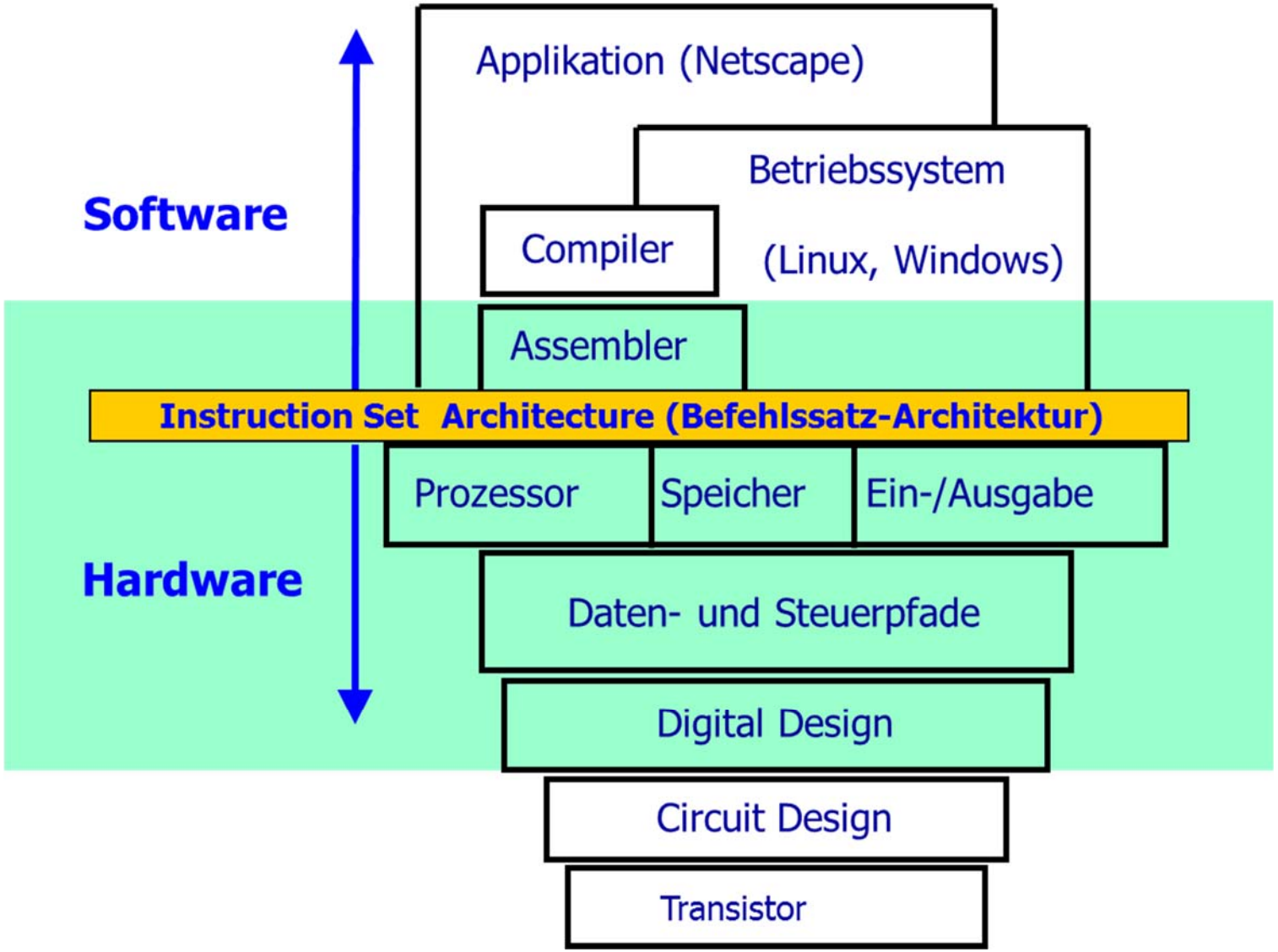
Kapitel 4
Die Befehlssatzarchitektur

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

4. Befehlssatzarchitektur



4. Befehlssatzarchitektur

- **Der Begriff „Befehlssatzarchitektur“ (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

