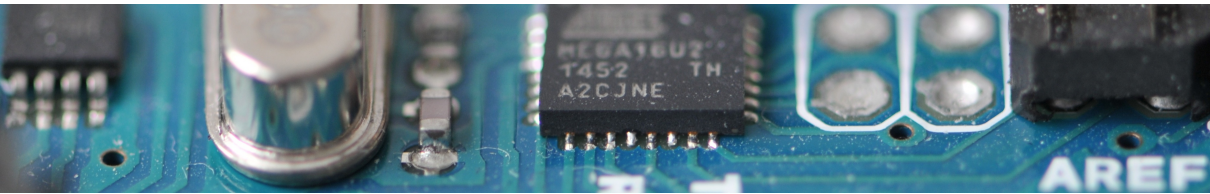
universität innsbruck



Rechnerarchitektur

Prozessorarchitekturen

Univ.-Prof. Dr.-Ing. Rainer Böhme

Wintersemester 2020/21 · 9. Dezember 2020

Vorbemerkung

Das heutige Thema ist eine Erweiterung Ihres Horizonts.

Behalten Sie den Überblick!

Kein Beispiel lässt sich direkt mit Ihren Kenntnissen der ARM-Architektur kombinieren oder umsetzen.

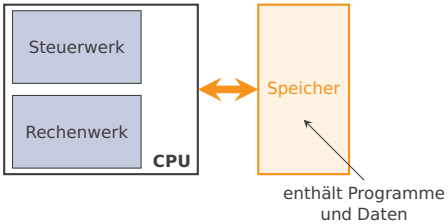
Im Proseminar und in der kommenden Vorlesung zur Ein- und Ausgabe nutzen und vertiefen wir weiterhin die ARM-Assemblerprogrammierung.

Gliederung heute

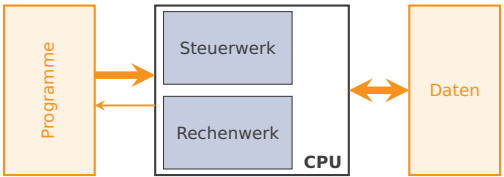
1. **Klassifikation von Prozessorarchitekturen**
2. Intel x86-Architektur
3. Datenparallele Architekturen

Klassifikation nach Anbindung des Speichers

Von-Neumann-Architektur



Harvard-Architektur

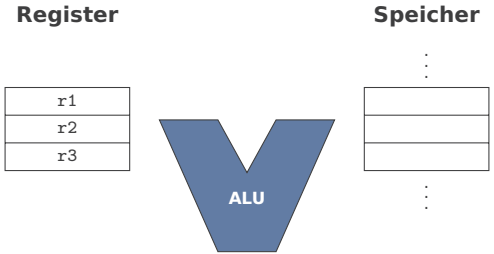


nach dem Relaisrechner Mark I von Aiken (Harvard Universität) und IBM (1943/44)

Klassifikation nach Anbindung der ALU

Einteilung von Mikroarchitekturen nach möglichen Quellen und Zielen von arithmetisch-logischen Operationen:

- Register/Register-basiert
- Register/Speicher-basiert
- Akkumulator-basiert
- Stapel-basiert



Register/Register-basierte Architekturen

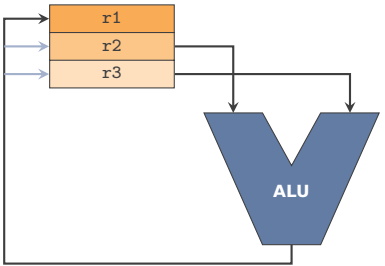
Werte aus zwei Registern werden miteinander verknüpft.
Speicherzugriffe erfolgen separat (*Load-Store*-Architekturen).

