Karlsruher Institut für Technologie

Rechnerorganisation

Prof. Dr. Wolfgang Karl

Vorlesung im Wintersemester 2025/2026 – Foliensatz: RO25-FS04

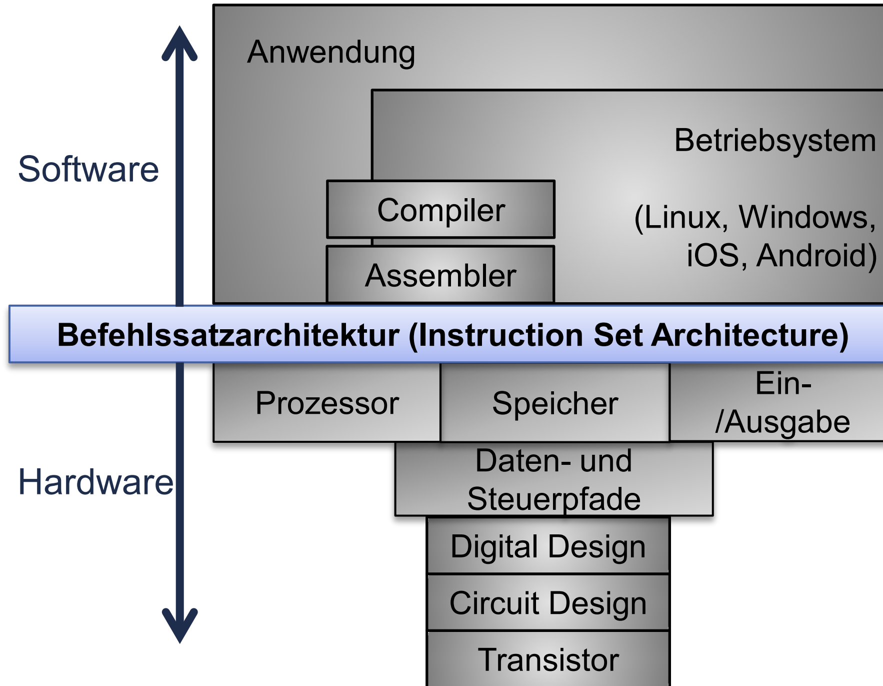
Kapitel 4
Befehlssatzarchitektur

- Grundlagen
- Ausführungsmodelle
- Datentypen, Datenformate
- Adressierungsarten
- Befehlsformat
- RISC-V Assembler-Programmierung
- Fallstudien (IA-32, RISC-V)
- Diskussion: RISC & CISC

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

Abstraktionsebenen eines Computersystems



Befehlssatzarchitektur

- **Definition Befehlssatzarchitektur (Instruction Set Architecture, ISA)**
  - Äußeres Erscheinungsbild
    - Sichtweise des Maschinenprogrammierers, Compiler-Bauers, Betriebssystementwicklers
  - Schnittstelle zwischen Hardware und Software
  - Abstraktion der Hardware
  - Beschreibung des funktionalen Verhaltens eines Prozessors

© Prof. Dr. Wolfgang Karl

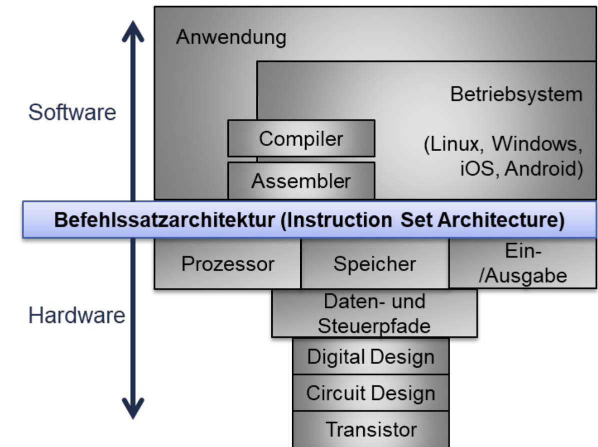
Befehlssatzarchitektur

- **Spezifikation einer Befehlssatzarchitektur**
  - Befehlssatz
  - Befehlsformat
  - Datentypen, und Datenformate
  - Adressierungsarten
  - Register-, Speichermodell
  - Unterbrechungssystem

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Rechner- / Prozessorfamilie**
  - Umfasst Computer bzw. Prozessoren mit unterschiedlichen Organisationformen und Technologiegenerationen, die eine Befehlssatzarchitektur unterstützen;
  - Aufwärtskompatibilität
    - Programme, die für Rechner einer früheren Generation entwickelt wurden, sind auch auf Rechnern nachfolgender Generationen ausführbar.



© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Familienkonzept**
  - Erstmals eingeführt bei IBM 1964
  - Großrechnerfamilie IBM /360
    - Eine Befehlssatzarchitektur, die heute noch von den IBM Mainframes unterstützt wird
    - Konzept der Mikroprogrammierung
  - Modellreihe (20, 30, 40, …, 91…)
    - Ein Programm läuft auf allen Modellen
    - Unterschiedlich leistungsfähig
    - Unterschiedliche Preise
    - Unterschiedliche Anwendungsbereiche

IBM System/360 Model 20

IBM System/360 Model 50

Quelle:Bundesarchiv_B_145_Bild-F038812-0014,_Wolfsburg,_VW_Autowerk.jpg

Befehlssatzarchitektur

- **Familienkonzept: Prozessorfamilien**
  - Beispiele:
    - IA-32 Befehlssatzarchitektur
      - Intel x86 Prozessorfamilie
      - AMD x86 Prozessorfamilie
    - Intel® 64, AMD64 Erweiterung der IA-32 auf 64 Bit
      - Intel Core-i, Xeon, …
      - AMD Athlon, Opteron
      - AMD Ryzen, …

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Entwurf eines Befehlssatzes**
  - **Befehle**
    - Art der Befehle
      - Mächtigkeit, einfache oder komplexe Befehle
    - Anzahl der Befehle
  - **Datentypen**
    - Verschiedene Typen von Daten, auf denen Operationen ausgeführt werden können
  - **Register- / Speicheroperanden**
    - Ort der zu verarbeitenden Operanden bzw. des Ergebnisses
  - **Adressierung**
    - Spezifikation der Adressen der Operanden
    - Adressierungsarten