4.1 Ausführungsmodelle

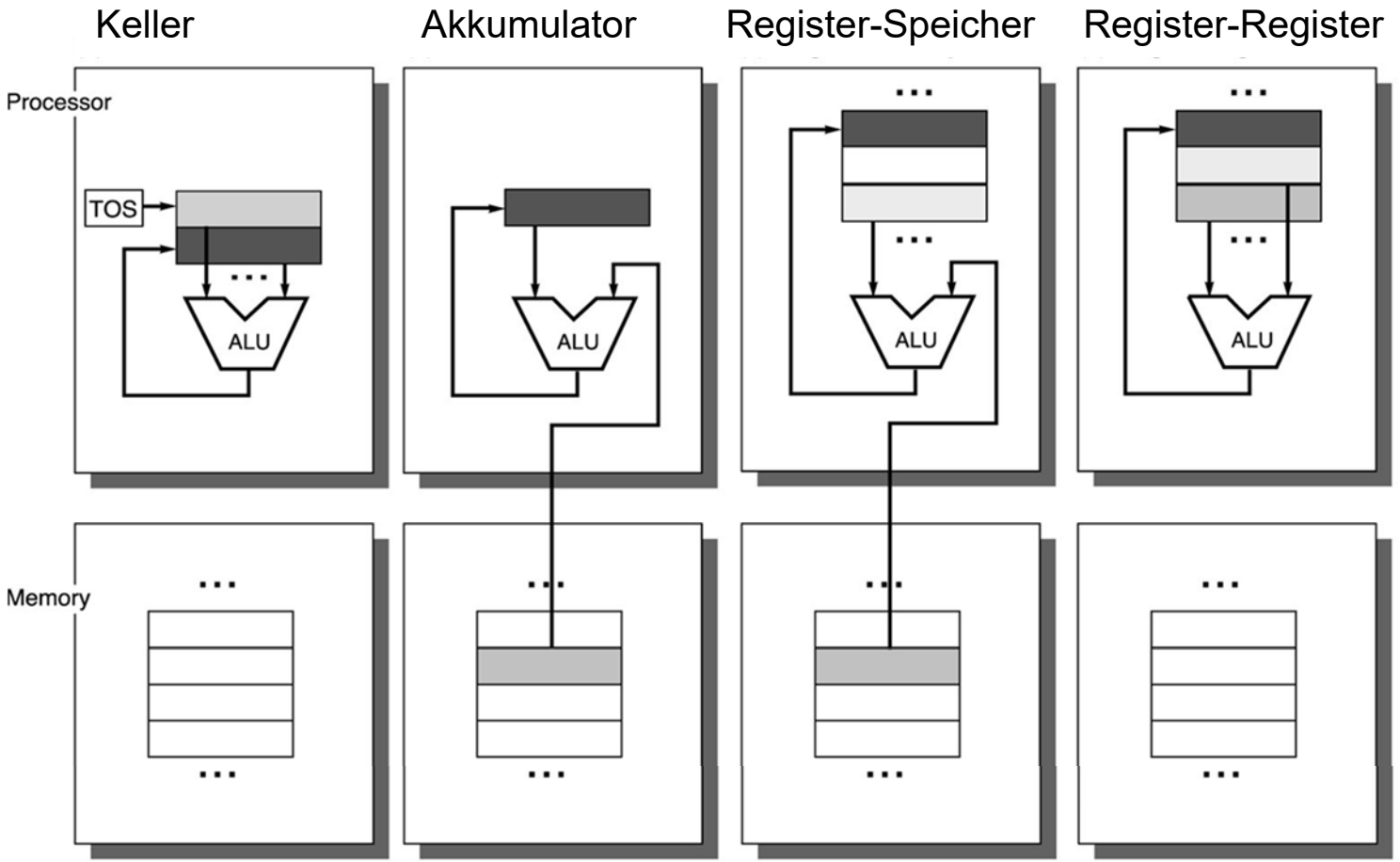
- **Klassen von Architekturen, Ausführungsmodelle**
  - Für eine zweistellige Operation, d.h. bei einer Operation, bei der die zwei Operanden miteinander verknüpft werden, sind folgende Angaben notwendig
    - Art der Operation
    - Adresse des ersten Operanden
    - Adresse des zweiten Operanden
 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

4.1 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 festes Befehlsformat

4.1 Ausführungsmodelle

Klassen von Architekturen, Ausführungsmodelle



4.1 Ausführungsmodelle

- **Register-Register-Modell**
- Alle Operanden und das Ergebnis stehen in Allzweckregistern

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

  - Dreiadressformat

- Load/Store-Architektur
  - Dedizierte 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
```

4.1 Ausführungsmodelle

- **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
    - ARM, MIPS, PowerPC, SPARC, …

4.1 Ausführungsmodelle

- **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)
    - Überdeckung einer Quelladresse mit einer Zieladresse

4.1 Ausführungsmodelle

- **Register-Speicher-Modell**
  - Vorteile:
    - 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

4.1 Ausführungsmodelle

- **Akkumulator-Modell**
- 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

4.1 Ausführungsmodelle

- **Keller-Modell**
- 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
```

  - Implizite Adressierung
    - über den Kellerzeiger (tos)
  - Überdeckung
  - Nulladressformat
  - Beispiele:
    - Burroughs B5000 (1968), 80x86 Gleitkomma-Architektur, Java Virtual Machine (JVM)

4.1 Ausführungsmodelle

- **Speicher-Speicher-Modell**
- 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

4.1 Ausführungsmodelle

- **Beispiel:**
  - **Programm C=A+B; D=C-B; in den vier Befehlsformatsarten codiert**