Register

Op. 1: Register
Op. 2: Register

Ergebnis: Register

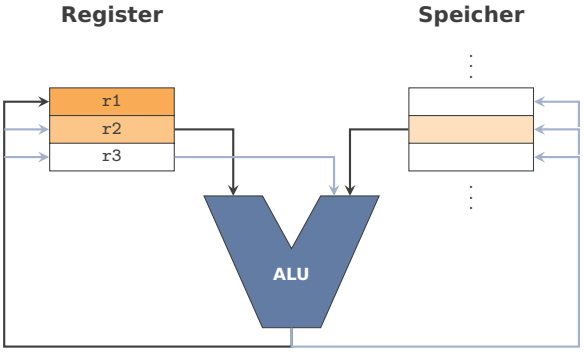
Beispiele: ARM, MIPS, DLX, RISC-V, SPARC, AVR



Register/Speicher-basierte Architekturen

Ein Wert aus einem der Register wird mit einem Wert aus einem Register oder dem Speicher verknüpft.

Op. 1:	Register
Op. 2:	Register o. Speicher
Ergebnis:	Register o. Speicher
Beispiele:	Intel x86, Motorola 68K, PowerPC



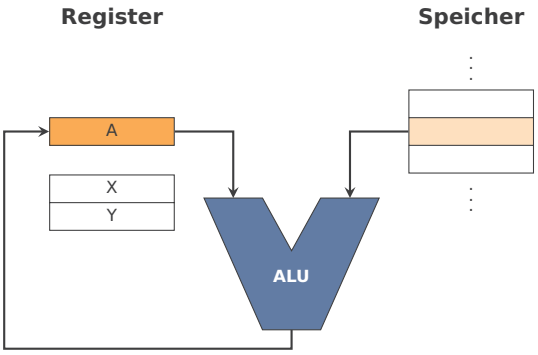
Akkumulator-basierte Architekturen

Der Wert aus einem Spezialregister (**Akkumulator**) wird mit einem Wert aus dem Speicher verknüpft. Das Ergebnis kommt immer in den Akkumulator.

Op. 1: Akkumulator
Op. 2: Speicher

Ergebnis: Akkumulator

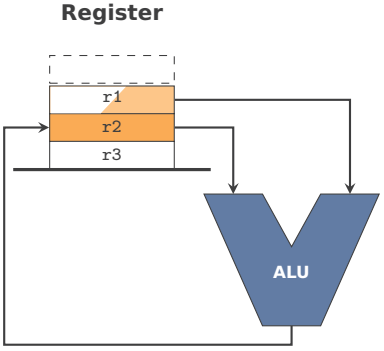
Beispiele: Intel 4040, MOS 6502, Zilog Z80



Stapel-basierte Architekturen

Alle Register werden als Stapel organisiert. Die **obersten beiden** Werte auf dem Stapel werden miteinander verknüpft und das Ergebnis wieder oben auf dem Stapel abgelegt.

Op. 1:	Wert unter Stapelspitze
Op. 2:	Stapelspitze
Ergebnis:	(neue) Stapelspitze
Beispiele:	x87 FPU, Java VM, Ethereum VM



Klassifikation nach Komplexität des Befehlssatzes

RISC: Reduced Instruction Set Computer

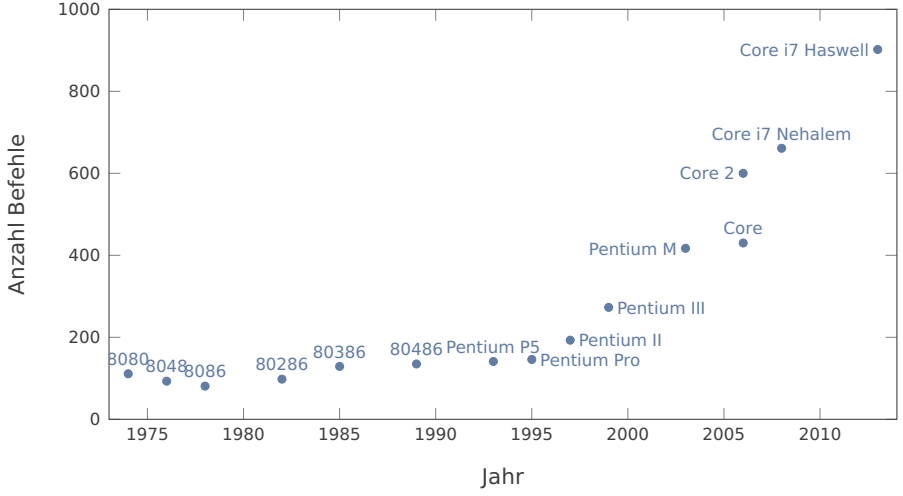
- wenige elementare Maschinenbefehle
- ermöglichen schlanke Pipelines: Richtwert ist ein Taktzyklus pro Stufe
- kompakte Instruktionskodierung, oft **orthogonal**
  ($\rightarrow$ Einschränkungen bei Immediate-Werten und Addressierungsarten)
- Load-Store-Architekturen

