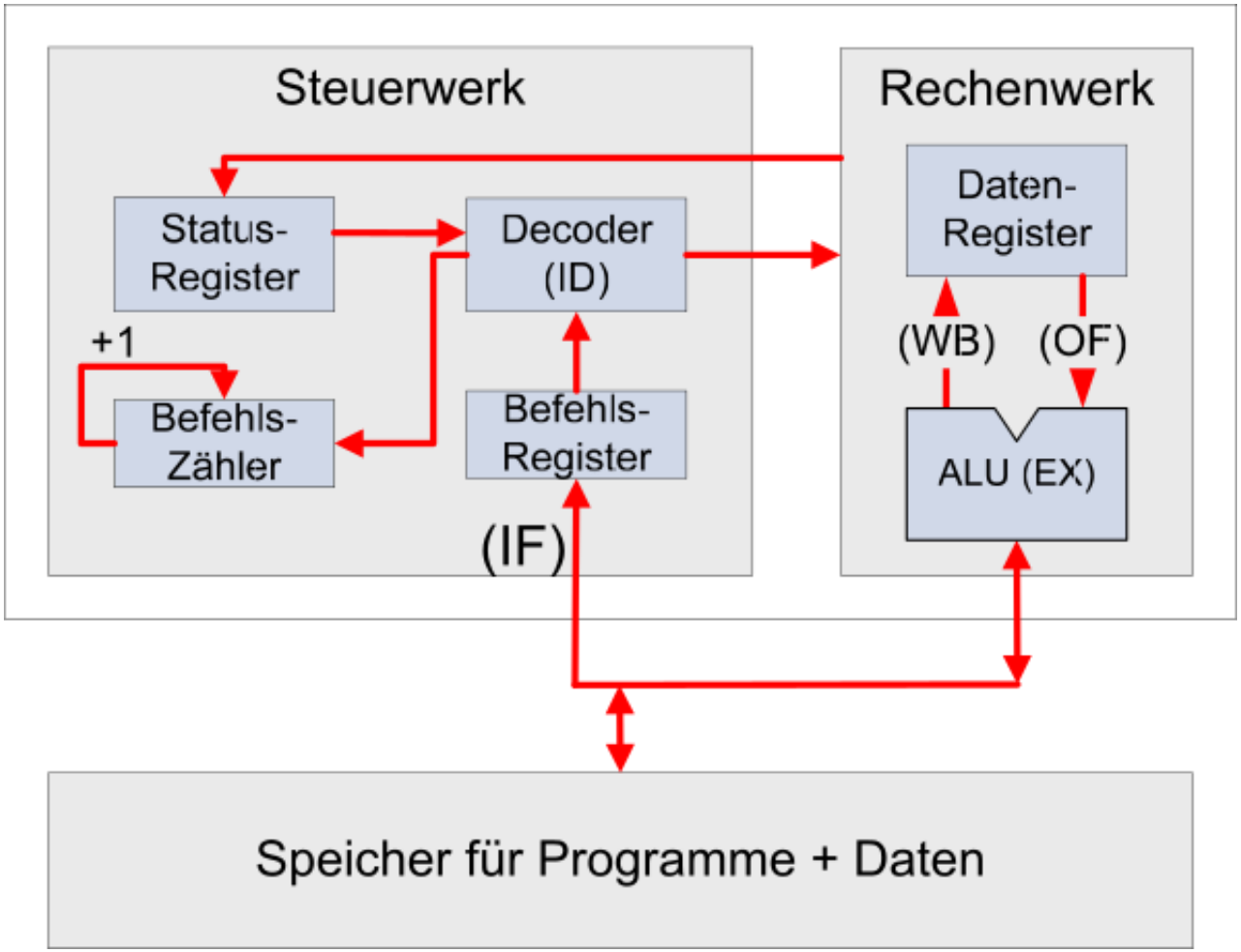
- Leistungsstarke eingebetteten Systemen (Druckern, Router)
- Workstations
- Supercomputern der Spitzenklasse
- Beispiele:
  - Cisco: Router und Switches bis zur Enterprise-Klasse
  - IBM: Supercomputer, Mittelklasse-Server, Workstations
  - Nintendo: Gamecube, Wii Spielkonsolen
  - Microsoft: Xbox 360 Spielkonsole
  - Motorola: Bordcomputer für PKW und andere Fahrzeuge

Systeme mit RISC CPU: ARM

ARM (Advanced RISC Machines)