Register-Register	Register-Speicher	Akkumulator	Keller

`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`
	`load Reg1,A`
	`add Reg1,B`
	`store C,Reg1`
	`load Reg1,C`
	`sub Reg1,B`
	`store D,Reg1`
		`load A`
		`add B`
		`store C`
		`load C`
		`sub B`
		`store D`
			`push B`
			`push A`
			`add`
			`pop C`
			`push B`
			`push C`
			`sub`
			`pop D`

4.2 Datentypen und Datenformate

- **Datentypen**
  - Spezifizieren die Wertebereiche, die Konstanten, Ausdrücke, Variablen oder Funktionen in Programmen annehmen können
  - Sind die unterschiedlichen Interpretationsarten, die vom Mikroprozessor direkt unterstützt werden (Hardwaresicht)
  - 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

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

4.2 Datentypen und Datenformate

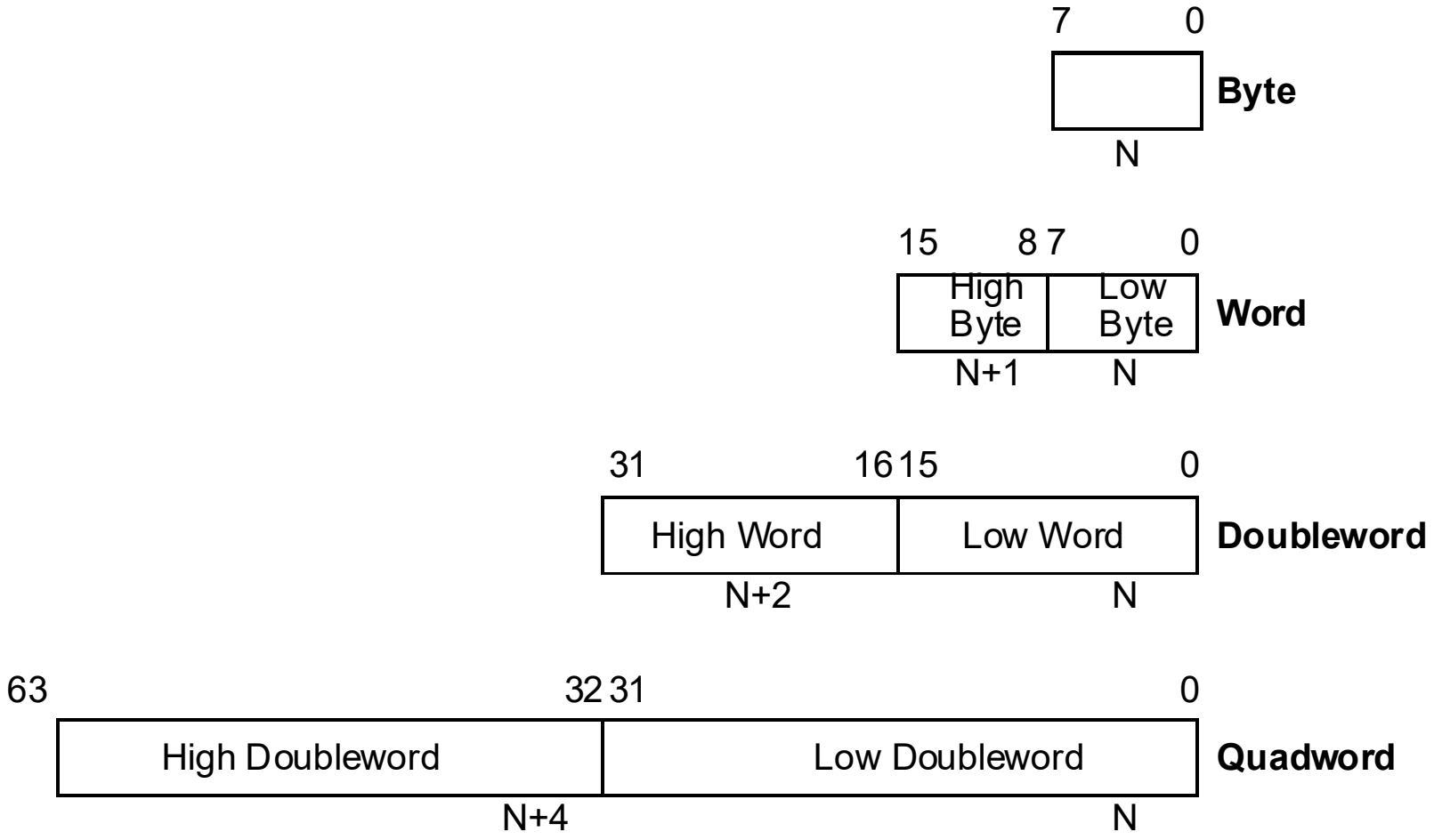
- **Datentypen**
  - Datentypen, die nicht von der Hardware unterstützt werden, müssen durch ein geeignetes Programm auf elementare Datentypen zurückgeführt und in mehreren Schritten berechnet werden
  - Prozessoren unterstützen auch Datentypen, die sie in ihrem Rechenwerk nicht verarbeiten können
    (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

4.2 Datentypen und Datenformate

- **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.

