

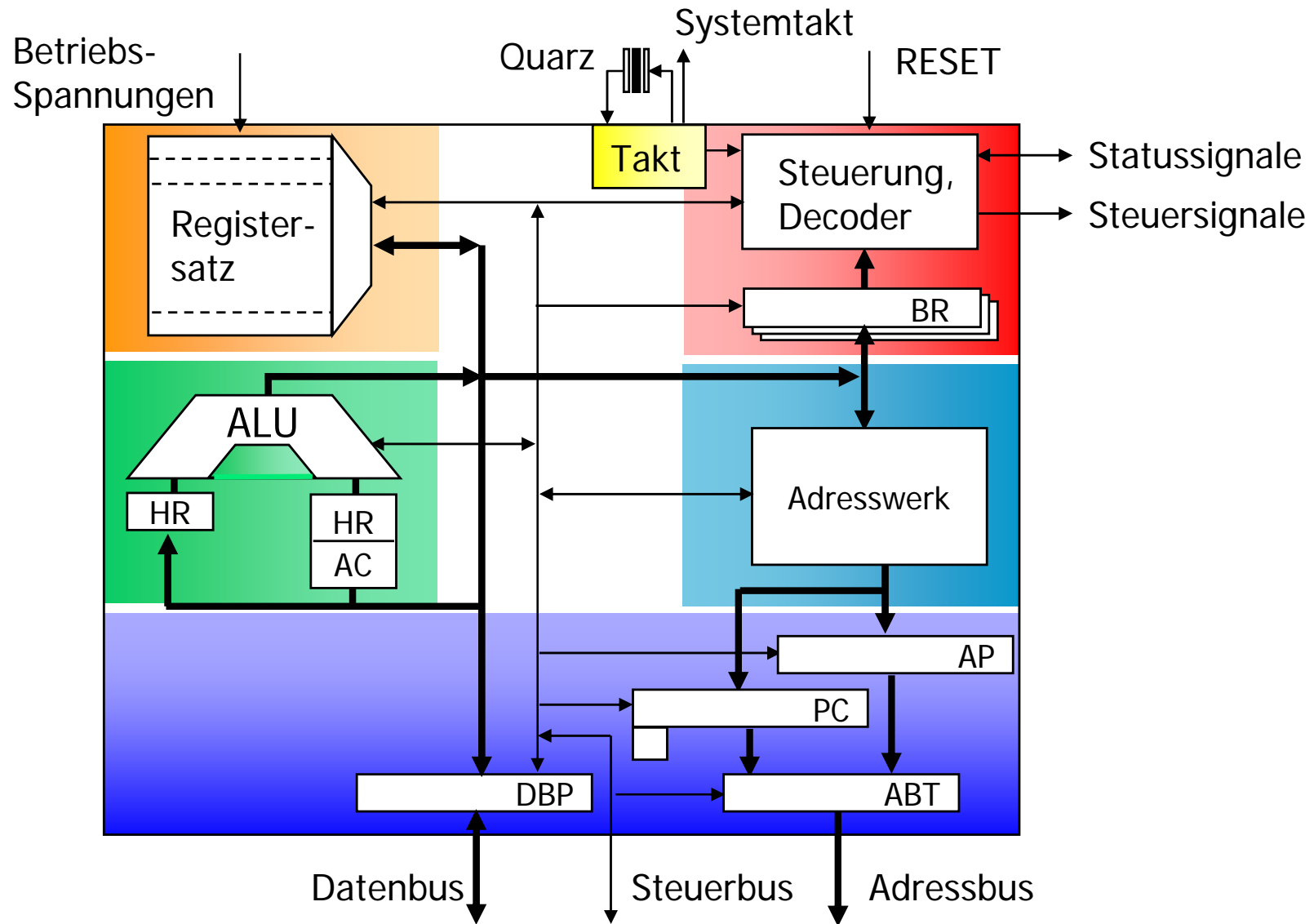
# Aufbau eines einfachen $\mu\text{P}$

---

- ❑ Steuerwerk
- ❑ Rechenwerk
- ❑ Registersatz
- ❑ **Adresswerk**
- ❑ Systembusschnittstelle
- ❑ Interne Busse