CISC: Complex Instruction Set Computer

- viele mächtige Spezialbefehle, z. T. in **Mikroprogrammen** realisiert
- Optimierung schwierig
- organisches Wachstum der Befehlssätze und Instruktionskodierung
- Compiler nutzen oft nur eine Untermenge der verfügbaren Befehle.

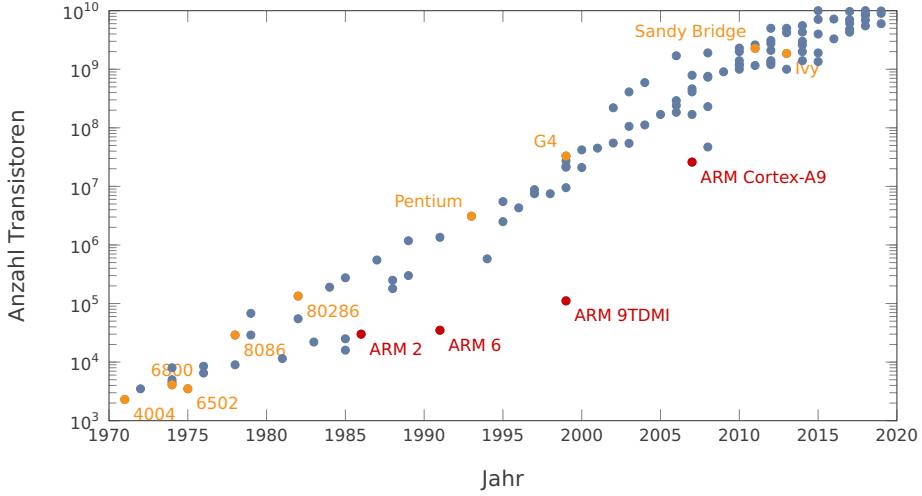
Kompromisse: RISC und CISC existieren heute selten in Reinform.

Anzahl der Instruktionen von CISC-Prozessoren



Datenquelle: Wikipedia 2016

Anzahl der Transistoren in Mikroprozessoren



Datenquelle: Wikipedia 2020

Klassifikation nach Art der Parallelverarbeitung

Anzahl der Datenströme		Anzahl der Befehlsströme: 1 *single instruction*	Anzahl der Befehlsströme: >1 *multiple instruction*
1	*single data*	**SISD**	MISD
>1	*multiple data*	SIMD	MIMD

Flynn 1966

Gliederung heute

1. Klassifikation von Prozessorarchitekturen
2. **Intel x86-Architektur**
3. Datenparallele Architekturen

Entwicklung der x86-CISC-Architektur

1978 **Intel 8086** erscheint (**16-Bit**-Architektur)