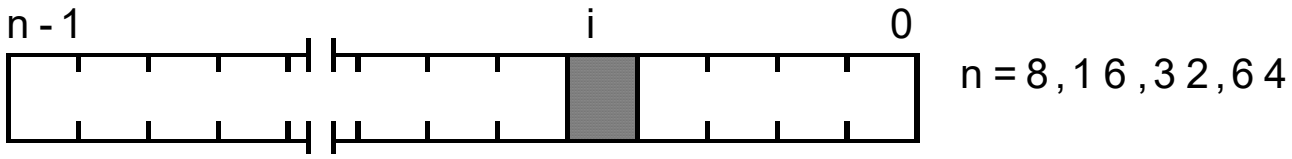
4.2 Datentypen und Datenformate

- **Datenformate: Fallstudie IA-32**
  - Grundlegende Datenformate

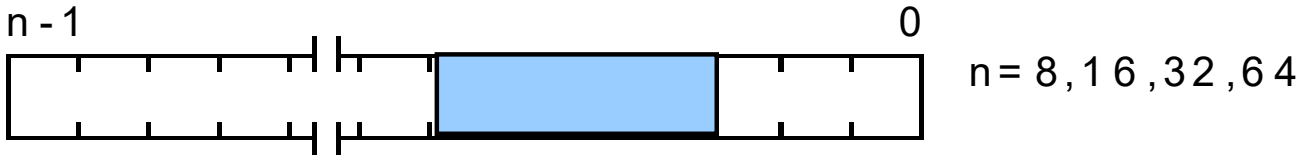


4.2 Datentypen und Datenformate

- **Datentypen**
  - Zustandsgröße Bit:
    - ein Bit in einem der Standardformate

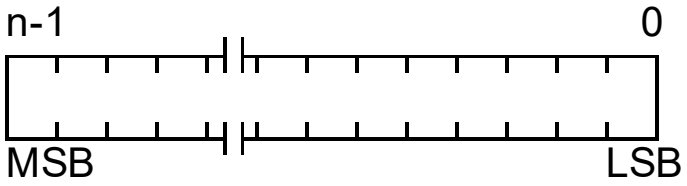


  - Bitvektor:
    - Zusammenfassung von Zustandsgrößen (Standardformate)



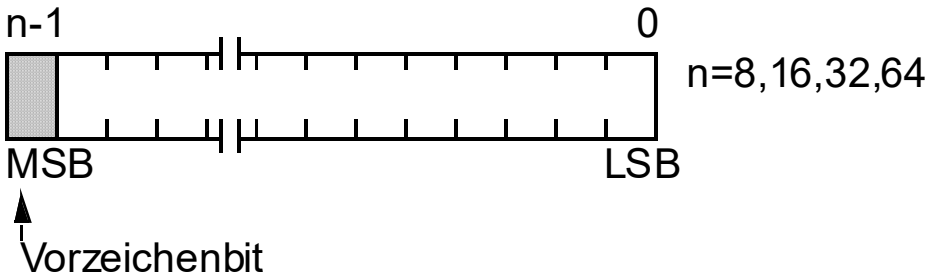
4.2 Datentypen und Datenformate

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



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

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



4.2 Datentypen und Datenformate

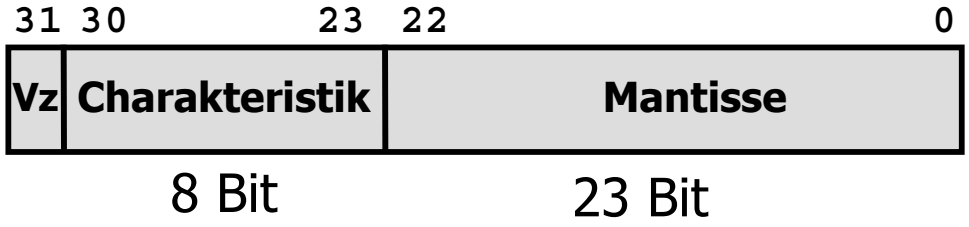
- **Datentypen:**
  - BCD-Ziffern in gepackter Form (packed binary digitals)
    - 2, 4 oder 8 Halbbytes werden in den Standardformaten zusammen gefasst;
    - ziffernweises Codieren der Dezimalzahlen im 8-4-2-1 Dualcode, d. h. das Codewort umfasst 4 Bits mit den Gewichten 8,4,2 und 1
    - Exakte Ergebnisse bezüglich Dezimalzahlen
      - 0,1010 in Binärdarstellung: $0{,}0001100110011\ldots_2$