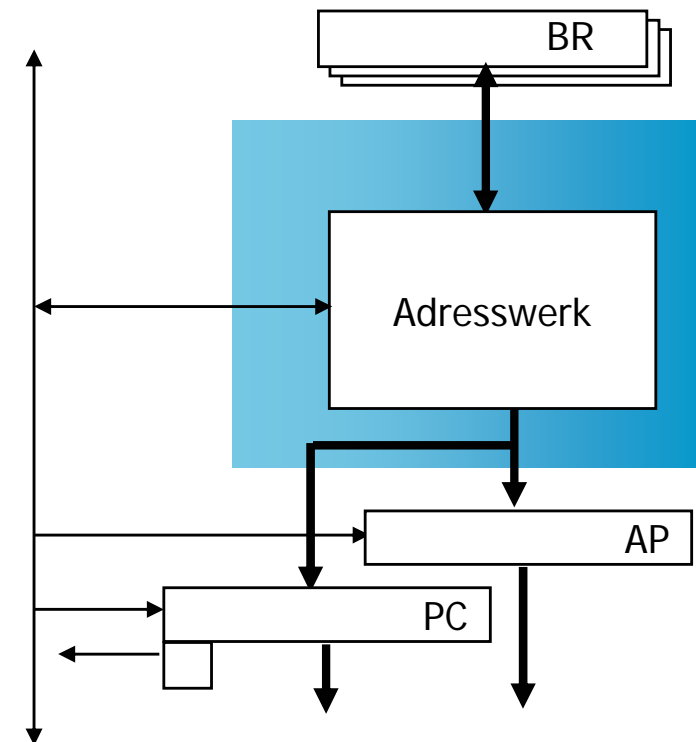


# Aufbau eines einfachen $\mu P$

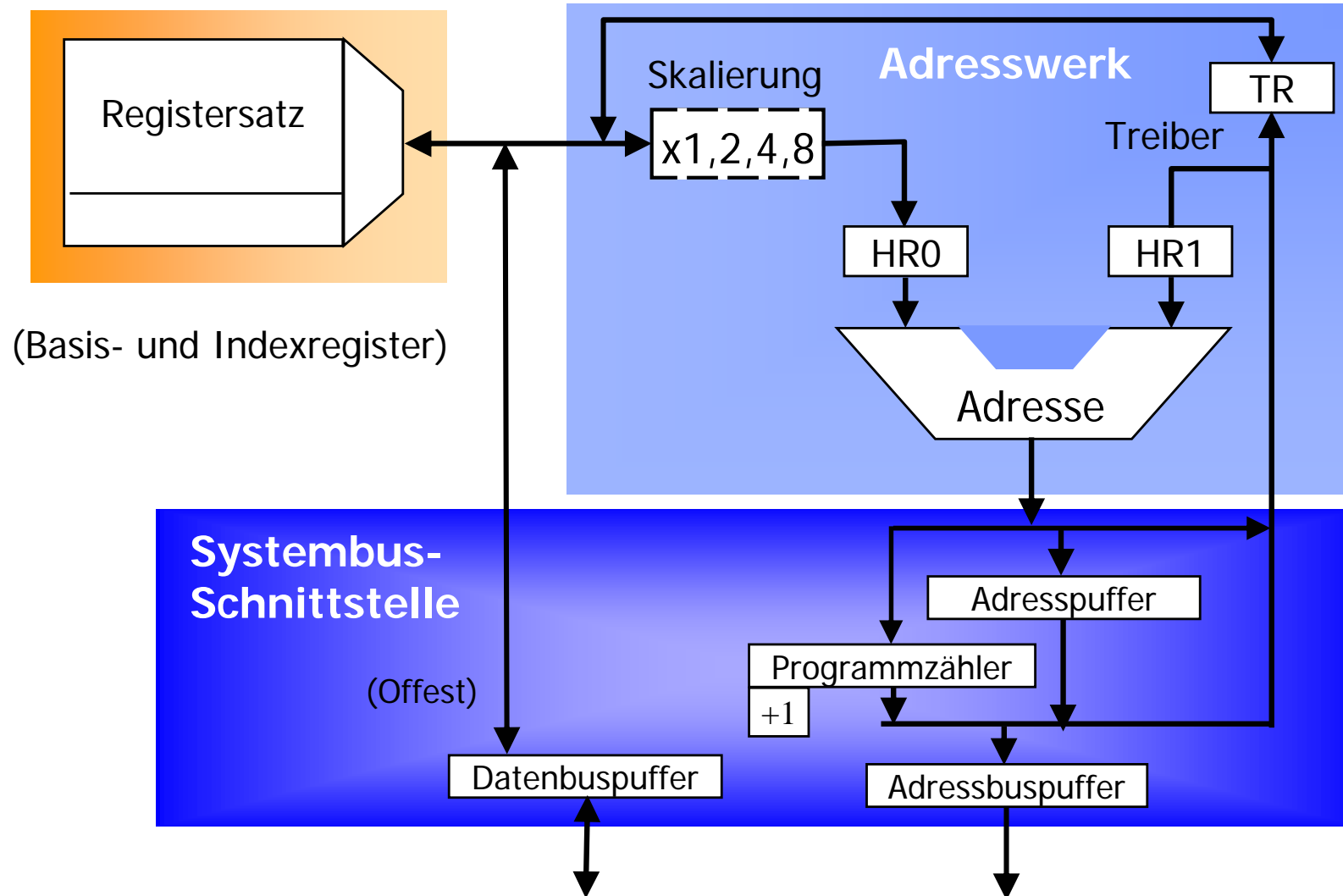


## 3.2.4 Adresswerk

- Berechnet nach den Vorschriften des Steuerwerks die Adresse eines Befehls oder eines Operanden
- Früher:  
Häufig Bestandteil des Rechenwerks
- Heute:  
Sehr komplex (viele verschiedene komplexe Adressierungsarten) und deshalb eigenständig



# Aufbau eines einfachen Adresswerks



# Funktionsweise

---

## Adress-Addierer mit 2 Eingängen

### Eingang A:

- Registersatz (Adresse aus Basis- oder Indexregister)
- Datenbuspuffer (absolute Adresse im Befehl oder absolute Adressdistanz zu einem Basisregister)

### Eingang B:

- Befehlszähler (Adresse des aktuellen Befehls)
- Adresspuffer (Adresse des aktuellen Operanden)

Der Ausgang kann über den Adresspuffer oder den Programmzähler auf Eingang **B** rückgekoppelt werden

➔ Adresspuffer und Programmzähler als Akkumulatoren



# Das Adresswerk

---

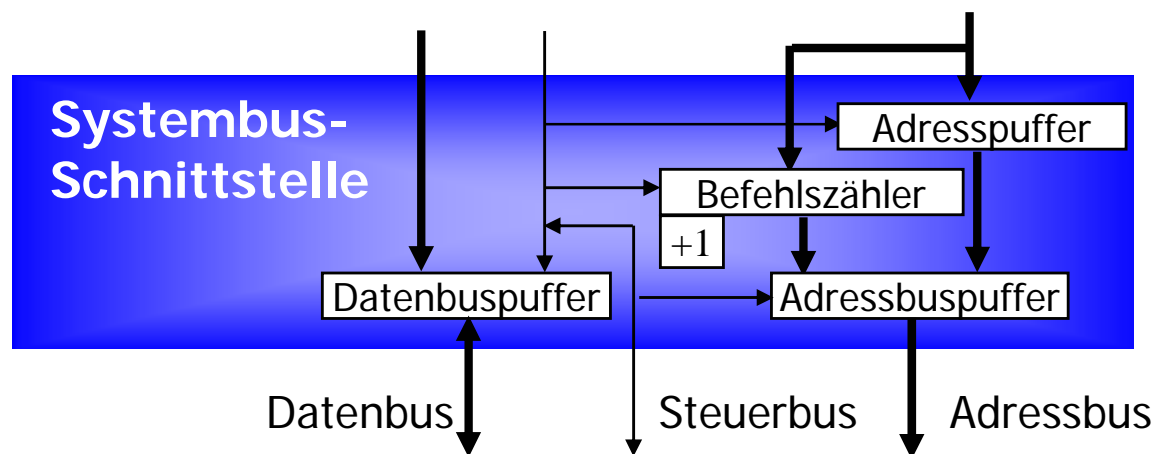
**Heute:** Eigene Adresswerke.

