

Kapitel 4

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

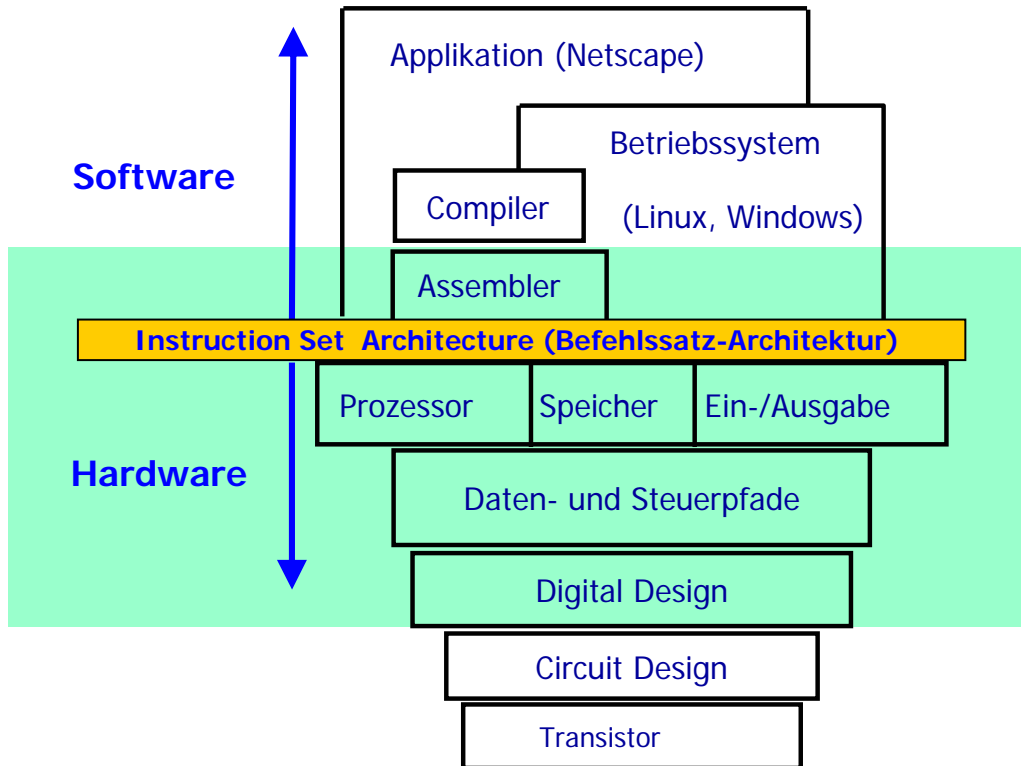
- Datentypen, Datenformate
- Befehlsformat, Befehlssatz
- Adressierungsarten
- Diskussion: RISC & CISC; Fallstudien (MIPS)