4.2 Datentypen und Datenformate

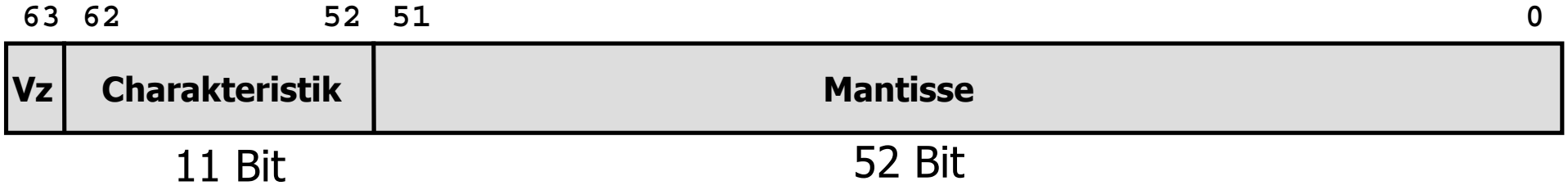
- **Datentypen:**
  - BCD-Ziffern in ungepackter Form
    - 1, 2, oder 4 Bytes werden in den Standardformaten zusammen gefasst;
    - Ein Byte enthält zusätzliche Bits im höherwertigen Halbbyte
    - Exakte Ergebnisse bezüglich Dezimalzahlen

4.2 Datentypen und Datenformate

- **Datentypen:**
  - Gleitkommazahlen
    - Einfache Genauigkeit:

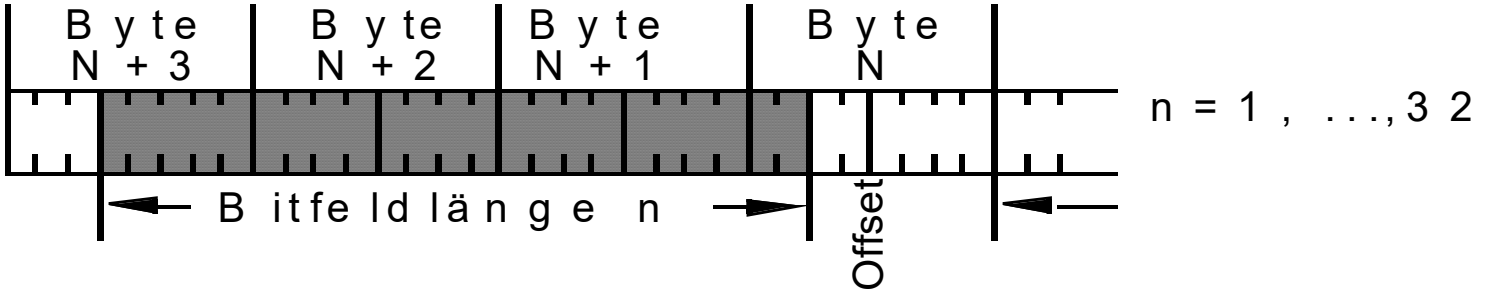


    - Doppelte Genauigkeit:



4.2 Datentypen und Datenformate

- **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



4.2 Datentypen und Datenformate

- **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

4.2 Datentypen und Datenformate

- **Datentypen: Fallstudie: IA-32**
  - Ordinal bzw. unsigned integers (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)
  - 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 232-1 Bits enthalten kann und Byte-String Bytes,Words oder Doublewords enthalten kann in einem Bereich von 0 bis 232-1 Bytes)
  - Floating-Point Data Types (Gleitkomma-, Integer und BCD-Zahlen)

4.2 Datentypen und Datenformate

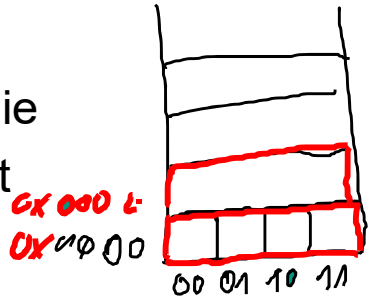
- **Datentypen: Fallstudie MIPS64 Architektur**
  - MIPS64 Operationen arbeiten auf 64 Bit Integer und 32- oder 64 Bit Gleitkommadaten
  - Byte-, Half Word- und Word-Daten werden in GPRS geladen mit Null- oder Vorzeichenerweiterung

4.3 Speicheradressierung

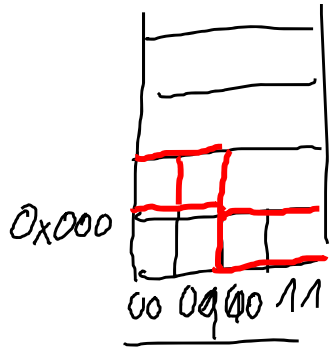
- **Datenzugriff**
  - **Byte-adressierbarer Speicher**
    - direkt zugreifbar im Speicher ist das Byte, Halbwort oder das Wort, wobei sich die Adressen, unabhängig vom Datenformat auf Bytegrenzen beziehen (Byteadressen)
  - **Wort-organisierter Speicher**
    - die Zugriffsbreite ist (bei optimaler Auslegung) gleich der Datenbusbreite
      - (z.B. 32 Bit bei 32-Bit Mikroprozessoren oder 64 Bit bei 64/32- bzw. 64-Bit Prozessoren)
    - die Zugriffsbreite kann bei einem Prozessor, der eine dynamische Anpassung der Datenbusbreite vorsieht, auf 16 Bit (Halbwort-organisiert) oder 8 Bit (Byte-organisiert) reduziert sein