- Adressberechnung findet parallel (im Pipelining-Verfahren) zu den Aktivitäten des Rechenwerks statt!
- Führt neben einfacher Adressrechnung auch die Adressrechnung zur virtuellen Speicherverwaltung durch
- Früher oft seperater Baustein, heute in den Prozessor integriert.
- Komplexe Adresswerke zur virtuellen Speicherverwaltung (später!)



# 3.2.5 Systembus Schnittstelle

- ❑ Enthält diverse Zwischenspeicher-Register (Puffer) zur kurzfristigen Aufbewahrung von Adressen und Daten, z. B.
  - **Befehlszähler:** Adresse des nächsten Befehls (Instruction Pointer, Program Counter)
  - **Adresspuffer:** Adresse des gewünschten Operanden im Hauptspeicher
  - **Datenbuspuffer:** Wert des gerade bearbeiteten Operanden
- ❑ Enthält Aus- und Eingangstreiber (Tristate-Treiber)



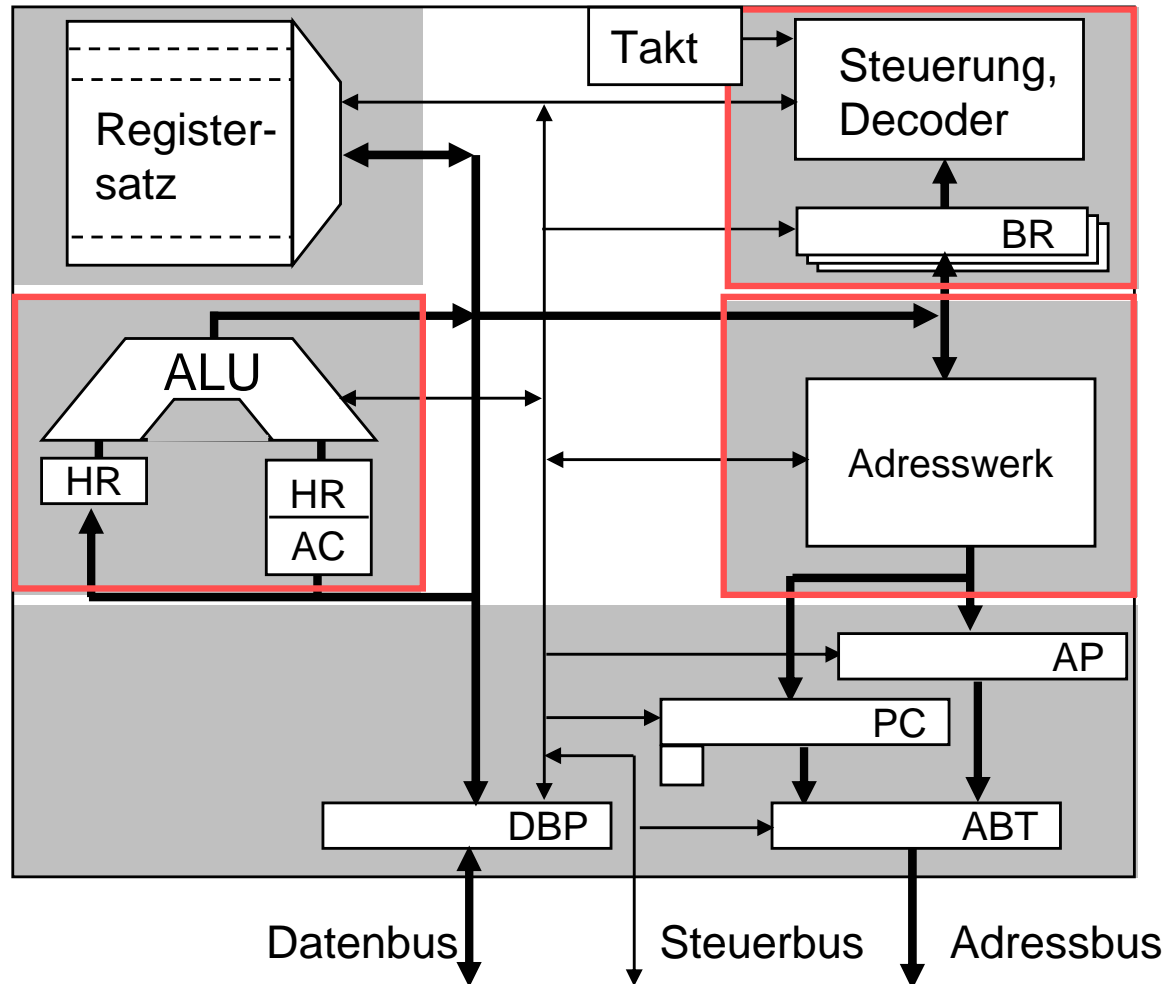
## 3.2.6 Interne Busse

---

- Dienen zur Verbindung der Prozessor-Komponenten untereinander
- Keine eindeutige Unterscheidung Daten- und Adressbus, da einmal Operanden und einmal Adressen übertragen werden



# Ablauf der Befehlsabarbeitung



**Pipelining**



# Kapitel 4

---

## **Befehlssatzarchitektur** *(Instruction Set Architektur ISA)* **Die Hardware-Software-Schnittstelle**

- Datentypen, Datenformate
- Befehlsformat, Befehlssatz
- Adressierungsarten
- Diskussion: RISC & CISC; Fallstudien (MIPS)