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Entwurf eines Befehlssatzes: Hinführung**
  - **C-Anweisung**

```
…
int summe=a+b;
…
```

  - **Wie sieht der Maschinenbefehl (Assemblerbefehl) aus?**
  - **So?**

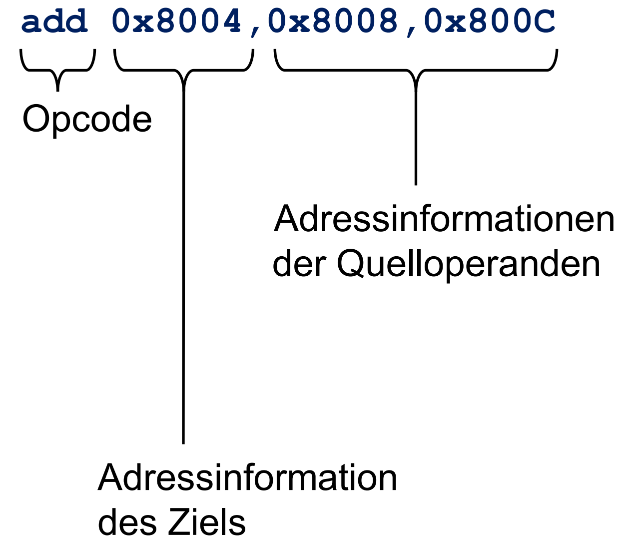
```
…
add 0x8004,0x8008,0x800C
…
```

	Hauptspeicher	
Variablen:	Adresse	Inhalt
summe	0x8004	00000111
a	0x8008	00000011
b	0x800C	00000100

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Entwurf eines Befehlssatzes: Hinführung**
  - **Bestandteile eines Maschinenbefehls**
    - **Operationscode (Opcode)**
      - Spezifiziert die auszuführende Operation
    - **Spezifikation der Adressinformationen der Quelloperanden**
      - **Quelloperanden**
        - Objekte, die durch die Operation verknüpft bzw. bearbeitet werden
    - **Spezifikation der Adressinformationen des Ergebnisoperanden (Ziel)**
      - **Ergebnisoperand:**
        - Das durch die Operation produzierte Ergebnis



© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Entwurf eines Befehlssatzes: Fragen**
  - Wo stehen die Operanden für die Operation und wo wird das Ergebnis (Ziel) abgelegt?
    - Hauptspeicher, Register, Akkumulator, Keller (Stack)
  - Wie werden die Operanden und das Ergebnis adressiert?
    - Explizite Angabe der Adressinformationen der Quelloperanden und des Ziels im Befehl
    - Implizite Adressierung der Operanden
    - Überdeckte Adressierung
      - Eine Adresse für Quelloperanden und Ziel

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Entwurf eines Befehlssatzes: Antwort**
  - **Ausführungsmodelle**
    - Bestimmen, wo die Operanden, die ein Befehl verarbeitet, stehen können;
  - **Klassen von Befehlssatzarchitekturen**
    - Speicher-Speicher-Architektur
    - Register-Register-Architektur
    - Register-Speicher-Architektur
    - Akkumulator-Architektur
    - Keller-Architektur

© Prof. Dr. Wolfgang Karl

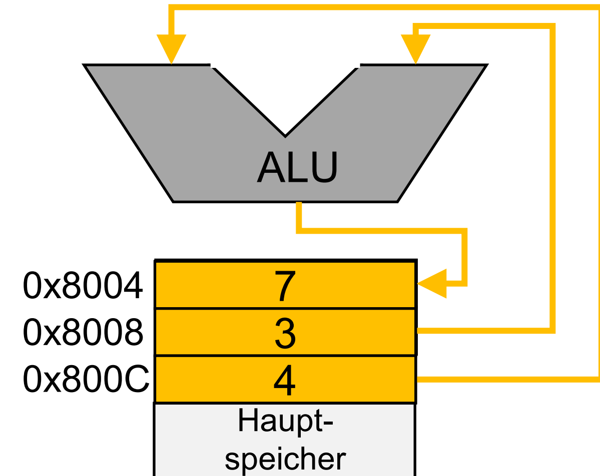
Ausführungsmodelle

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

```
add S,A,B ; mem[S]← mem[A]+mem[B]
```

- **Explizite Adressierung**
  - Speicheradressen sind im Befehl spezifiziert.
- **Dreiadressformat**

- Nachteil:
  - Speicherzugriffe können zu Engpässen führen.
  - Unterschiedlich lange Zugriffszeiten;

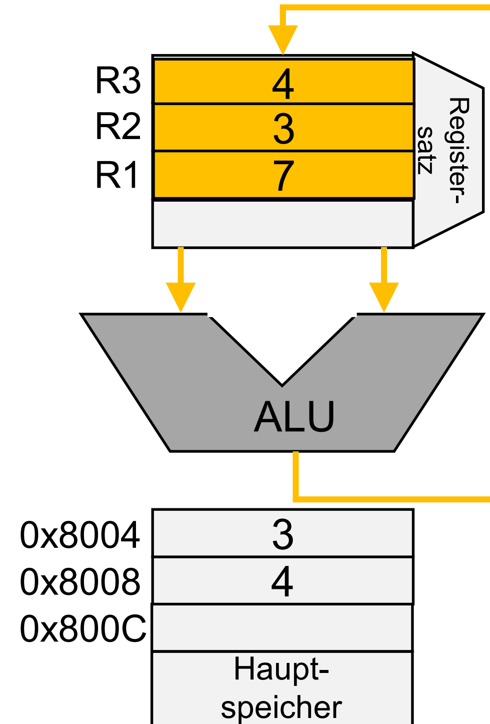


© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

- **Register-Register-Architektur**
  - Die beiden Operanden und das Ergebnis stehen in Allzweckregistern

```
add R1,R2,R3  ; R1← R2+R3
```