1980 Intel **8087 FPU (Koprozessor für Gleitkommazahlen)**

1982 Intel 80286, 24 Bit Adressraum, neue Instruktionen

1985 Intel **80386** (**32-Bit**-Architektur), neue Adressierung

1989 Intel 80486 (weniger Mikrokode, **Integration der FPU** ab 486 DX)

1993 Intel **Pentium**: Integration von **RISC**-Prinzipien (1995: Pentium Pro)

1997 Pentium II mit 57 neuen **MMX-Instruktionen** (Ganzzahl-**SIMD** für Multimedia)

1999 Pentium III mit 70 neuen **SSE-Instruktionen** (Gleitkomma-**SIMD**)

2001 Pentium 4 mit 144 neuen SSE2-Instruktionen

2003 **64-Bit**-Architektur von **AMD**, seit 2004 von Intel unterstützt

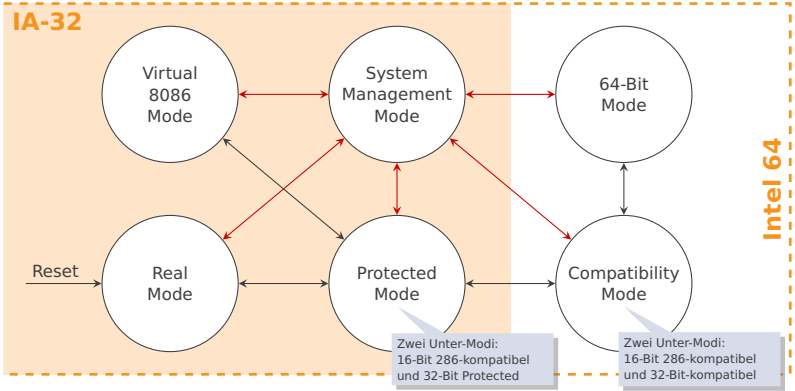
2006 Hardwareunterstützte **Virtualisierung** (AMD-V bzw. Intel VT-x)

2015 Vertrauenswürdige Laufzeitumgebung (TEE) – „Enklave“

Fett gedruckte Begriffe sollte jede InformatikerIn kennen und sind prüfungsrelevant.

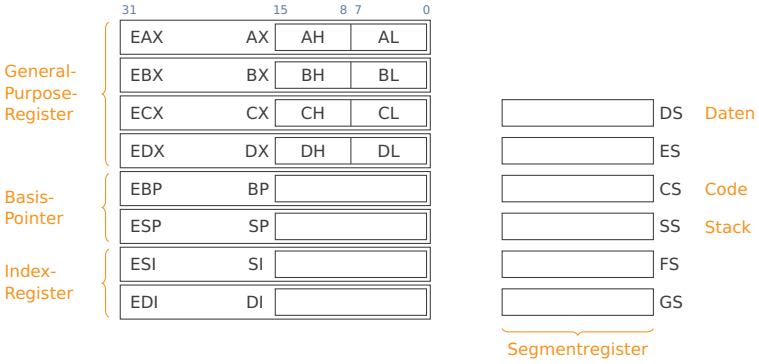
Problemfeld Abwärtskompatibilität

am Beispiel der Betriebsmodi aller modernen x86-Prozessoren



Alternative: Kontinuität; sonst neu kompilieren, zur Not emulieren

Registersatz



Im 16-Bit-Modus steuern **Segmente** die höchstwertigen Adressbits.

(vereinfacht: 32-Bit-Modus; ohne Koprozessor, Kontrollregister, 128-Bit-Media-Register)

ALU-Anbindung

ALU-Befehle haben in der Regel **zwei Operanden:**

- Der erste Operand ist gleichzeitig Quelle und Ziel.

```
ADD   AX, BX   ; a = a + b   (Gleichheitszeichen ist Zuweisung)
ADD   DX, 13   ; d = d + 13
```

- Maximal ein Operand darf ein Speicherwort sein.

```
ADD   AX, mem16   ; a = a + m16
ADD   mem16, AX   ; m16 = m16 + a
ADD   mem16, 42   ; m16 = m16 + 42
```

- Kürzere Spezialbefehle:

```
INC   ESI    ; i = i + 1
DEC   mem8   ; m8 = m8 - 1
```

Achtung: Einige Befehle sind fest mit definierten Registern verknüpft (z. B. `MUL` und `DIV` für Ganzzahl-Punktoperationen mit AX und DX)

Adressierungsarten

Absolut

```
MOV   EAX, adr        ; Register mit Inhalt an Speicheradresse adr laden
```

Indirekt

```
MOV   EAX, [EBX]      ; Register EBX zeigt auf Speicheradresse
```

Basis mit **Index** (nur {EBP | EBX} plus {ESI | EDI})

```
MOV   EAX, [EBX+ESI]          ; Adresse aus EBX und ESI berechnen
```

. . . sowie **Displacement** (8, 16, 32 Bit) und **Skalierung** (Faktor 2, 4, 8)

```
MOV   EAX, [EBX+ESI*4+2]      ; nützlich für Zugriff auf Felder
```