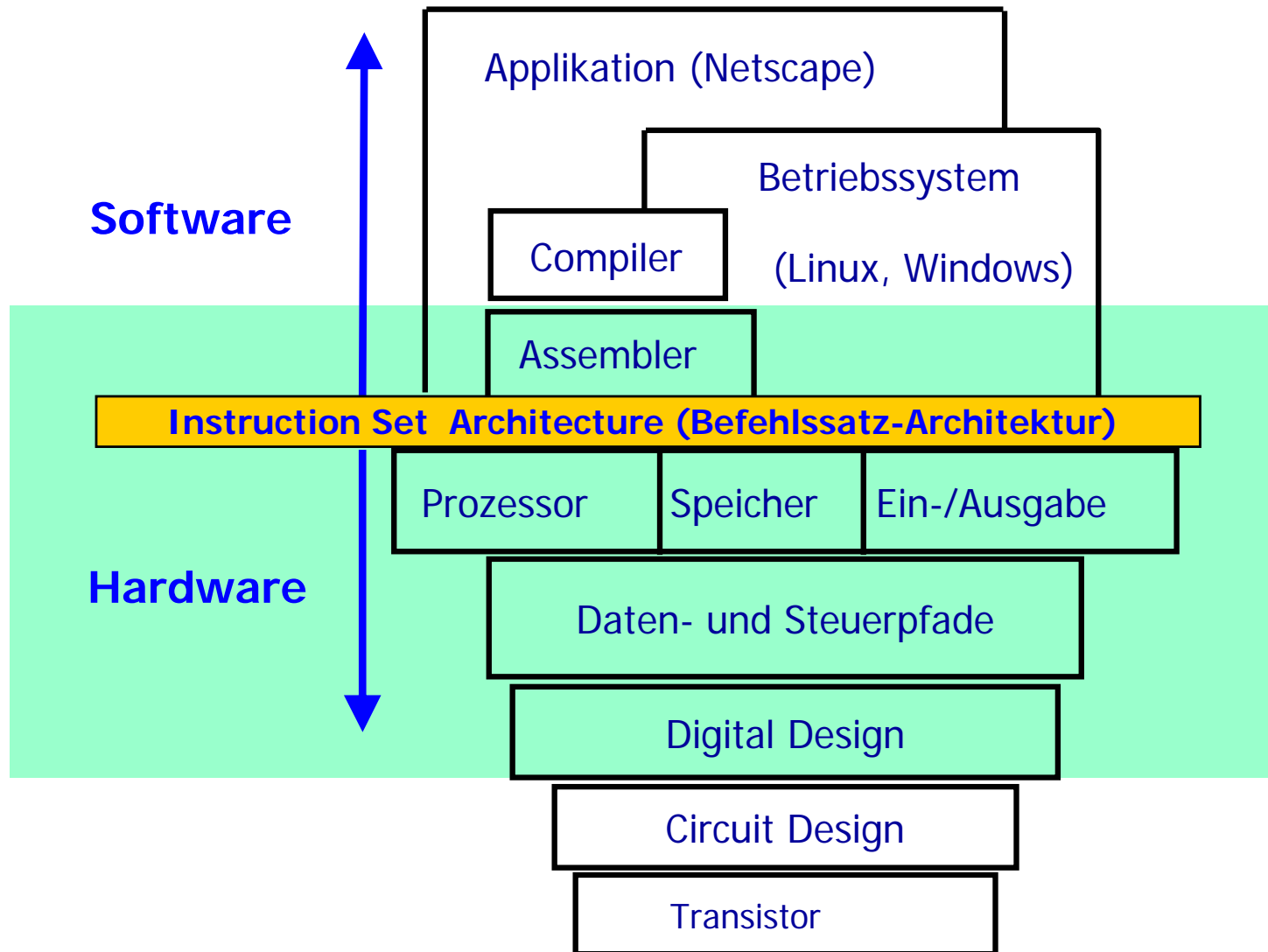
# Architektur (ISA)

---

- **Der Begriff „Architektur“ (Instruction Set Architecture, ISA)**
  - Beschreibung der Attribute und des funktionalen Verhaltens eines Prozessors
  - Äußeres Erscheinungsbild
    - Sichtweise des Maschinenprogrammierers
  - Spezifikation einer Architektur
    - Befehlssatz
    - Befehlsformat
    - Datentypen, und Datenformate
    - Adressierungsarten
    - Register-, Speichermodell
    - Unterbrechungssystem



# Instruction Set Architecture (Befehlssatz-Architektur)



# Programmierersicht eines Prozessors

---

- Wie werden Daten repräsentiert?
- Wo werden die Daten gespeichert?
- Welche Operationen können auf den Daten ausgeführt werden?
- Wie werden die Befehle codiert?
- Wie wird auf die Daten zugegriffen?

→ ISA



# Architektur (ISA)

---

- **Klassen von Architekturen, Ausführungsmodelle**
  - Für eine zweistellige Operation, d. h. bei einer Operation, bei der die zwei Operanden miteinander verknüpft werden, sind folgende Angaben notwendig
    - Art der Operation
    - Adresse des ersten Operanden
    - Adresse des zweiten Operanden
    - Adresse des Resultats