© Prof. Dr. Wolfgang Karl

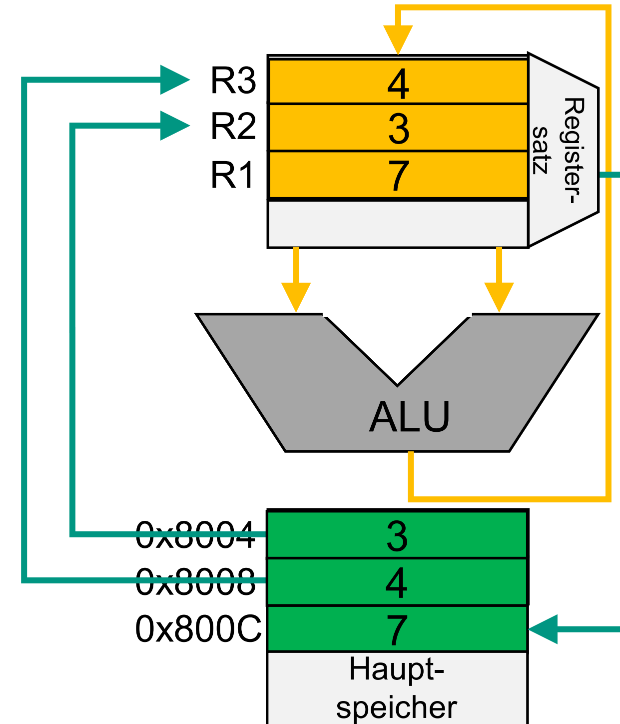
Ausführungsmodelle

- **Register-Register-Architektur**
  - Die beiden Operanden und das Ergebnis stehen in Allzweckregistern

```
add R1,R2,R3  ; R1← R2+R3
```

- **Load/Store-Architektur**
  - Dedizierte Befehle holen die Operanden aus dem Hauptspeicher und schreiben die Inhalte von Registern in den Speicher.

```
ld R2,A         ; R2 ← mem[A]
ld R3,B         ; R3 ← mem[B]
add R1,R2,R3    ; R1 ← R2+R3
st S,R1         ; mem[S] ← R1
```



© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

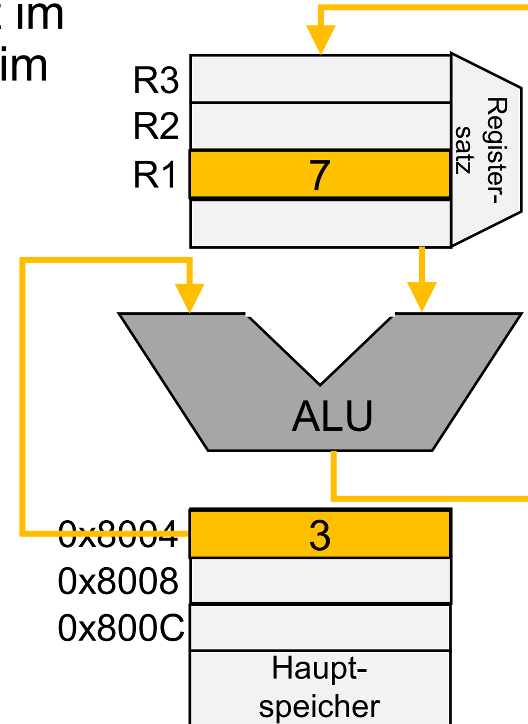
- **Register-Register-Architektur**
  - Vorteil:
    - Einfaches Code-Generierungsmodell!
    - Etwa gleich lange Ausführungszeiten der Befehle;
      - Ausnahme: Speicherzugriffsbefehle (Lade- und Speicherbefehle), Verzweigungen
  - Nachteil:
    - Höhere Anzahl von Befehlen im Vergleich zu Architekturen mit Speicherreferenzen;
    - Mehr Instruktionen und geringere Befehlsdichte führen zu längeren Programmen;
  - Beispiele
    - ARMv7, MIPS, PowerPC, SPARC, RISC-V

© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

- **Register-Speicher-Architektur**
  - Ein Operand steht im Speicher, der zweite Operand steht im Register. Das Ergebnis steht entweder im Speicher oder im Register.

```
add R1,A      ; R1 ← R1 + mem[A]
```

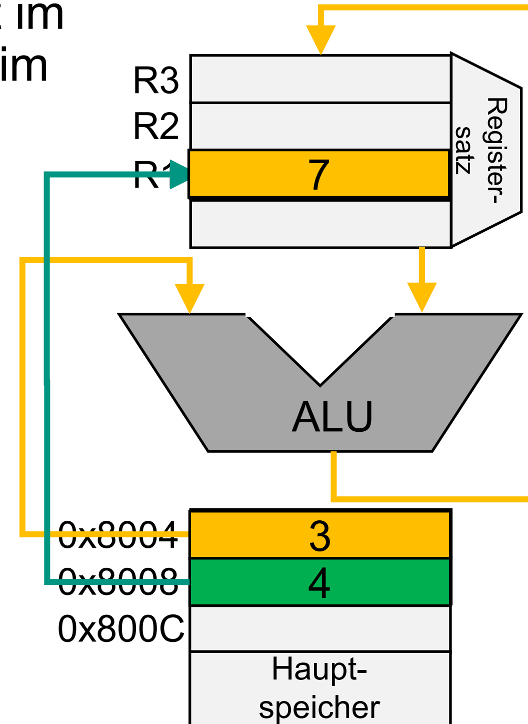


© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

- **Register-Speicher-Architektur**
  - Ein Operand steht im Speicher, der zweite Operand steht im Register. Das Ergebnis steht entweder im Speicher oder im Register.

```
ld R1, B      ; R1 ← mem[B]
add R1,A      ; R1 ← R1+mem[A]
```