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





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



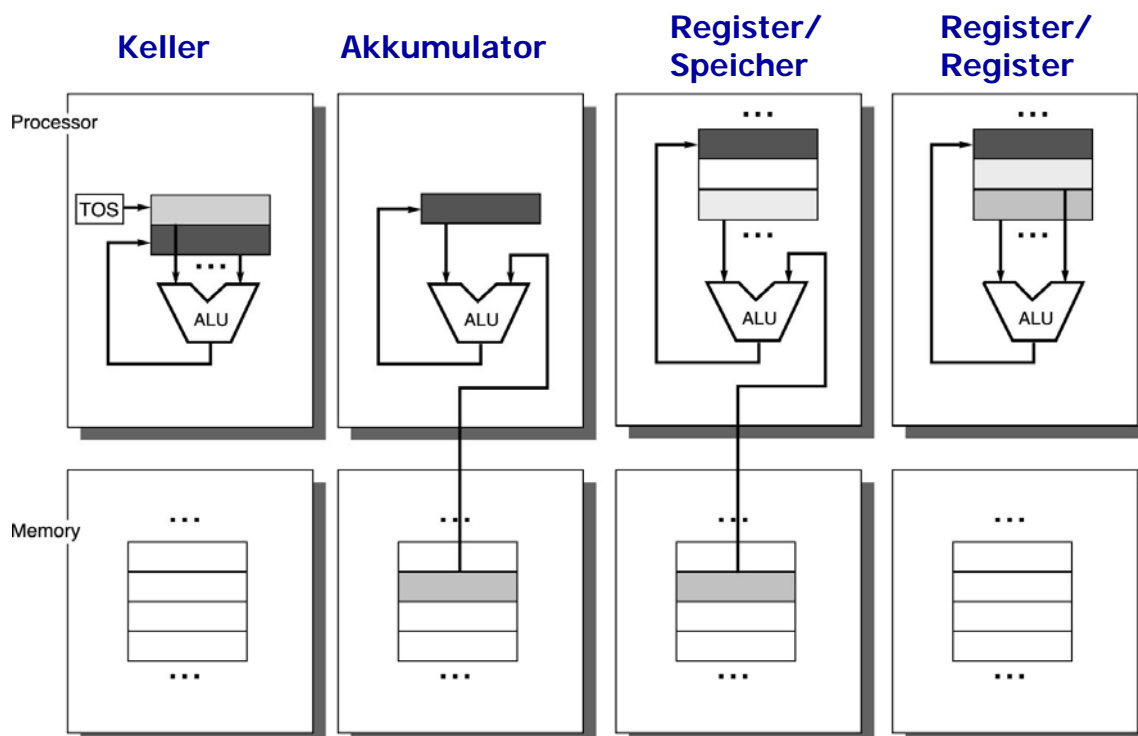
Ausführungsmodelle

□ 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` $R1 \leftarrow R2 + R3$

- Load/Store-Architektur

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

`load R2, A` $R2 \leftarrow \text{mem}[A]$

`load R3, B` $R3 \leftarrow \text{mem}[B]$

`add R1, R2, R3` $R1 \leftarrow R2 + R3$

`store C, R1` $\text{mem}[C] \leftarrow R1$

- 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` $\text{mem}[A] \leftarrow \text{mem}[A] + R1$
 - `add R1, A` $R1 \leftarrow R1 + \text{mem}[A]$

- explizite und überdeckte Adressierung
- 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 (Überdeckung)
 - 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** \leftarrow **acc** + **mem[A]**
 - **addx A** **acc** \leftarrow **acc** + **mem[A+x]**
- Implizite und überdeckte Adressierung
- Spezielle Befehle ermöglichen den Transport von Operanden
- Einadressformat



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



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



Ausführungsmodelle

□ Vergleich

- Bytes pro Instruktion → Code Größe
- Anzahl Instruktionen → Code-Größe, Laufzeit
- Zyklen pro Instruktion → Laufzeit
- Beispiel: `C = A+B`

| Keller | Akkumulator | Register (r/mem) | Register (load/store) |
|----------------------------------|----------------------------|----------------------------------|--|
| Push a Push b ADD Pop C | Load A ADD B Store C | Load R1,A ADD R1,B Store C | Load R1,A Load R2,B Add R3,R1,R2 Store C,R3 |