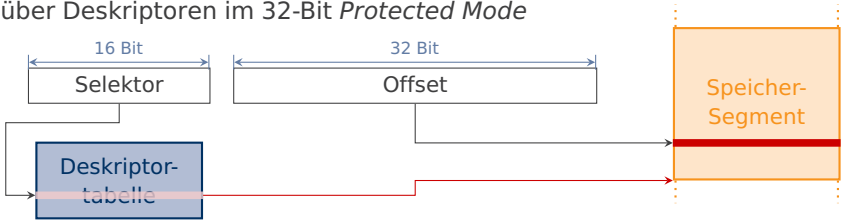
Segmentüberschreibung (kombinierbar mit allen Adressierungsarten)

```
MOV   EAX, ES:[EBX]           ; abweichend vom Datensegment DS
```

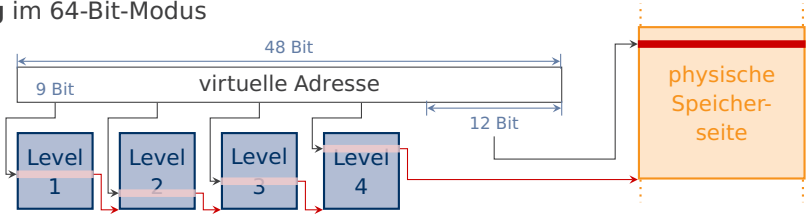
Die Art der Speichersegmentierung ist abhängig vom Betriebsmodus.

Berechnung der physischen Adresse

Segmentierung über Deskriptoren im 32-Bit *Protected Mode*



Mehrstufiges **Paging** im 64-Bit-Modus

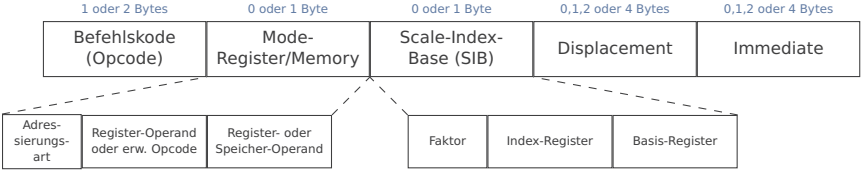


Instruktionskodierung

CISC-Instruktionen setzen sich aus mehreren Komponenten zusammen:

1. Optionale Präfixe

2. Allgemeines Instruktionsformat



Die Länge von x86-Instruktionen variiert zwischen 1 und 16 Bytes.

Stapelorganisation

Ein *full descending* Stapel wird vom Prozessor über das Registerpaar SS:ESP (*stack segment* : *extended stack pointer*) organisiert.

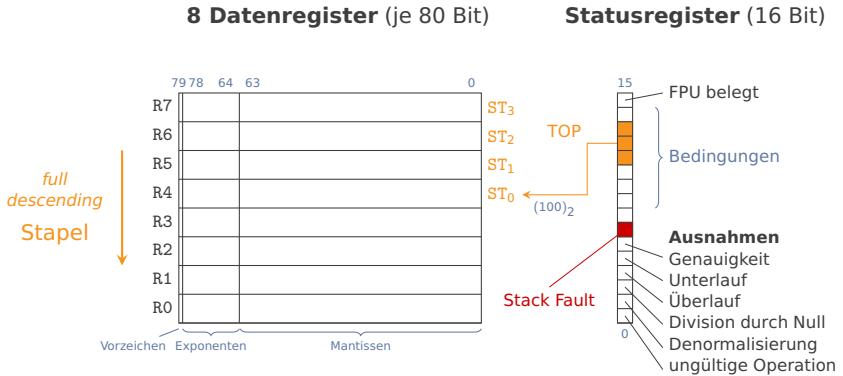
Spezialbefehle (trifft man oft beim Disassemblieren an)

Mnemonic	Kommentar
`PUSH`	Legt Register, Speicherinhalt oder Konstante auf Stapel.
`POP`	Holt Wert vom Stapel in Register oder Speicherinhalt.
`CALL`	Aufruf eines Unterprogramms
`RETN`	Rücksprung von Unterprogramm
`ENTER`	Platz aus dem Stapel für lokale Variablen reservieren
`LEAVE`	Platz für lokale Variablen freigeben

Gleitkommaeinheit

(engl. *floating point unit*, FPU)