# Ausführungsmodelle

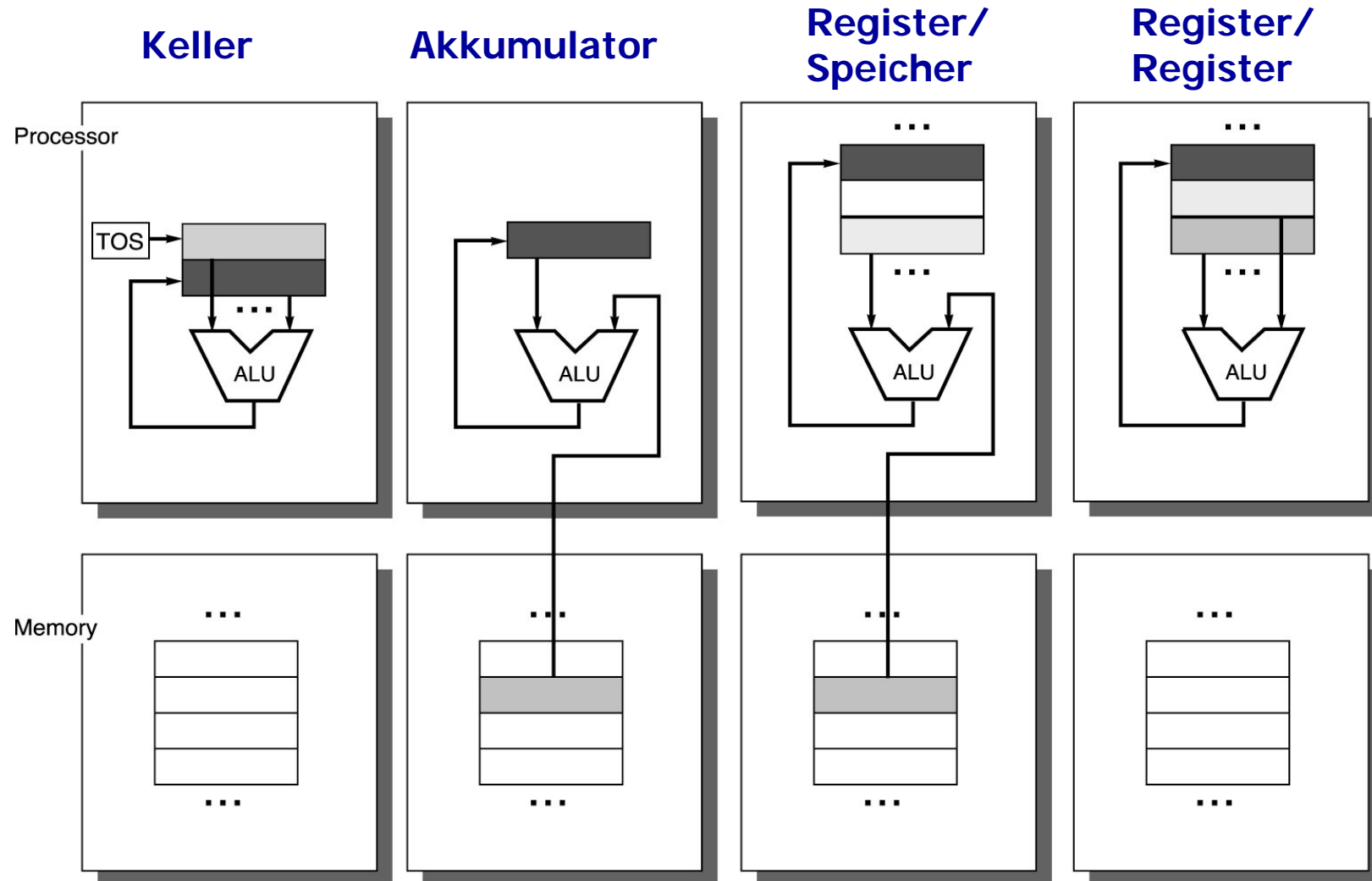
---

## □ Fragen

- Wo stehen die Operanden bzw. das Ergebnis?
  - Hauptspeicher, Allzweckregister, Akkumulator, Keller
- Explizite oder implizite Adressierung
  - Explizite Angabe der Adresse oder implizit im Opcode enthalten
- Überdeckte Adressierung
  - Fallen zwei Adressen (Quelle, Ziel) zusammen
- Variables oder festen Befehlsformat



# Ausführungsmodelle



# Ausführungsmodelle

## □ Allzweckregister-Architekturen

### ➤ Register-Register-Modell

- Alle Operanden und das Ergebnis stehen in Allzweckregistern

```
add R1,R2,R3          R1 ← R2+R3
```

- Load/Store-Architektur

- Ausgezeichnete Befehle holen die Operanden aus dem Hauptspeicher und schreiben die Inhalte von Registern in den Speicher

```
load R2,A             R2 ← mem[A]
load R3,B             R3 ← mem[B]
add R1,R2,R3          R1 ← R2+R3
store C,R1            mem[C] ← R1
```

- Dreiadressformat
- Beispiele
  - Alpha, ARM, MIPS, PowerPC, SPARC, Trimedia TM5200



# Ausführungsmodelle

---

## □ Allzweckregister-Architekturen

### ➤ Register-Register-Modell

- Vorteil:

- Einfaches und festes Befehlsformat
- Einfaches Code-Generierungsmodell
- Etwa gleiche Ausführungszeit der Befehle

- Nachteil:

- Höhere Anzahl von Befehlen im Vergleich zu Architekturen mit Speicherreferenzen
- Mehr Instruktionen und geringere Befehlsdichte führen zu längeren Programmen

- Beispiele

- Alpha, ARM, MIPS, PowerPC, SPARC, Trimedia TM5200



# Ausführungsmodelle

## □ Allzweckregister-Architekturen

### ➤ Register-Speicher-Modell

- Ein Operand steht im Speicher, der zweite Operand steht im Register; das Ergebnis steht entweder im Speicher oder im Register

– **add A,R1**            **mem[A] ← mem[A]+R1**

– **add R1,A**            **R1 ← R1+mem[A]**

- explizite Adressierung mit/ohne Überdeckung
- Zweiadressformat
  - Befehlsformat sieht zwei explizite Adressangaben vor
    - » Registerspeicher (prozessorintern)
    - » Hauptspeicher (prozessorextern) beziehen
  - Überdeckung einer Quelladresse mit einer Zieladresse
- Beispiele: IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x



# Ausführungsmodelle

---

## □ Allzweckregister-Architekturen

### ➤ Register-Speicher-Modell

- Vorteil:
  - Auf die Daten kann ohne vorherige Lade-Operation zugegriffen werden
  - Kodierung im Befehlsformat führt zu höherer Code-Dichte
- Nachteile:
  - Operanden können nicht gleich behandelt werden, wenn eine Überdeckung vorliegt
  - Anzahl der Taktzyklen pro Instruktion variiert in Abhängigkeit der Adressrechnung
- Beispiele:
  - IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x



# Ausführungsmodelle

## □ Akkumulator-Register-Architektur

- Ein ausgezeichnetes Register (Akkumulator)
  - Der Akkumulator wird bei einer zweistelligen Operation als Quelle einer der beiden Operanden angesprochen, gleichzeitig dient der Akkumulator als Ziel für das Resultat

– **add A**                      **acc ← acc + mem[A]**

– **addx A**                     **acc ← acc + mem[A+x]**

– **add r1**                     **acc ← acc + r1**

- Implizite und überdeckte Adressierung
- Spezielle Befehle ermöglichen den Transport von Operanden
- Einadressformat