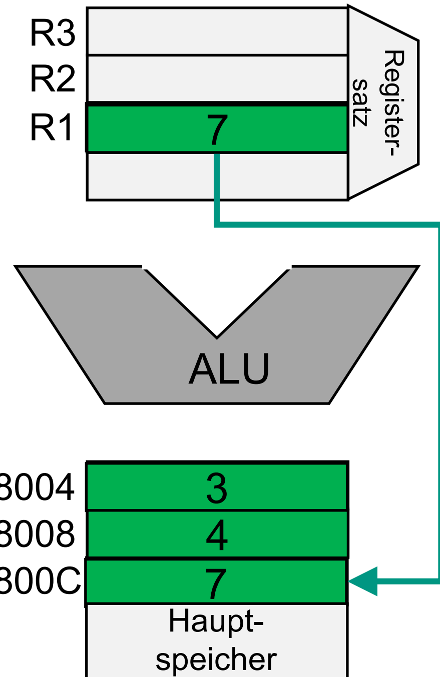


© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

- **Register-Speicher-Architektur**
  - Ein Operand steht im Speicher, der zweite Operand steht im Register. Das Ergebnis steht entweder im Speicher oder im Register.

```
ld R1,B        ; R1 ← mem[B]
add R1,A       ; R1 ← R1+mem[A]
st S,R1        ; mem[S] ← R1
```



- **Zweiadressformat**
- Befehlsformat sieht zwei **explizite Adressangaben** vor:
  - Registerspeicher (prozessorintern)
  - Hauptspeicher (prozessorextern)
- **Überdeckung** einer Quelladresse mit einer Zieladresse

© Prof. Dr. Wolfgang Karl

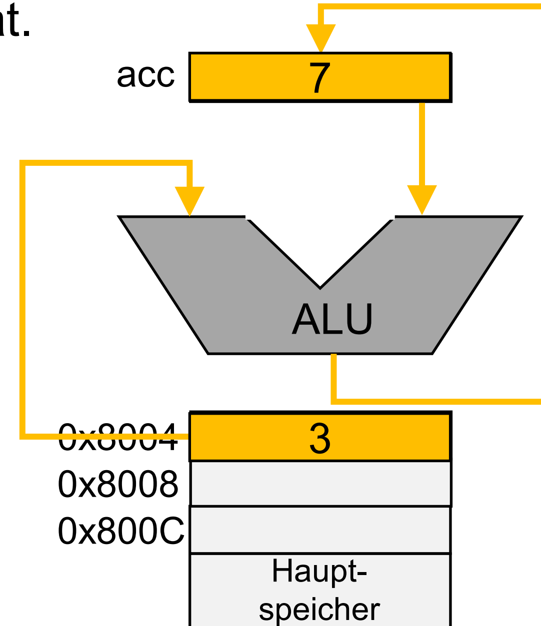
Ausführungsmodelle

- **Register-Speicher-Architektur**
  - Vorteile:
    - Auf ein Datum 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, Intel 80x86, Motorola 68000, TI TMS320C54x

Ausführungsmodelle

- **Akkumulator-Architektur**
  - 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
```

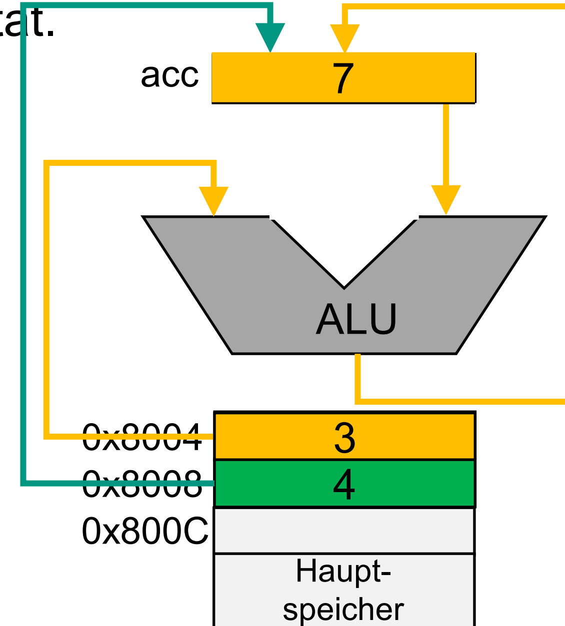


- **Einadressformat**
  - Befehlsformat sieht eine **explizite Adressangaben** vor
    - Hauptspeicher
- **Implizite Adressierung: Akkumulator**
- **Überdeckung**
  - Operand im Akkumulator wird mit Ergebnis überschrieben

Ausführungsmodelle

- **Akkumulator-Architektur**
  - Der Akkumulator wird bei einer zweistelligen Operation als Quelle einer der beiden Operanden angesprochen, gleichzeitig dient der Akkumulator als Ziel für das Resultat.

```
ld B          ; acc ← mem[B]
add A         ; acc ← acc + mem[A]
```

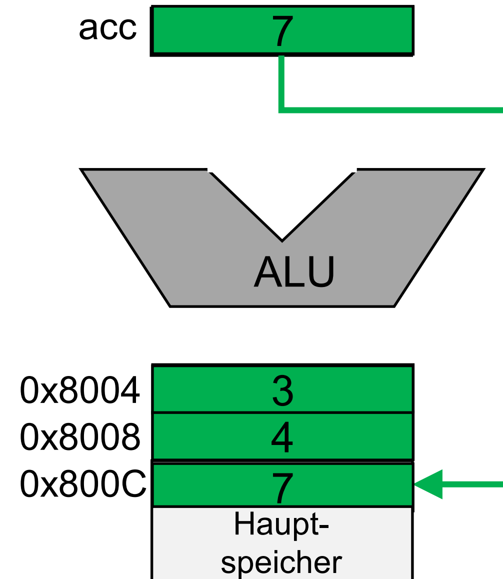


© Prof. Dr. Wolfgang Karl

Ausführungsmodelle

- **Akkumulator-Architektur**
  - Der Akkumulator wird bei einer zweistelligen Operation als Quelle einer der beiden Operanden angesprochen, gleichzeitig dient der Akkumulator als Ziel für das Resultat

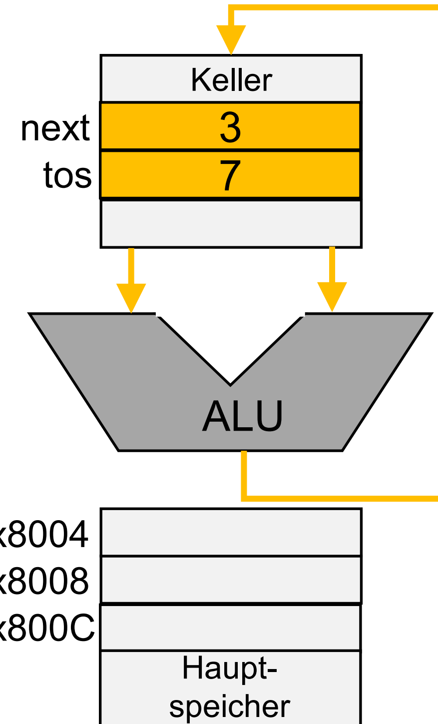
```
ld B        ; acc ← mem[B]
add A       ; acc ← acc + mem[A]
st S        ; mem[S] ← (acc)
```



Ausführungsmodelle

- **Keller-Architektur**
  - Die beiden Operanden einer zweistelligen Operation stehen auf den beiden obersten Kellerelementen.
  - Das Ergebnis wird wieder auf dem Keller abgelegt.