Realisierung einer Stapel-basierten Befehlssatzarchitektur



Ausgewählte FPU-Befehle zum Datentransfer

Mnemonic			Kommentar
Gleitkomma-, Ganzzahl, BCD*-Zahl			
`FLD`	`FILD`	`FBLD`	aus dem Speicher nach $\mathtt{ST}_0$ laden
`FST`	`FIST`		aus $\mathtt{ST}_0$ in den Speicher schreiben
`FSTP`	`FISTP`	`FBSTP`	– ” – und vom Stapel löschen (*pop*)

- Die FPU konvertiert automatisch zwischen Zahlendarstellungen.
- Die FPU unterstützt keine Immediate-Werte.
- Wichtige mathematische Konstanten liegen im ROM vor (z. B. `FLDPI` lädt die Kreiszahl π nach $\mathtt{ST}_0$).

*: Die BCD-Kodierung speichert zwei Dezimalstellen pro Byte.
Werte, deren Hex-Darstellung Buchstaben erfordert, sind unzulässig.

Ausgewählte Arithmetik-Befehle der FPU

Mnemonics			Kommentar
FADD	FADDP	FIADD	Gleitkomma-Addition
FSUB	FSUBP	FISUB	Gleitkomma-Subtraktion
FMUL	FMULP	FIMUL	Gleitkomma-Multiplikation
FDIV	FDIVP	FIDIV	Gleitkomma-Division
FSQRT			Gleitkomma-Quadratwurzel
FABS			Absolutbetrag
FRNDINT			auf Ganzzahl runden

- Alle Gleitkomma-Operationen erfolgen nach IEEE 754.
- Varianten mit I erlauben Ganzzahl (Integer) als ersten Operand.
- Für Subtraktion und Division existieren “reverse”-Varianten mit vertauschten Operanden, z. B. FSUBP → FSUBRP, FIDIV → FIDIVR.

Beispiel für FPU-Programmierung

Berechnung eines Skalarprodukts $y = a_1 \cdot b_1 + a_2 \cdot b_2$

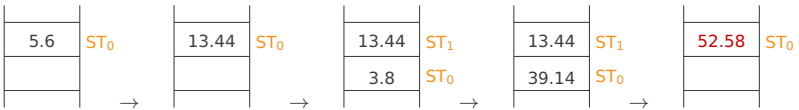
Assembler-Befehlsfolge

```
sprod:
        FLD     adr1
        FMUL    adr3
        FLD     adr2
        FMUL    adr4
        FADDP
```

Belegungsbeispiel

Variable	Wert	Label (Speicheradresse)
a_1	5.6	adr1
a_2	3.8	adr2
b_1	2.4	adr3
b_2	10.3	adr4

Ablauf



Hörsaalfrage

24 82 94 16

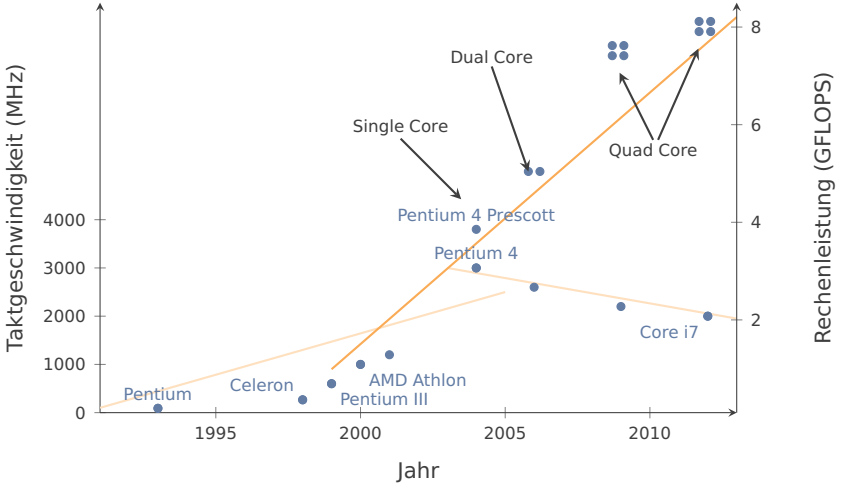
Wie viele freie Speicherplätze benötigen Sie auf dem Stapel, um folgenden Ausdruck mit der FPU effizient zu berechnen (ohne dabei Zwischenergebnisse in den Speicher zu schreiben)?