# Ausführungsmodelle

---

## □ Speicher-Speicher-Architektur

- Die beiden Operanden einer zweistelligen Operation sowie das Ergebnis stehen im Speicher

– `add A, B, C`      `mem[A] ← mem[B] + mem[C]`

- Explizite Adressierung
- Dreiadressformat
- Nachteil:
  - Speicherzugriffe führen zu Speicherengpass
- Beispiele
  - DEC VAX



# Programm $C=A+B$ ; $D=C-B$ ; in den vier Befehlsformatsarten codiert

Register-Register	Register-Speicher	Akkumulator	Keller
<pre>load Reg1,A load Reg2,B Add Reg3,Reg1,Reg2 store C,Reg3 load Reg1,C load Reg2,B sub Reg3,Reg1,Reg2 store D,Reg3</pre>	<pre>load Reg1,A add Reg1,B store C,Reg1 load Reg1,C sub Reg1,B store D,Reg1</pre>	<pre>load A add B store C load C sub B store D</pre>	<pre>push B push A add pop C push B push C sub pop D</pre>



# Architektur (ISA)

---

## □ Datentypen

- Spezifizieren die Wertebereiche, die Konstanten, Ausdrücke, Variablen oder Funktionen in Programmen annehmen können
- Sind die unterschiedlichen Interpretationsarten, die vom Mikroprozessor direkt unterstützt werden (Hardware-sicht)
- Sind charakterisiert durch ein Datenformat und durch die inhaltliche Interpretation
- Die Interpretation wird durch die einzelnen arithmetischen und logischen Befehle vorgegeben
- Die für ein Datentyp bestimmten Operationen interpretieren die Operanden in gleicher Weise



# Architektur (ISA)

---

## □ Datentypen

- Alternative: Datentyparchitektur
  - Daten führen Typinformation mit sich
  - Typinformation wird durch die Hardware interpretiert und wählt Operation entsprechend aus
  - Keine Bedeutung!



# Architektur (ISA)

---

## □ Datentypen

- Datentypen, die nicht von der Hardware unterstützt werden, müssen durch ein geeignetes Programm auf elementare Datentypen zurückgeführt und in mehreren Schritten berechnet werden
- Prozessoren unterstützen Datentypen, die sie in ihrem Rechenwerk nicht verarbeiten können (z. B. Gleitkommazahlen)
  - Mit speziellen Befehlen können Operanden solcher Datentypen aus dem Hauptspeicher in spezielle Register geladen bzw. von dort wieder in den Speicher zurück geschrieben werden
    - Vereinfachung der Software-Emulation



# Architektur (ISA)

---

## □ Datenformate

### ➤ Standardformate

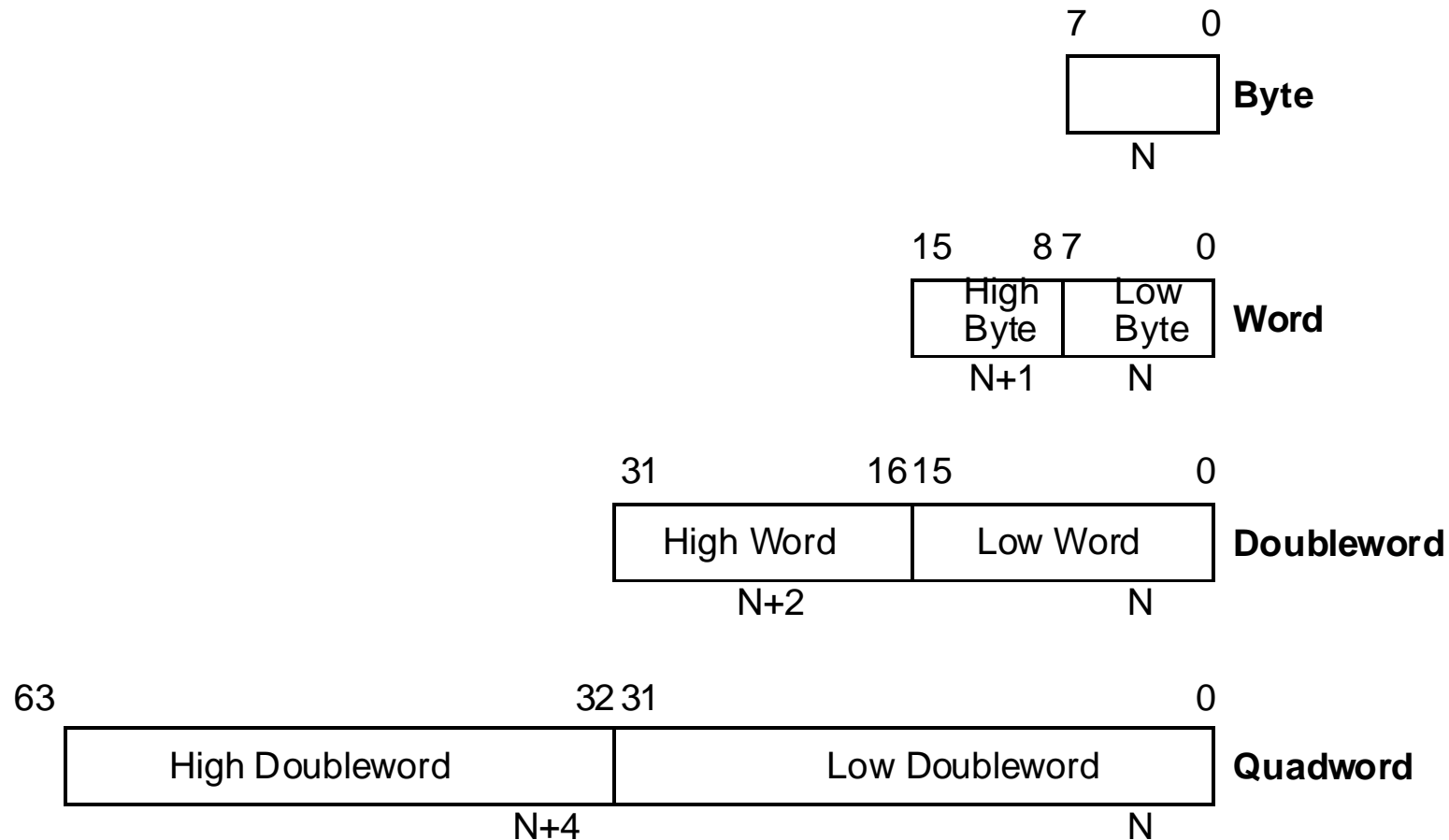
- Byte: 8 Bit
- Halbwort: 16 Bit
- Wort: 32 Bit
- Doppelwort: 64 Bit
  - Der Begriff “Wort” wird von den Herstellern unterschiedlich verwendet. Er orientiert sich z. B. an der Verarbeitungsbreite von 32-Bit Prozessoren. Bei Intel wird der Begriff “Wort” für 16-Bit benutzt.
- Beispiel: MIPS Architektur