Speicheradressierung

- **Datenzugriff: Ausrichten der Daten im Speicher**
  - **Ausgerichtete Daten** (data alignment)
    - ein Datum 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 *A mod s = 0* ist
    - ein Speicherzugriff pro Operand: Gilt nur dann, wenn die Operandenbreite ≤ Wortbreite des Speichers ist



  - **Nicht-ausgerichtete Daten (data misalignment)**
    - Speicherung von Operanden in den Datenformaten Halbwort, Wort, Doppelwort etc. an beliebigen Byteadressen

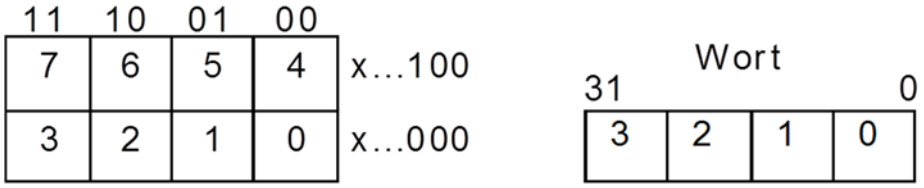


Speicheradressierung

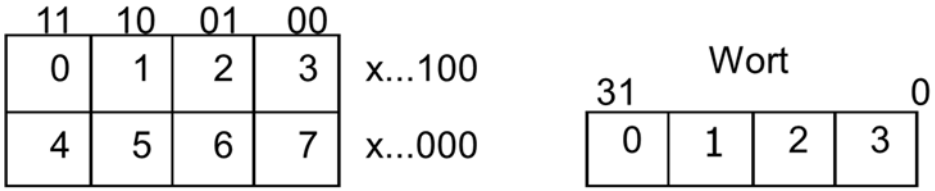
- **Datenzugriff**
  - **Nicht-ausgerichtete Daten (data misalignment)**
    - Vorteil: lückenlose Nutzung des Speichers bei beliebiger Mischung der Datenformate
    - Nachteil: zusätzlich Speicherzugriffe
    - Beispiel Intel Pentium Pro:
      - erlaubt den Zugriff auf Operanden im Word-, Doubleword oder Quadword-Format, die über die 4-Bytes- oder 8-Bytes-Grenzen abgelegt sind, was allerdings einen zusätzlichen Taktzyklus erfordert; ein Wort, das an einer ungeraden Adresse beginnt und nicht über eine 4-Bytes-Grenze geht, wird als ausgerichtet betrachtet
    - Beispiel MIPS R2000, alle RISC Prozessoren
      - Ausrichten der Daten ist vorgeschrieben

Speicheradressierung

- **Anordnung der Daten im Speicher**
  - **Little Endian Ordering**
    - Daten in Formaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niedrigstwertige Byte an der niedrigstwertigen Adresse und das höchstwertige Byte an der höchstwertigen Adresse steht



  - **Big Endian Byte Ordering**
    - Daten in Formaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niedrigstwertige Byte an der höchstwertigen Adresse und das höchstwertige Byte an der niedrigstwertigen Adresse steht



4.4 Adressierungsarten

- **Programm-, Prozess-, Maschinenadresse**
  - **Programmadresse:**
    - In Befehlen und als Adresswerte im Programm vorliegende Adressen
    - **Adressierungsarten**
      - Beschreiben die verschiedenen Möglichkeiten, die Speicheradressen von Operanden oder Sprungzielen im Befehl zu spezifizieren
    - Nach Vorgaben des Programms erzeugt der Prozessor aus Programmadressen Prozessadressen
      - Adressberechnung