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

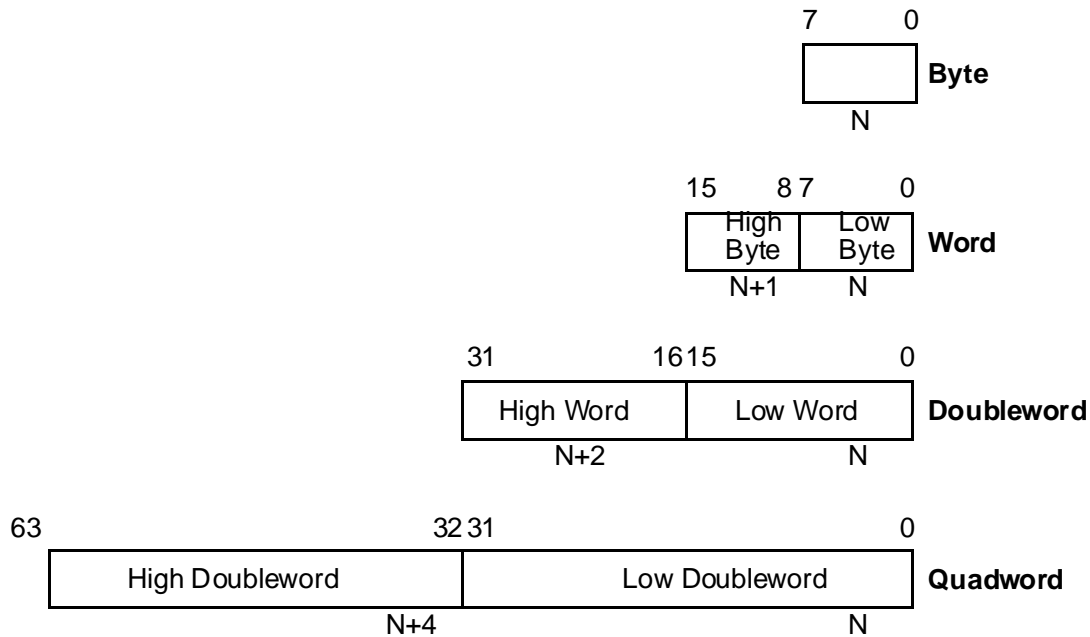
- Standardformate
 - Byte: 8 Bit
 - Halbwort: 16 Bit
 - Wort: 32 Bit
 - Doppelwort: 64 Bit
 - Der Begriff "Wort" wird von den Herstellern unterschiedlich verwendet. Er orientiert sich z. B. an der Verarbeitungsbreite von 32-Bit Prozessoren. Bei Intel wird der Begriff "Wort" für das 16-Bit
- Beispiel: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



Architektur (ISA)

□ Datenformate: Fallstudie IA-32

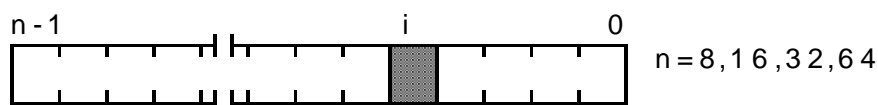
➤ Fundamental Data Types



Datentypen

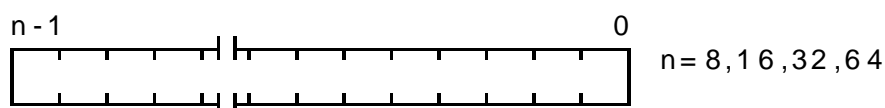
□ Zustandsgröße Bit:

➤ ein Bit in einem der Standardformate



□ Bitvektor:

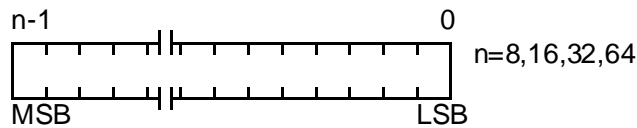
➤ Zusammenfassung von Zustandsgrößen (Standardformate)



Datentypen

□ Vorzeichenlose Dualzahl

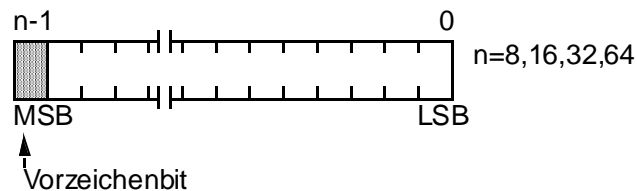
- Standardformate $n=8,16,32$ oder 64 ; Wertebereich: 0 bis 2^n-1



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

□ 2'er Komplement (signed Integer):