$$\boldsymbol{y} = \begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

a. 1
b. 2
c. 3
d. 4
e. 8
f. 12
g. 13
h. 16

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

Entwicklung der Leistung von x86-Prozessoren



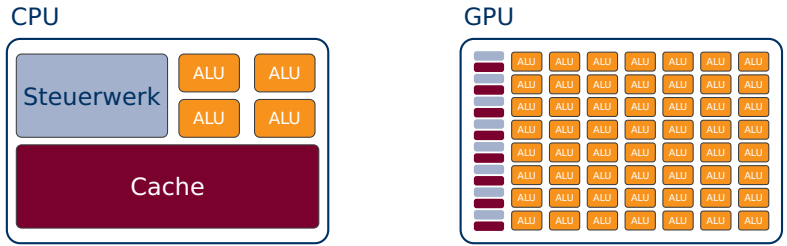
Quelle: `https://en.wikipedia.org/wiki/Comparison_of_Intel_processors`

Gliederung heute

1. Klassifikation von Prozessorarchitekturen
2. Intel x86-Architektur
3. **Datenparallele Architekturen**

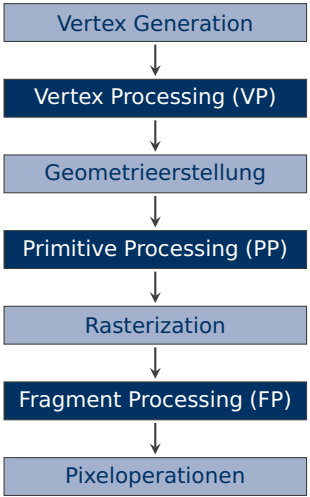
Datenparallele Architekturen am Beispiel von GPUs

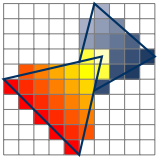
Nutzung der Chipfläche von CPUs und Grafikprozessoren (GPUs)

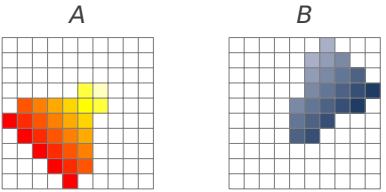


- **Datenparallelität:** Sehr effiziente, da gleichzeitige Bearbeitung vieler gleichartiger Daten auf die gleiche Weise.
- Voraussetzung: **geringe Datenabhängigkeit**; der Kontrollfluss ist weitgehend unabhängig von Zwischenergebnissen.
- Früher: **Vektorprozessoren** in Supercomputern, z. B. Cray

Grafik-Pipeline mit Hardwareunterstützung



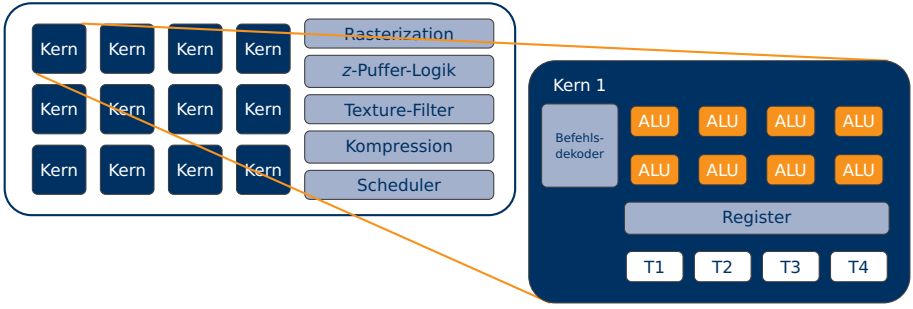




programmierbare Shader

feste Logik

Aufbau moderner GPUs



- Ca. 10^2 Kerne sind in einer Matrix-Struktur organisiert.
- **SIMD**-Prinzip: In jedem Kern teilen sich 10^1 ALUs ein Steuerwerk.
- Hardware-**Threads:** Die ALUs erledigen andere anstehende Aufgaben um die Speicherlatenz (10^2–10^3 Taktzyklen!) zu überbrücken.