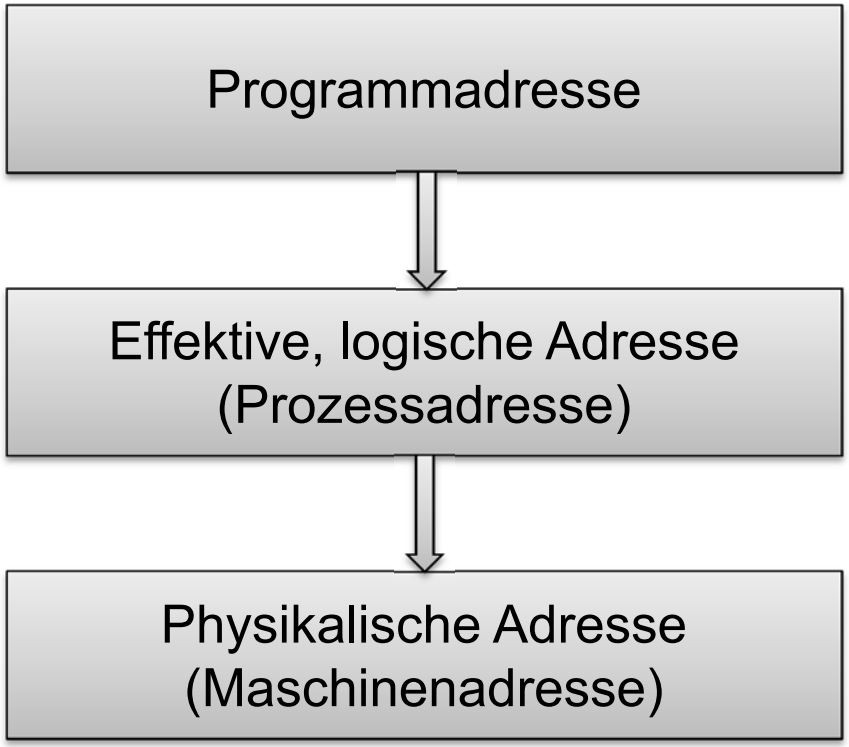
4.4 Adressierungsarten

- **Programm-, Prozess-, Maschinenadresse**
  - **Prozessadresse (effektive Adresse):**
    - Verwendet der Prozessor
    - Nach Vorgaben des Betriebssystems erzeugt der Prozessor aus Prozessadressen Maschinenadressen
    - Hauptziel:
      - Beliebige Lage des Programms und seiner Werte
      - partielle Lagerung im Speicher
  - **Maschinenadresse:**
    - Verwendet der Prozessor gegenüber Hauptspeicher

4.4 Adressierungsarten

- **Programm-, Prozess-, Maschinenadresse**

Ablauf der Adressberechnung



Dynamische Adressberechnung
(gemäß der im Befehl spezifizierten Adressierungsart)

Speicherverwaltungseinheit
(Virtuelle Speicherverwaltung)

4.4 Adressierungsarten

- **Programm-, Prozess-, Maschinenadresse**
  - **Effektive Adresse**:
    - Eine effektive Adresse entsteht im Prozessor nach Ausführung der Adressrechnung.
    - Adressiert eine Speicherzelle im Hauptspeicher (kein virtuellen Speicher)
  - **Virtueller Speicher**
    - Es gilt: effektive Adresse = logische (virtuelle) Adresse
    - Die effektive Adresse wird mit Hilfe der Speicherverwaltungseinheit (Memory Management Unit, MMU) in eine physikalische Adresse umgesetzt
      - Physikalische Adresse: Adresse, mit der auf den Hauptspeicher zugegriffen wird
  - Im folgenden betrachten wir nur die Erzeugung einer effektiven Adresse aus den Angaben in einem Maschinenbefehl.

4.4 Adressierungsarten

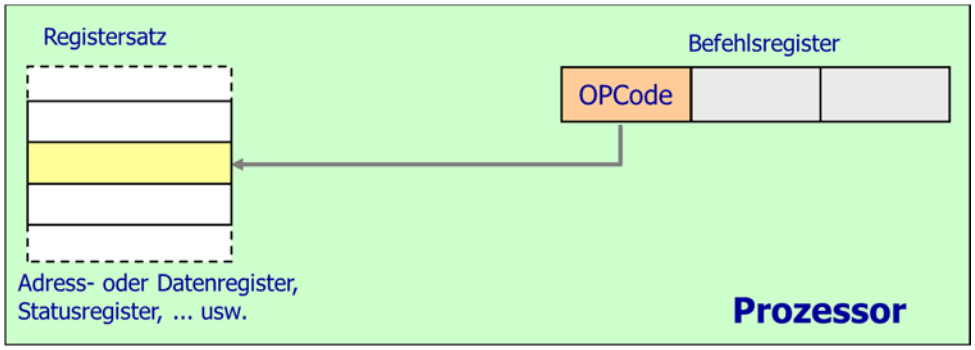
- **Registeradressierung**
  - Operand steht bereits im Register
    ➔ kein Speicherzugriff erforderlich
  - Implizite Adressierung
  - Explizite Registeradressierung
  - Flag-Adressierung.