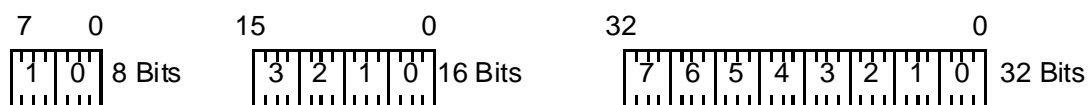
- Standardformate $n=8,16,32$ oder 64 ; Wertebereich: -2^{n-1} bis $+2^{n-1}-1$



Datentypen

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

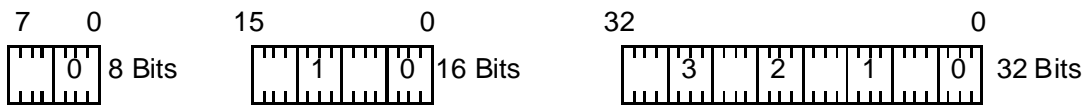
- 2, 4 oder 8 Halbbytes werden in den Standardformaten zusammen gefasst;
- ziffernweises Codieren der Dezimalzahlen im 8 4 2 1 Dualcode, d.h das Codewort umfasst 4 Bits mit den Gewichten 8,4,2 und 1
- Exakte Ergebnisse bezüglich Dezimalzahlen
 - $0,10_{10}$ in Binärdarstellung: $0,0001100110011..._2$



Datentypen

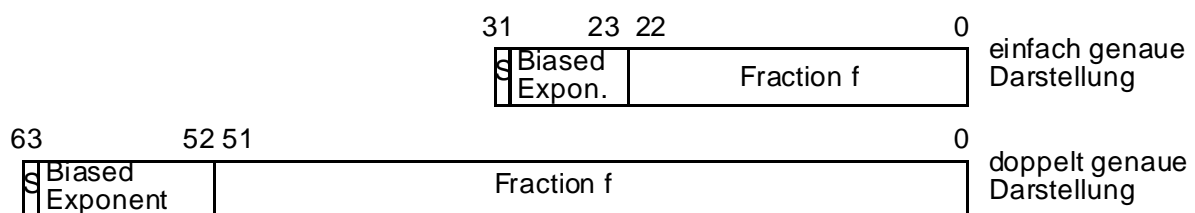
□ BCD-Ziffern in ungepackter Form

- 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

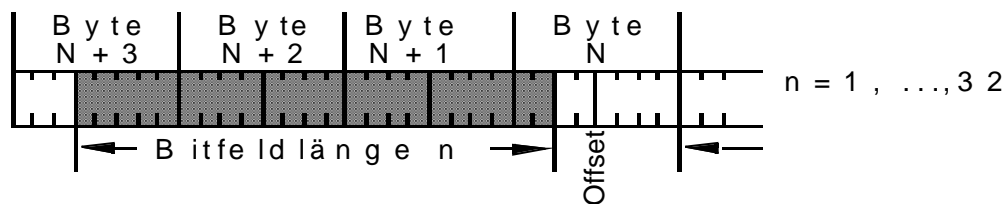
□ Gleitkommaformat



Datentypen

□ Bitfeld

- Darstellung und Verarbeitung von Bitvektoren, vorzeichenlose 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 folgend gespeicherten Bytes, Halbwörter oder Wörter
- *Byte-Strings*: bestehend aus ASCII Zeichen;
- *Byte-, Halbwort- oder Wort-Strings*: mit Operanden 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 Gleitkommadaten
- Byte , Half Word und Word Daten werden in GPRS geladen mit Null oder Vorzeichenerweiterung



Speicheradressierung

□ Datenzugriff

➤ Byteadressierbarer Speicher

- direkt zugreifbar im Speicher ist das Byte, Halbwort oder das Wort, wobei sich die Adressen, unabhängig vom Datenformat auf Bytegrenzen beziehen (Byteadressen);

➤ Wortorganisierter 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 (halbwortorganisiert) oder 8 Bit (byteorganisiert) reduziert sein;



Speicheradressierung

□ Datenzugriff

➤ Ausrichten der Daten im Speicher (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 $A \bmod s = 0$ ist;
- ein Speicherzugriff pro Operand