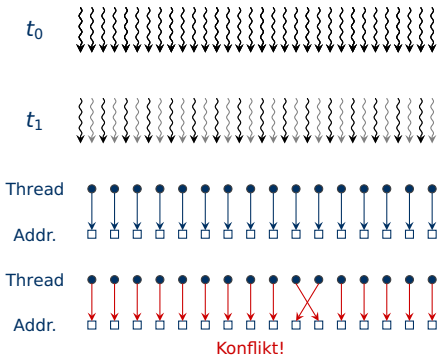
General Purpose GPU: Schnittstelle für **beliebige** Rechenaufgaben

Vertiefung in **Parallele Programmierung**, Pflichtmodul, 4. Semester BSc Informatik

Optimierung für Datenparallelität

(am Beispiel der Nvidia-CUDA-Architektur)

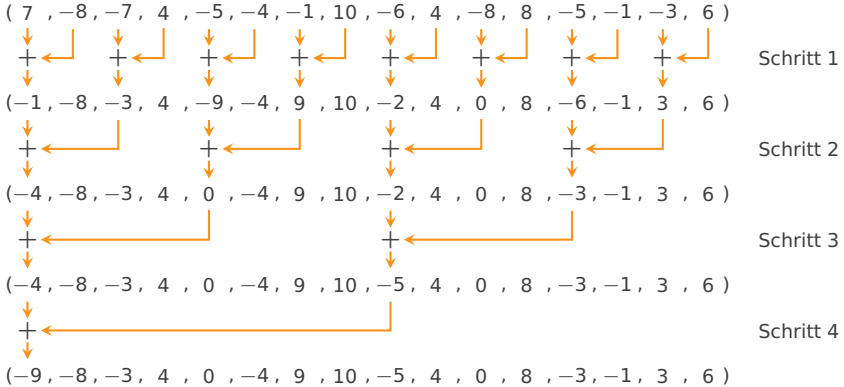
- **SIMD-Breite 32:** mind. 32 Threads müssen das Gleiche tun – bis auf Bedingungen
- **Globaler Speicher** ohne Cache: Alignment nötig, um 16-fache (!) Verzögerung zu vermeiden



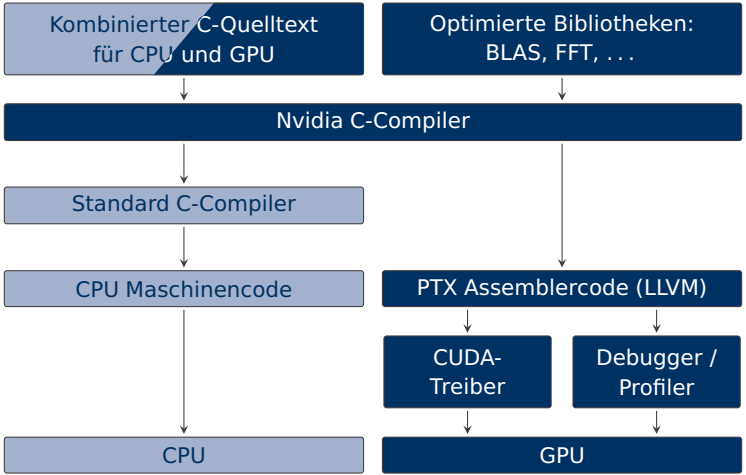
Grundsatz: Besser in Datenlayouts statt in Algorithmen denken !

Beispiel: Parallele Reduktion

Summe der Elemente eines Vektors $x_1 \leftarrow \sum_{i=1}^{|\boldsymbol{x}|} x_i$



Entwicklungswerkzeuge am Beispiel Nvidia CUDA



Spezial- versus Universalhardware

On the Design of Display Processors

T. H. Myer
Bolt Beranek and Newman Inc, Cambridge, Mass.
and
I. E. Sutherland*
Harvard University, Cambridge, Mass.

> *"We have built up the display channel until it is itself a general purpose processor with a display.*
>
> *[...]*
>
> *In short, we have come exactly once around the wheel of reincarnation."*

Fortschritt passiert oft in Kreisläufen mit Wiederentdeckungen.

Myer & Sutherland, *Communications of the ACM*, **11** (6), 1968, S. 412

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)**