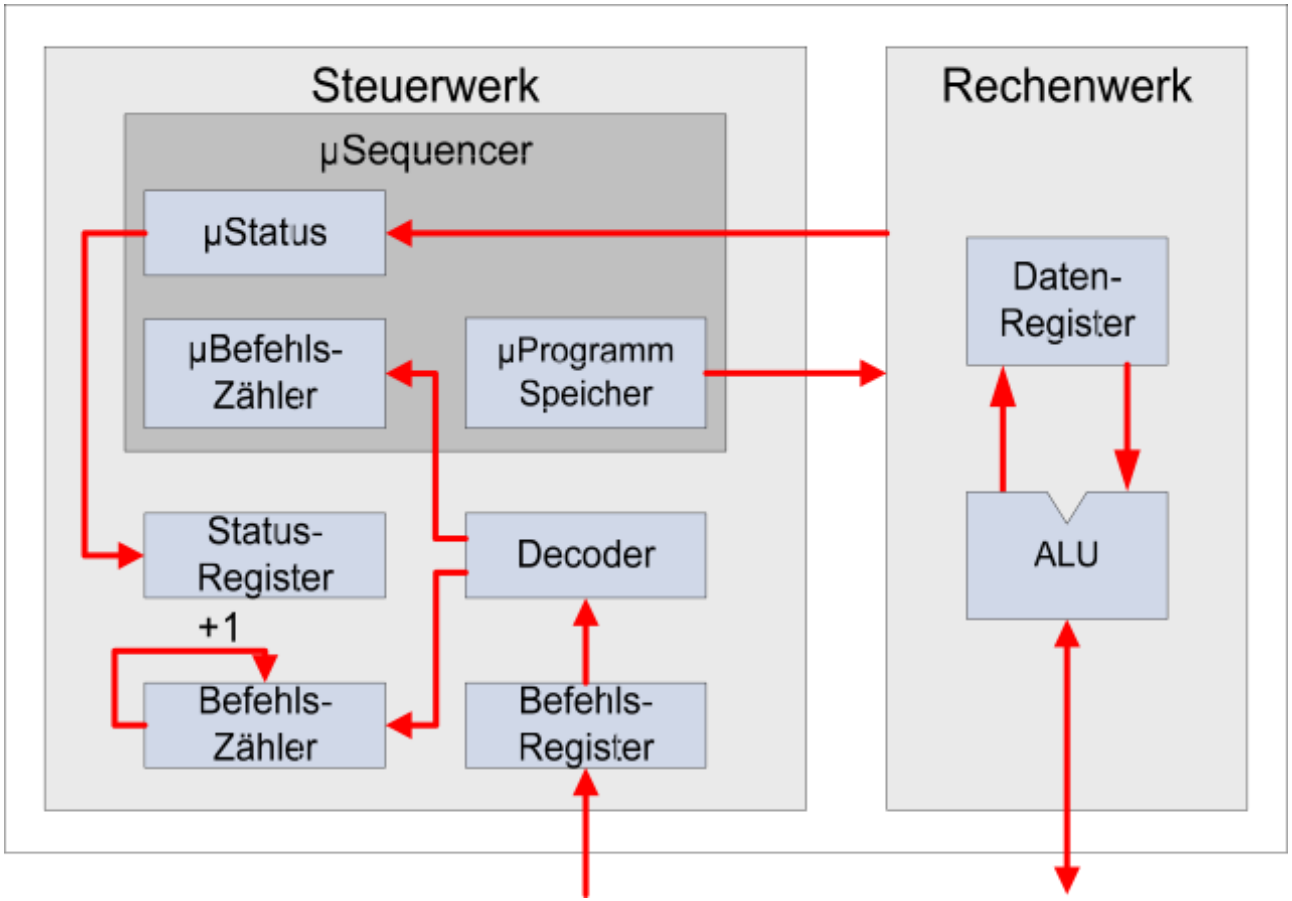
- hohe Leistung, geringen Stromverbrauch, niedrige Kosten
- Typisch 400 MHz – 2 GHz (2011)
- Milliarden ARM-CPUs im Einsatz in
  - Apple: iPods (ARM7TDMI SoC), iPhone (Samsung ARM1176JZF), IPod touch (ARM11)
  - Canon: IXY Digital 700 Kamera (ARM-basiert)
  - Hewlett-Packard: HP-49/50 Taschenrechner (ARM9TDMI)
  - Linksys: NSLU2 Netzwerkspeicher/NAS [Intel XScale IXP420]
  - Nintendo: Game Boy Advance (ARM7), Nintendo DS (ARM7, ARM9)
  - Sony: verschiedene Mobiltelefone, Network Walkman (Eigenentwicklung, ARM-basiert)