```
add          ; tos ← tos + next
             ; tos: top of stack
```

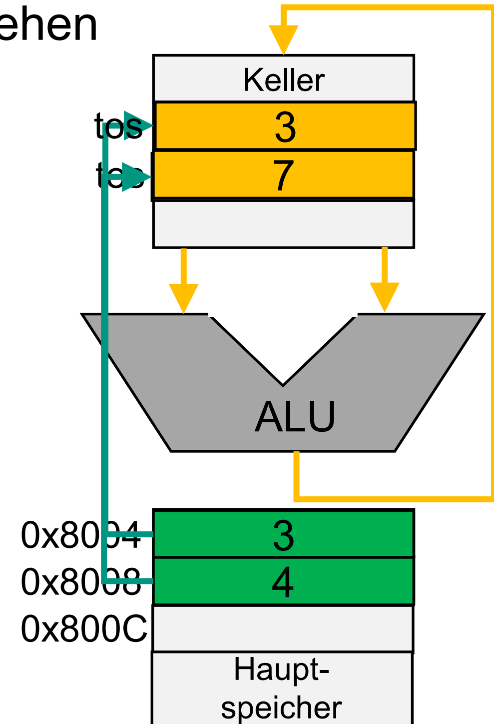


- **Nulladressformat**
- **Implizite Adressierung**
  - Kellerzeiger (tos)
- **Überdeckung**
  - Operand im obersten Kellerelement wird mit Ergebnis überschrieben

Ausführungsmodelle

- **Keller-Architektur**
  - Die beiden Operanden einer zweistelligen Operation stehen auf den beiden obersten Kellerelementen
  - Das Ergebnis wird wieder auf dem Keller abgelegt

```
push A          ; tos ← mem[A]
push B          ; tos ← mem[B]
add             ; tos ← tos + next
```



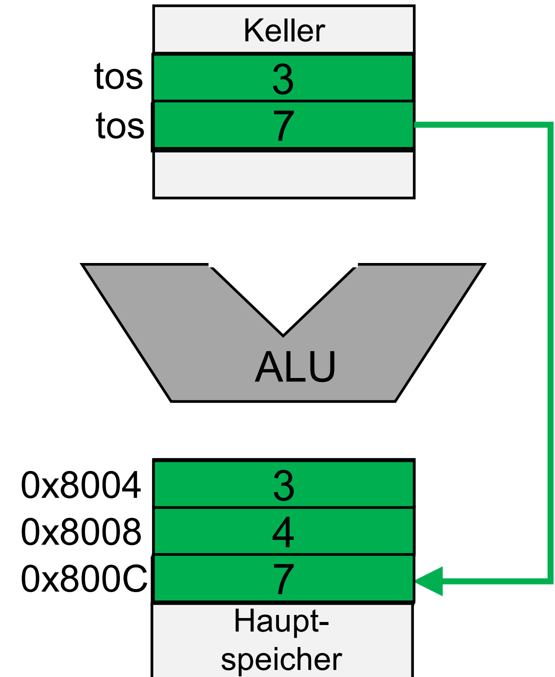
Ausführungsmodelle

Keller-Architektur

- Die beiden Operanden einer zweistelligen Operation stehen auf den beiden obersten Kellerelementen
- Das Ergebnis wird wieder auf dem Keller abgelegt