Speicheradressierung

□ Datenzugriff

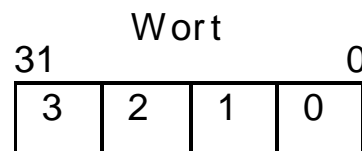
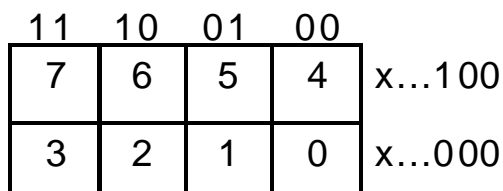
- Nicht ausgerichtete Daten (data misalignment)
 - Speicherung von Operanden in den Datenformaten Halbwort, Wort, Doppelwort etc. an beliebigen Byteadressen;
 - nicht bei allen Prozessoren erlaubt;
 - Vorteil:
 - lückenlose Nutzung des Speichers bei beliebiger Mischung der Datenformate;
 - Nachteil:
 - 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 R10000:
 - schreibt das Ausrichten der Daten auf vor;



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



- Problem:
 - Vergleich von Strings
 - » SDRAWKCAB (backwards)



Speicheradressierung

□ Anordnung der Daten im Speicher

➤ 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;

| | | | |
|----|----|----|----|
| 11 | 10 | 01 | 00 |
| 4 | 5 | 6 | 7 |
| 0 | 1 | 2 | 3 |

x...100
x...000

| | | | |
|------|---|---|---|
| Wort | | | |
| 31 | | | 0 |
| 3 | 2 | 1 | 0 |



Speicheradressierung

Woher der Name:

Jonathan Swift: Gulliver's Reisen (1726):

„Gulliver gelangt in das Land Liliput, dessen Einwohner nur 6 Zoll groß sind. Dort war es verboten, ein Ei an der stumpfen Seite zu öffnen, weil sich der Prinz des Landes dabei in den Finger geschnitten hatte. Ein Teil der Bevölkerung, die BIGENDIANS, wollte das nicht hinnehmen, es gab 6 Rebellionen, ein Kaiser starb, einer musste gehen. Die BIGENDIANS wurden aus dem öffentlichen Dienst ausgeschlossen, 100 BIGENDIAN-Bücher verboten. Viele BIGENDIANS retteten sich in das benachbarte Blefescu. Es kam zum Krieg, der 3 Jahre dauerte und Liliput 30000 Soldaten, vierzig große und noch mehr kleinere Schiffe kostete. Die Invasion von Liliput konnte nur dadurch verhindert werden, dass Gulliver („der Menschenberg“) die feindliche Flotte raubte....“



Speicheradressierung

□ Adressierungsarten

- Spezifikation der Adresse eines Objekts (Konstante, Register, Speicherzelle);
- neben der Angabe der Speicheradresse direkt im Befehlswort, sehen Prozessoren verschiedene Möglichkeiten der Adressmodifikationen vor;
- Berechnung der effektiven Adresse (tatsächliche Adresse) aus mehreren Teilen, die im Befehlswort, in Registern oder Speicherzellen stehen, während der Befehlsausführung (Adressrechnung zur Laufzeit, *dynamische Adressrechnung*)



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

- Programmadresse:
 - In Befehlen und als Adresswerte im Programm vorliegende Adressen
 - Nach Vorgaben des Programms erzeugt der Prozessor aus Programmadressen Prozessadressen
 - Indexmodifikation
 - Substitution,
 - (Befehlszähler)relativer Adressierung
 - „offener“ Basisadressierung



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

- Prozessadresse (effektive Adresse):
 - Verwendet der Prozessor
 - Nach Vorgaben des Betriebssystems erzeugt der Prozessor aus Prozessadressen Maschinenadressen
 - „verdeckte“ Basisadressierung
 - Seitenadressierung
 - Hauptziel:
 - Beliebige Lage des Programms und seiner Werte
 - partielle Lagerung im Speicher



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

- Maschinenadresse
 - Verwendet der Prozessor gegenüber Hauptspeicher