4.4 Adressierungsarten

- **Registeradressierung**
  - **Implizite Adressierung** (inherente Adressierung, implied-, inherent addressing)
    - Die Nummer, d. h. die effektive Adresse des Registers ist implizit im Befehl (z. B. im Opcode) kodiert
    - Assemblerschreibweise:
      <Mnemo> A (A Akkumulator)
    - Effektive Adresse:
      EA ist codiert im OpCode enthalten

4.4 Adressierungsarten

- **Registeradressierung**
  - **Implizite Adressierung** (inherente Adressierung, implied-, inherent addressing)
    - Die Nummer, d. h. die effektive Adresse des Registers ist implizit im Befehl (z. B. im Opcode) kodiert
    - Assemblerschreibweise:
      <Mnemo> A (A Akkumulator)
    - Effektive Adresse:
      EA ist codiert im OpCode enthalten



Beispiel:
LSRA (logical shift right accumulator)
(Verschiebe den Inhalt des Akkumulators A eine Bitposition nach rechts)

4.4 Adressierungsarten

- **Registeradressierung**
  - **Flag-Adressierung:**
    - Spezialfall der impliziten Adressierung.
    - Bei ihr wird nicht ein ganzes Register angesprochen, sondern nur ein einzelnes Bit (Flag) in einem Register

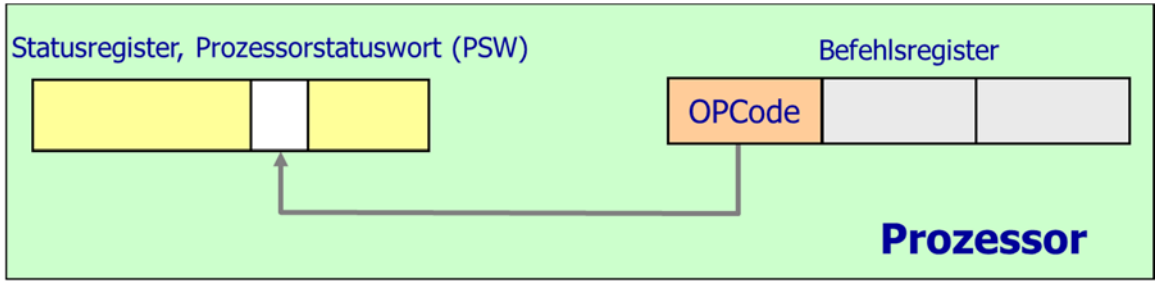
    - Assemblerschreibweise:

      SE<flag> (Flag setzen)

      CL<flag> (Flag rücksetzen)

    - Effektive Adresse:

      EA ist codiert im OpCode enthalten



Beispiel:
SEI/CLI (set / clear interrupt flag)
SEC/CLC (set / clear carry flag)

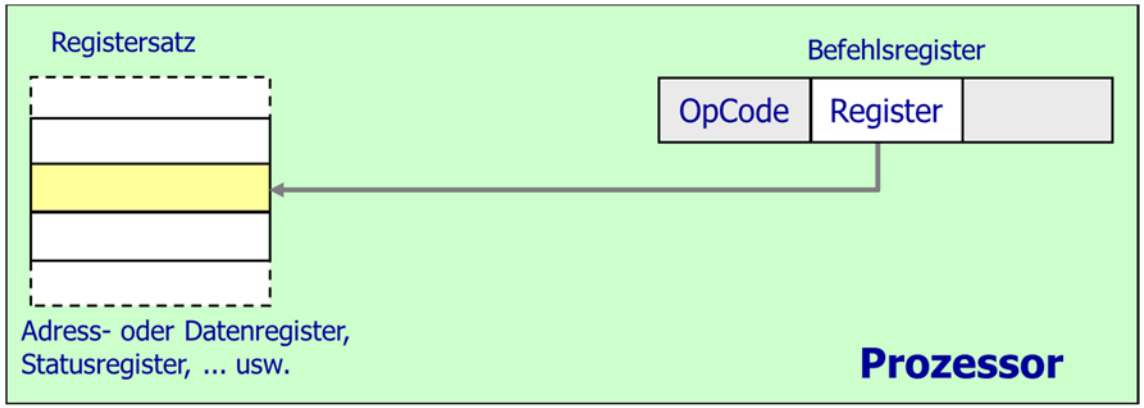
(Setzen / Zurücksetzen des Interrupt Enable Flags bzw. des Carry Flags.)

4.4 Adressierungsarten

- **Registeradressierung**
  - **Explizite Register-Adressierung (register operand addressing):**
    - Die Adresse (Nummer) des Registers wird im Operandenfeld des Befehls angegeben
    - Assemblerschreibweise:

      <Mnemo> Ri (Register i)
    - Effektive Adresse:

      EA = i



Beispiel:
DEC R0 (Decrement R0)

(Dekrementiere den Inhalt des Registers R0)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig
  - Unmittelbare Adressierung (immediate addressing)
  - Direkte Adressierung (direct addressing)
    - Absolute Adressierung
    - Seiten-Adressierung
  - Register-indirekte Adressierung (register indirect addressing)
  - Indizierte Adressierung (indexed addressing)
    - Speicher-relative Adressierung (memory relative addressing)
    - Register-relative Adressierung (register relative addressing)
    - Register-relative Adressierung mit Index (Based indexed mode)
  - Befehlszähler-relative Adressierung (PC relative addressing)