```
push A          ; tos ← mem[A]
push B          ; tos ← mem[B]
add             ; tos ← (tos) + (next)
pop S           ; mem[S] ← tos
```

- Beispiele:
  - 80x86 Gleitkomma-Registerorganisation,
  - Java Virtual Machine (JVM)



© Prof. Dr. Wolfgang Karl

KIT

Befehlssatzarchitektur

- **Datentypen**
  - Sind die unterschiedlichen Interpretationsarten, die vom Mikroprozessor direkt unterstützt werden (Hardwaresicht);
  - Kategorien:
    - Zahlen
    - Buchstaben
    - Logische Daten
  - Sind charakterisiert durch ein **Datenformat** und die **inhaltliche Interpretation**
    - Spezifikation der Wertebereiche, die Konstanten, Ausdrücke, Variablen oder Funktionen in Programmen annehmen können;
    - Die Interpretation wird durch die einzelnen Befehle vorgegeben und im Opcode codiert.
  - Die für einen Datentyp bestimmten Operationen interpretieren die Operanden in gleicher Weise.

© Prof. Dr. Wolfgang Karl

Befehlssatzarchitektur

- **Datentypen**
  - Datentypen, die nicht in der Hardware unterstützt werden, müssen durch ein geeignetes Programm auf elementare Datentypen zurückgeführt und in mehreren Schritten berechnet werden.
  - Befehlssatzarchitekturen stellen auch Datentypen bereit, die nicht in Hardware verarbeitet werden können (z.B. Gleitkomma-Datentypen)
    - 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;

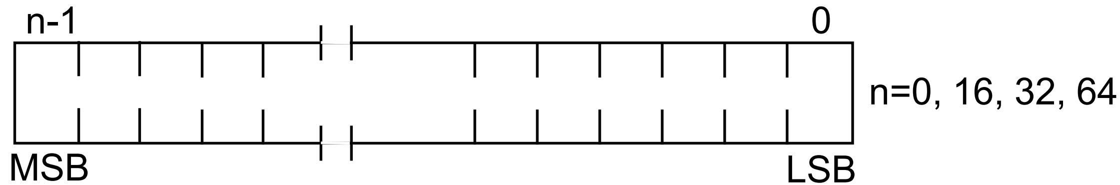
Befehlssatzarchitektur

- **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 wurde der Begriff “Wort” für
    16-Bit benutzt.

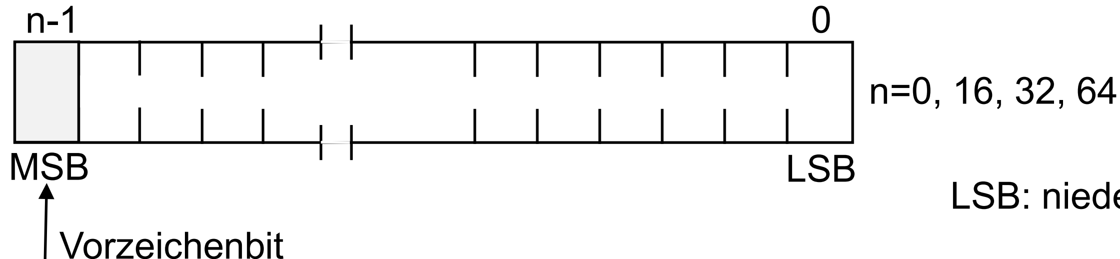
© Prof. Dr. Wolfgang Karl

Datentypen

- **Zahlen**
  - **Vorzeichenlose Dualzahl**
    - Standardformate n = 8,16, 32 oder 64; Wertebereich: 0 bis 2^n-1



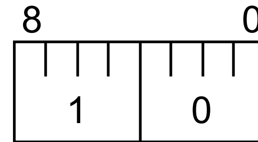
  - **2'er Komplement (signed Integer):**
    - Standardformate n = 8,16, 32 oder 64; Wertebereich: -2^{n-1} bis $+2^{n-1}-1$

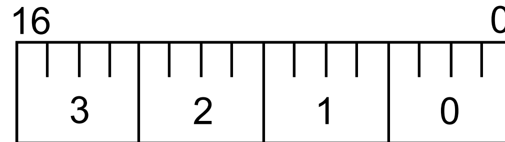


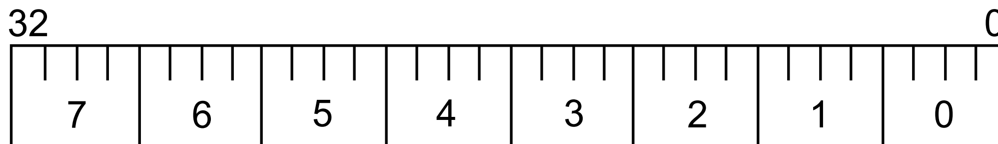
LSB: niederwertigstes Bit (Least Significiant Bit)

Datentypen

- **Zahlen**
  - **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;



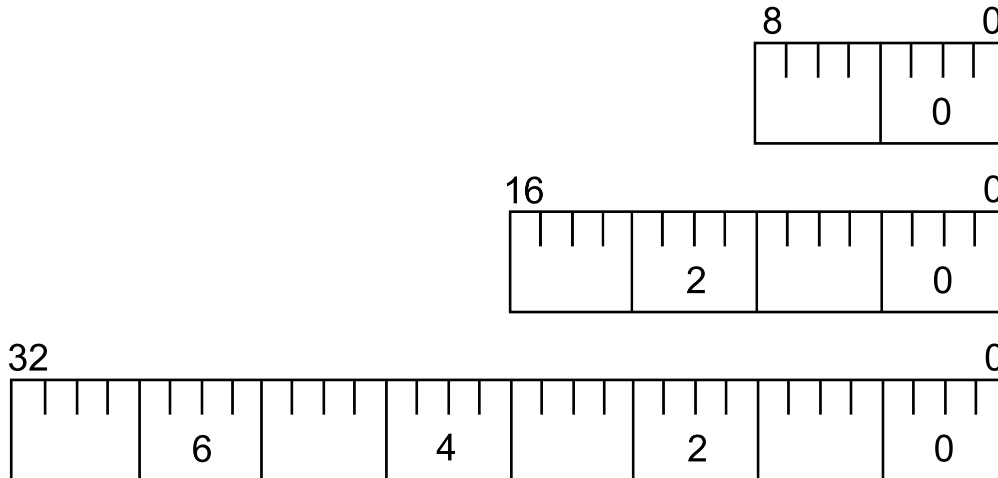




© Prof. Dr. Wolfgang Karl

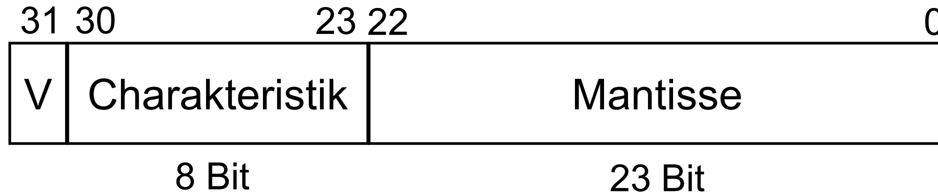
Datentypen

- **Zahlen**
  - **BCD-Ziffern in ungepackter Form (unpacked binary digitals)**
    - 1, 2, oder 4 Bytes werden in den Standardformaten zusammen gefasst.
    - Ein Byte enthält zusätzliche Bits im höherwertigen Halbbyte.
    - Exakte Ergebnisse bezüglich Dezimalzahlen.

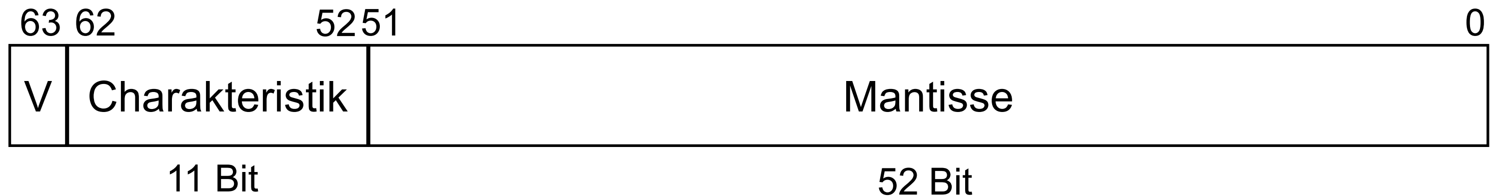


Datentypen

- **Zahlen**
  - **Gleitkommazahlen**
    - Einfache Genauigkeit

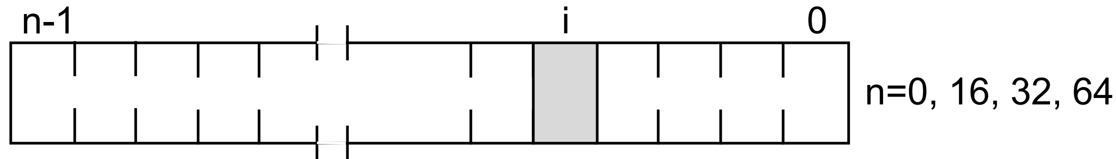


    - Doppelte Genauigkeit

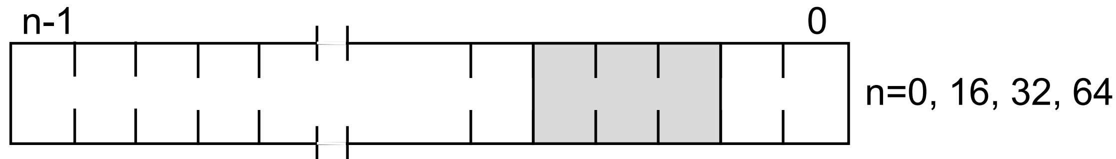


Datentypen

- **Logische Daten**
  - **Zustandsgröße Bit:**
    - In einem der Standardformate



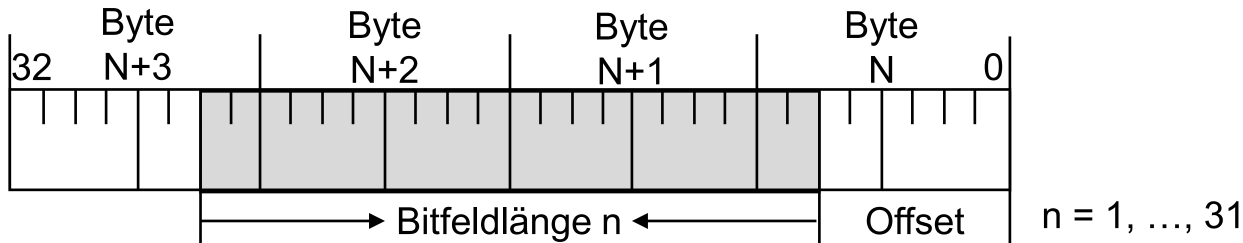
  - **Bitvektor**
    - Zusammenfassung von Zustandsgrößen (Standardformate)



© Prof. Dr. Wolfgang Karl

Datentypen

- **Bitfeld**