# Architektur (ISA)

## □ Datenformate: Fallstudie IA-32

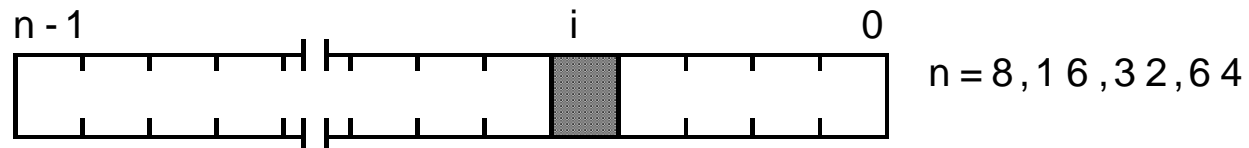
### ➤ Fundamental Data Types



# Datentypen

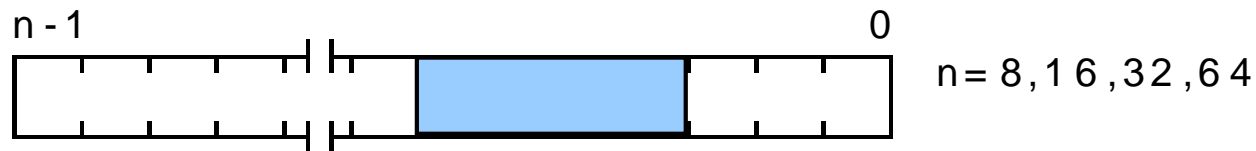
## □ Zustandsgröße Bit:

- ein Bit in einem der Standardformate



## □ Bitvektor:

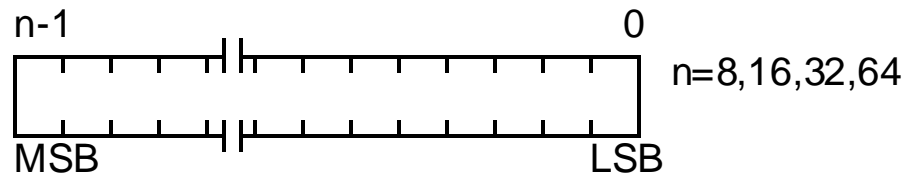
- Zusammenfassung von Zustandsgrößen (Standardformate)



# Datentypen

## □ Vorzeichenlose Dualzahl

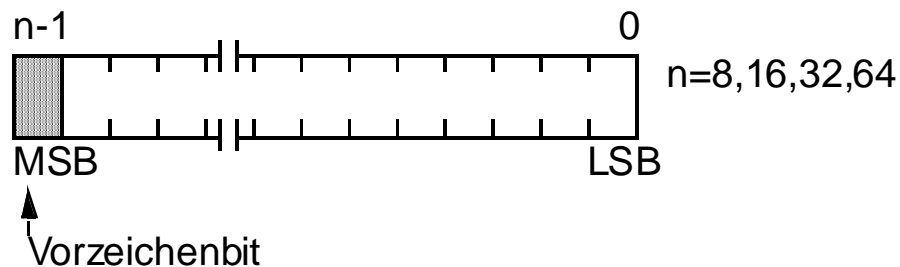
- Standardformate  $n=8,16,32$  oder  $64$ ; Wertebereich:  $0$  bis  $2^n-1$



LSB: niederwertigstes Bit (least significant Bit)  
MSB: höchstwertiges Bit (most significant Bit)

## □ 2'er Komplement (signed Integer):

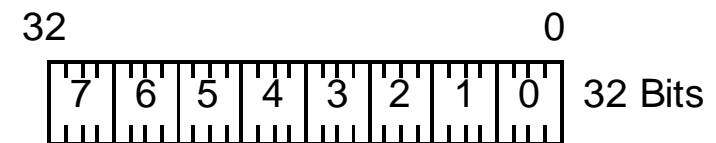
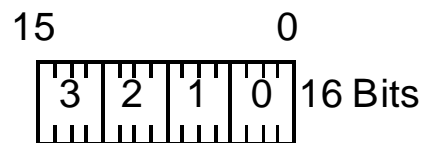
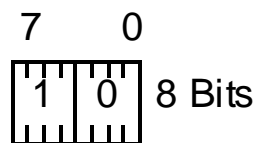
- Standardformate  $n=8,16,32$  oder  $64$ ; Wertebereich:  $-2^{n-1}$  bis  $+2^{n-1}-1$



# Datentypen

## □ BCD-Ziffern in gepackter Form (packed binary digitals)

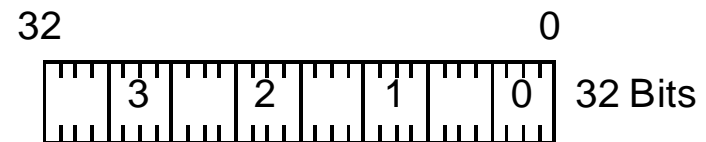
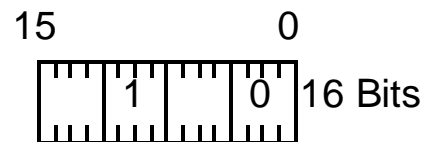
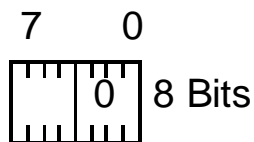
- 2, 4 oder 8 Halbbytes werden in den Standardformaten zusammen gefasst;
- ziffernweises Codieren der Dezimalzahlen im 8-4-2-1 Dualcode, d. h. das Codewort umfasst 4 Bits mit den Gewichten 8,4,2 und 1
- Exakte Ergebnisse bezüglich Dezimalzahlen
  - $0,10_{10}$  in Binärdarstellung:  $0,0001100110011..._2$