Reduced Instruction Set Computer



- entspricht vereinfachter Prozessor-grundstruktur der vorangegangen Folien

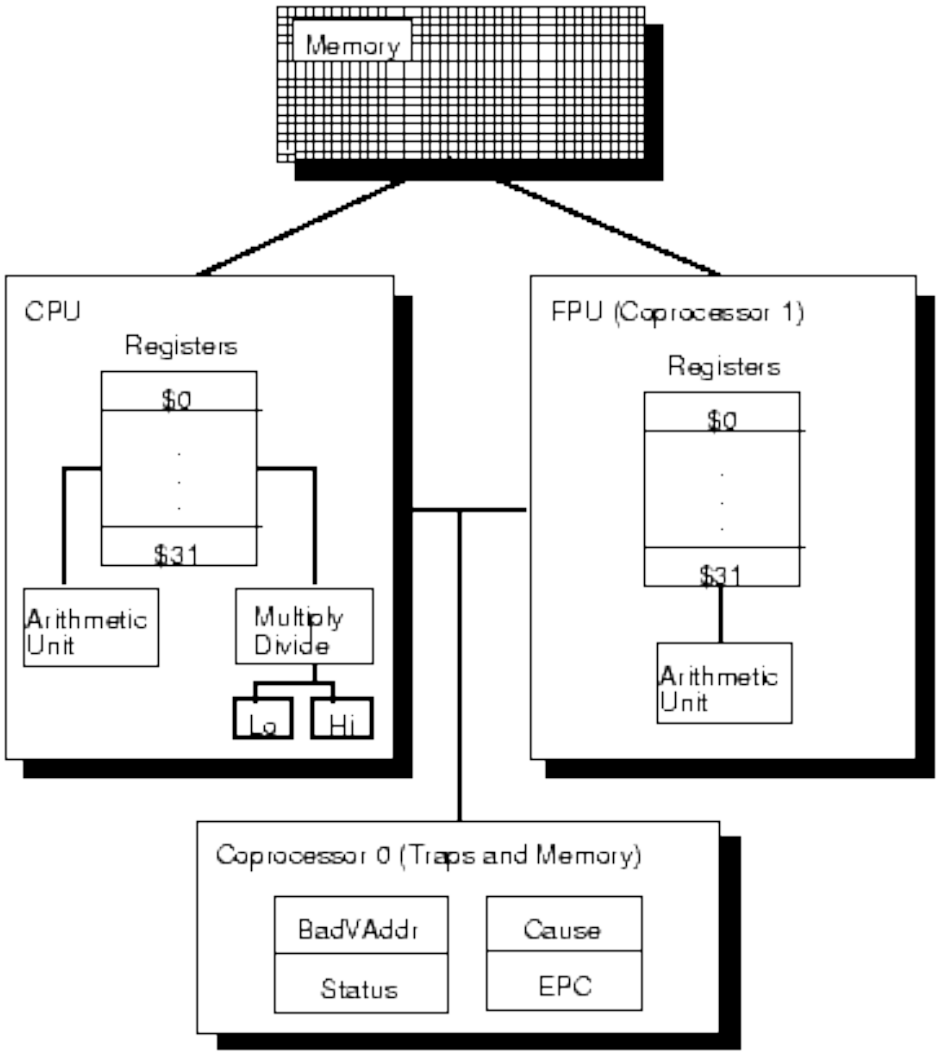
Complex Instruction Set Computer



- Steuerwerk muss komplexer ausfallen, um komplexe Instruktionen in ihre Bestandteile aufzuspalten und auszuführen

Struktur des MIPS Prozessors

Nicht alle Funktionen sind im Prozessor selbst realisiert, sondern in Coprozessoren ausgelagert.



MIPS-Register (32 Bit breit)

Name	Nummer	Verwendung
$zero	0	Enthält den Wert 0, kann nicht verändert werden.
$at	1	temporäres Assemblerregister. (Nutzung durch Assembler)
$v0	2	Funktionsergebnisse 1 und 2 auch für Zwischenergebnisse
$v1	3	
$a0	4	Argumente 1 bis 4 für den Prozeduraufruf
$a1	5	
$a2	6	
$a3	7	
$t0,…,$t7	8-15	temporäre Variablen 1-8. Können von aufgerufenen Prozeduren verändert werden.

MIPS-Register 16-31

Name	Nummer	Verwendung
$s0,…, $s7	16 … 23	langlebige Variablen 1-8. Dürfen von aufgerufenen Prozeduren nicht verändert werden.
$t8,$t9	24,25	temporäre Variablen 9 und 10. Können von aufgerufenen Prozeduren verändert werden.
$k0,k1	26,27	Kernel-Register 1 und 2. Reserviert für Betriebssystem, wird bei Unterbrechungen verwendet.
$gp	28	Zeiger auf Datensegment
$sp	29	Stackpointer Zeigt auf das erste freie Element des Stacks.
$fp	30	Framepointer, Zeiger auf den Prozedurrahmen
$ra	31	Return Adresse

Adressierung

Adressierung: byteweise!

Wort mit Adresse n => Nächstes Wort: Adresse n+4

Adresse	**...**	**0xA8**	**0xA9**	**0xAA**	**0xAB**	**0xAC**	**0xAD**	**0xAE**	**0xAF**	
Bytegrenze	...	byte 168	byte 169	byte 170	byte 171	byte 172	byte 173	byte 174	byte 175	
Wortgrenze	...	word 42				word 43				...