  - Darstellung und Verarbeitung von Bitvektoren, vorzeichenlosen Dualzahlen und 2-Komplementzahlen;
  - variable Bit-Anzahl: 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

- **Zeichen**
  - **Zeichenkette (String)**
    - Verarbeitung von aufeinander folgenden gespeicherten Bytes, Halbwörter oder Wörter
    - Byte-Strings: bestehend aus ASCII-Zeichen, Unicode

© Prof. Dr. Wolfgang Karl

Datentypen

- **Fallstudie IA-32 (Intel Architecture 32-Bit, Intel x86 ISA)**
  - Ordinal
    - vorzeichenlose Dualzahl in den Standardformaten
  - Integer
    - 2er-Komplementzahl in den Standardformaten
  - Packed BCDs
    - binär codierte Dezimalzahl in gepackter Darstellung
  - Unpacked BCDs
    - binär codierte Dezimalzahl in ungepackter Darstellung

Datentypen

- **Fallstudie IA-32**
  - Floating-Point Data Types (Gleitkomma-Zahlen)
    - Halbe Genauigkeit (16 Bit Format)
    - Einfache Genauigkeit (32 Bit Format)
    - doppelte Genauigkeit (64 Bit Format)
    - Erweiterte Genauigkeit (80 Bit Format)

© Prof. Dr. Wolfgang Karl

Datentypen

- **Fallstudie IA-32**
  - Bit fields
    - Bitfelder
  - Bit Strings
    - Folge von Bits, Bytes, Wörtern oder Doppelwörtern
    - Kann an jeder Position in einem Byte beginnen und bis zu 2^{32}-1 Bits enthalten
  - Byte-String
    - Folge von Bytes, Words oder Doublewords in einem Bereich von 0 bis 2^{32}-1 Bytes
  - 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

© Prof. Dr. Wolfgang Karl

Datentypen

- **Fallstudie MIPS64 Architektur**
  - MIPS64 Operationen arbeiten auf 64 Bit Integer und 32- oder 64 Bit Gleitkommadaten
  - Byte-, Half Word- und Word-Daten werden in Allzweckregister (GPRS) geladen mit Null- oder Vorzeichenerweiterung

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Byte-adressierbarer Speicher**
  - Direkt zugreifbar im Speicher sind das Byte, Halbwort oder das Wort, wobei sich die Adressen, unabhängig vom Datenformat, auf Bytegrenzen beziehen.
  - Byte-Adressen:
    - kleinste adressierbare Einheit: Byte

Hauptspeicher

Byte-Adresse	**Inhalt**
0x0111	00010100
0x0110	10101010
0x0101	10010110
0x0100	11010010
0x0011	10101111
0x0010	10001111
0x0001	00000101
0x0000	00000001

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Wort-organisierter Speicher**
  - die Zugriffsbreite ist, bei optimaler Auslegung, gleich der Datenbusbreite
    - z.B. 32 Bit bei 32-Bit Mikroprozessoren oder 64 Bit bei 64/32- bzw. 64-Bit Prozessoren

Hauptspeicher

Byte-Adresse	Inhalt
0x0111	00010100
0x0110	10101010
0x0101	10010110
0x0100	11010010
0x0011	10101111
0x0010	10001111
0x0001	00000101
0x0000	00000001

Die Adressen von aufeinanderfolgenden Wörtern unterscheiden sich um 4.

Eine **Wortadresse** ist die Adresse eines der 4 Bytes in einem Wort.

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Wort-orientierter Speicher**
  - **Ausgerichtete Daten (data alignment, alignment restriction)**
    - Wörter (4 Bytes) müssen an Adressen beginnen, die ein Vielfaches von 4 sind.
    - Allgemein:
      - Ein Wort in einem Format bestehend aus **`s`** Bytes ist im Speicher ausgerichtet abgelegt, wenn seine Adresse **`A`** ein ganzzahliges Vielfaches von **`s`** ist, d.h. wenn gilt:

$$A \bmod s = 0$$

    - Falls **`s`** keine Potenz von 2 ist, muss **`s`** auf die nächsthöhere Potenz von 2 aufgerundet werden.
      - Beispiel: Ein 6-Byte langes Datenelement muss gemäß einem 8-Byte langem Datenelement ausgerichtet werden.

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Wort-orientierter Speicher**
  - **Ausgerichtete Daten (data alignment, alignment restriction)**
    - Pro Taktzyklus kann auf eine Folge von **m** Bytes zugegriffen werden, wobei **m** die Breite des Datenbusses in Bytes ist.
    - Wenn die Operandenbreite **s** kleiner gleich der Wortbreite **m** des Speichers ist, dann ist nur Speicherzugriff notwendig.
    - Vorgeschrieben bei RISC-V und allen RISC-Prozessoren;

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Wort-orientierter Speicher**
  - **Nicht-ausgerichtete Daten (data misalignment)**
    - Speicherung von Operanden in den Datenformaten Halbwort, Wort, Doppelwort etc. an beliebigen Byteadressen;
    - Vorteil: lückenlose Nutzung des Speichers bei beliebiger Mischung der Datenformate;
    - Nachteil: abhängig von der Anordnung der Daten im Speicher müssen ggf. mehr Speicherzugriffe erfolgen, als eigentlich für ein solches Datenelement mindestens nötig wären.

© Prof. Dr. Wolfgang Karl

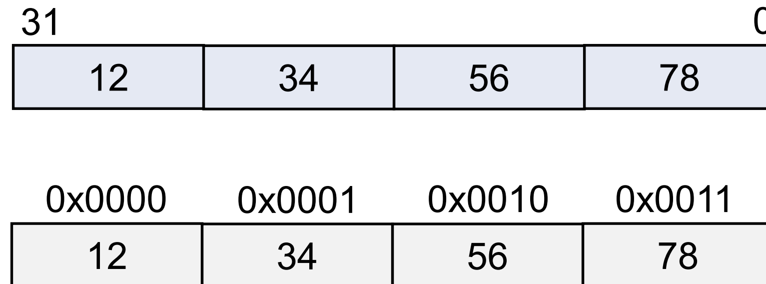
Datenzugriff

- **Wort-orientierter Speicher**
  - **Nicht-ausgerichtete Daten (data misalignment)**
    - **Beispiel Intel Prozessoren:**
      - Schreiben die Ausrichtung von Daten im Speicher nicht vor.
      - Erlauben also den Zugriff auf Datenelemente im Word-, Doubleword oder Quadword-Format, die nicht an 4-Bytes- oder 8-Bytes-Grenzen ausgerichtet abgelegt sind;
      - Beim Speicherzugriff ist eventuell ein zusätzlicher Taktzyklus erforderlich.
      - Ein Wort, das an einer ungeraden Adresse beginnt und nicht über eine 4-Bytes-Grenze geht, wird als ausgerichtet betrachtet.

© Prof. Dr. Wolfgang Karl

Datenzugriff

- **Anordnung der Daten im Speicher**
  - **Big Endian Byte Ordering**
    - Datenelemente in Datenformaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niederwertige Byte an der höchstwertigen Adresse und das höchstwertige Byte an der niederwertigen Adresse steht.
    - Die Wortadresse ist die Adresse des höchstwertigen Bytes.
    - Beispiel: 32-Bit Wert 12345678_{16}



Hauptspeicher

Byte-Adresse	Inhalt
0x0011	78
0x0010	56
0x0001	34
0x0000	12

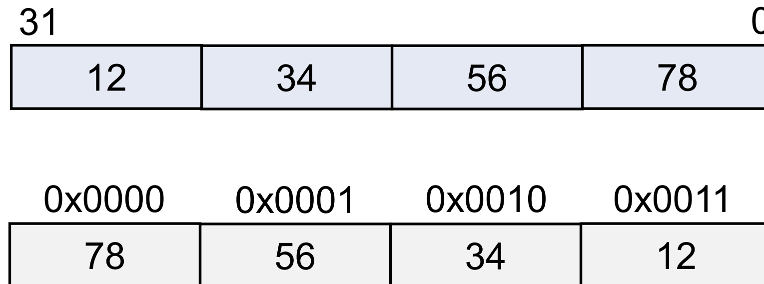
© Prof. Dr. Wolfgang Karl

Datenzugriff

Anordnung der Daten im Speicher

Little Endian Byte Ordering

- Datenelemente in Datenformaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niederwertige Byte an der niederwertigen Adresse und das höchstwertige Byte an der höchstwertigen Adresse steht.
- Die Wortadresse ist die Adresse des niederwertigen Bytes.
- Beispiel: 32-Bit Wert 12345678_{16}



Hauptspeicher

Byte-Adresse	Inhalt
0x0011	12
0x0010	34
0x0001	56
0x0000	78

© Prof. Dr. Wolfgang Karl