# Datentypen

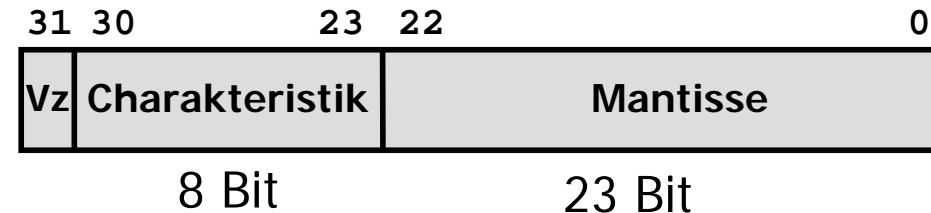
## □ BCD-Ziffern in ungepackter Form

- 1, 2, oder 4 Bytes werden in den Standardformaten zusammen gefasst;
- Ein Byte enthält neben der binärcodierten Dualzahl zusätzliche Bits im höherwertigen Halbbyte
- Exakte Ergebnisse bezüglich Dezimalzahlen

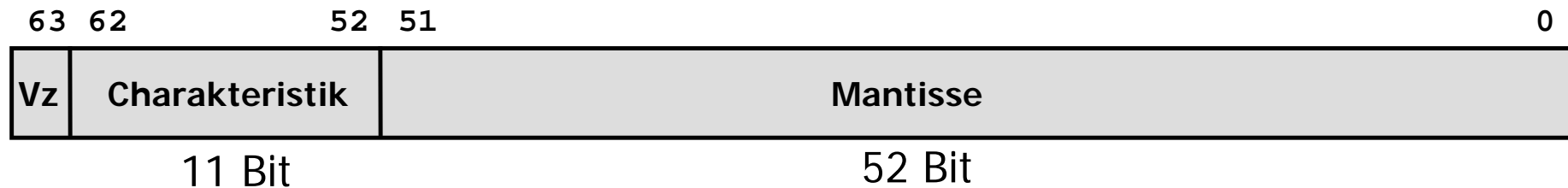


# Datentypen: Gleitkommazahlen

Einfache Genauigkeit:



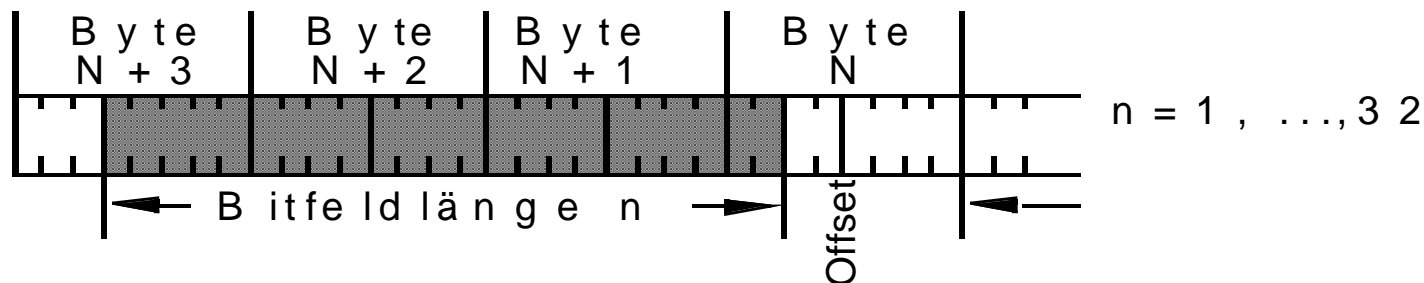
Doppelte Genauigkeit:



# Datentypen

## □ Bitfeld

- Darstellung und Verarbeitung von Bitvektoren, vorzeichenlosen Dualzahlen und 2-Komplementzahlen;
- variable Bitanzahl: bis zu 32 Bit;
- wird im Speicher durch eine Byte-Adresse und einen darauf Bezug nehmenden Bitfeld-Offset und die Angabe der Bitfeldlänge angesprochen



# Datentypen

---

## □ String

- Verarbeitung von aufeinander folgenden gespeicherten Bytes, Halbwörter oder Wörter
- *Byte-Strings*: bestehend aus ASCII-Zeichen;
- *Byte-, Halbwort- oder Wort-Strings*: mit Elementen in vorzeichenloser Dualzahl- oder 2-Komplementdarstellung



# Datentypen

---

## □ Fallstudie: IA-32

- Ordinal bzw. unsigned integers (vorzeichenlose Dualzahl) in den Standardformaten;
- Integer (2-Komplementzahl) in den Standardformaten;
- Packed BCDs (binär codierte Dezimalzahl in gepackter Darstellung);
- Unpacked BCDs (binär codierte Dezimalzahl in ungepackter Darstellung);
- Near Pointer (effektive Adresse innerhalb eines Segments);
- Far Pointer (logische Adresse, die sich aus dem 16-Bit breiten Segmentselektor und dem 32-Bit breitem Offset zusammensetzt);
- Bit Fields (Bitfelder)
- Strings (Folge von Bits, Bytes, Wörtern oder Doppelwörtern, wobei ein Bit-String an jeder Position in einem Byte beginnen kann und bis zu  $2^{32}-1$  Bits enthalten kann und Byte-String Bytes, Words oder Doublewords enthalten kann in einem Bereich von 0 bis  $2^{32}-1$  Bytes )
- Floating-Point Data Types (Gleitkomma-, Integer und BCD-Zahlen)



# Datentypen

---

## □ Fallstudie: MIPS64 Architektur

- MIPS64 Operationen arbeiten auf 64 Bit Integer und 32- oder 64 Bit Gleitkommandaten
- Byte-, Half Word- und Word-Daten werden in GPRS geladen mit Null- oder Vorzeichenerweiterung

