

# Inhaltsverzeichnis

## 1 Arithmetik 5

- 1.1 Stellenwertsysteme 5
  - Surjektivität (6), Injektivität (7)
- 1.2 Moduloarithmetik und Dualzahlen 9
- 1.3 Die Addition 10
- 1.4 Ganze Zahlen 12
  - Bereichserweiterung (13), Überlauf (14)
- 1.5 Shifts 15
- 1.6 Gleitkommazahlen 17
- 1.7 Zusammenfassung 18
- 1.8 Aufgaben 19

## 2 Maschinensprache 31

- 2.1 Der von-Neumann Rechner 32
- 2.2 Der MIPS-Prozessor 32
- 2.3 Der Assembler 35
  - Adressierung (38), Das Datensegment (39), Aufruf des Betriebssystems (41)
- 2.4 Zusammengesetzte Datentypen 42
  - Reihungen (42), Verbunde (44), Ausrichtung (engl. alignment) (44), Speicherallokation (46)
- 2.5 Das Laden des Programms und die Speicheraufteilung 48
  - Speicherverwaltung auf der Halde (48)
- 2.6 Der Linker 49
- 2.7 Aufruf von Unterprogrammen 50
  - Verwendung der Registerbank (52)
- 2.8 Beispiele 53
- 2.9 Aufgaben 62

### 3 Einführung in C 85

- 3.1 Ein Beispiel 87
- 3.2 Übersetzungseinheiten 88
  - main (88), Unterprogramme aus anderen Übersetzungseinheiten aufrufen (89), Makefiles (91)
- 3.3 Variablen und Behälter 92
  - Sichtbarkeitsbereich (92), Lebensdauer (92)
- 3.4 Basistypen 93
- 3.5 Ausdrücke 94
  - Automatische Typanpassung (94), L/R-Auswertung (95), Nebenwirkungen (95), Faule Auswertung (97)
- 3.6 Aufrufen von Unterprogrammen 98
- 3.7 Reihungen und Zeiger 99
  - Zeiger (100), Zeigerarithmetik (101), const (101)
- 3.8 Dynamische Speicherverwaltung 103
  - realloc (104)
- 3.9 Verbunde 105
  - typedef (106), Unvollständige Verbunde (107)
- 3.10 Ein- und Ausgabe 109
  - Formatstrings (109)
- 3.11 undefiniertes Verhalten 112
  - Unspezifiziertes und implementierungsspezifisches Verhalten (114), Sicherheitsprobleme (115), Übersetzer (117), Beispiele aus der Praxis (118)
- 3.12 Beispiele 122
- 3.13 Aufgaben 123

### 4 Fehlersuche 139

- 4.1 Grundlagen 139
- 4.2 Funktionale Korrektheit 142
- 4.3 Fehler 143
- 4.4 Testen 148
  - Das Orakelproblem (149), Funktionsbasiertes (black-box) Testen (150), Abdeckung (151), Fuzzing (153)
- 4.5 Aufgaben 154

### 5 Einführung in Java 163

- 5.1 Übersetzung, Hauptprogramm, Pakete 163
  - Mehrere Klassen (164), Pakete (165)
- 5.2 Typen 167
- 5.3 Referenztypen 168
  - Reihungen (168)

## 6 Objektorientierte Programmierung 171

- 6.1 Klassen 172
  - Konstruktoren (173)
- 6.2 Kapselung 175
- 6.3 Referenzen, Aliasing und unveränderliche Objekte 178
  - Aliase (178), Unveränderliche Klassen (179), Wann unveränderliche Klassen? (180)
- 6.4 Vererbung 181
  - Schnittstellen (181), Überschreiben von Methoden (183), Der Methodenaufruf (185), Die Klasse `Object` (186), `equals` (186), `hashCode` (187), `toString` (188)
- 6.5 Überladen von Methoden 189
  - Überladen und Überschreiben (190)
- 6.6 Exkurs: Objektorientierte Programmierung in C 192
  - Der `void`-Zeiger (192), Klassen und Methoden (192), Vererbung (195)

## 7 Einfache Datenstrukturen 197

- 7.1 Listen 197
- 7.2 Die Reihungsliste 199
  - `set` und `get` (199), `add` (200), `remove` (201), `contains` (201), Iteration (202), Der Reihungslisten-Iterator (202)
- 7.3 Mengen 204
  - Indexleisten und Bitvektoren (204)
- 7.4 Kollisionslisten 207
  - Lastfaktor (208), Hashtabellen und veränderliche Objekte (208)
- 7.5 Hashfunktionen 210
  - Eine einfache Hashfunktion (211), Hashen von Reihungen (213), Rechnen im Restklassenring  $\mathbb{Z}/p\mathbb{Z}$  (214)
- 7.6 Sondieren 217
  - Lineares Sondieren (218), Quadratisches Sondieren (218), Hashing im Vergleich (219)

## 8 Entwurfsmuster 221

- 8.1 Adapter 222
  - Delegation (222), Vererbung (222), Diskussion (223)
- 8.2 Abstrakte Fabrik 224
- 8.3 Kompositum 226
- 8.4 Besucher 227

## 9 Ein einfacher C(o)-Übersetzer 233

- 9.1 Abstrakte Syntax von C0 234
  - Beispiel (234), Bezeichner (235), Typen (235), Operatoren (235), Ausdrücke (235), Anweisungen (235), Übersetzungseinheit (235)

## 9.2 Statische Semantik 237

Ausdrücke (237), Anweisungen (239), Übersetzungseinheit (241)

## 9.3 Elaborierung 242

## 9.4 Syntaxgesteuerte Code-Erzeugung 246

Anweisungen (246), Ausdrücke (248), Nichtdeterminismus bei der Code-Erzeugung binärer Ausdrücke (251), Funktionsdefinition (252)

# 10 Korrekte Programme 253

## 10.1 Hoare-Logik 253

Verstärken und Abschwächen (257)

## 10.2 Schwächste Vorbedingungen 257

Weitere Eigenschaften (259), Einfache Anweisungen (259)

## 10.3 Schleifen 262

Schleifeninvarianten (263), Terminierung (264), Zusammenfassung (265)

## 10.4 Beispiele 267

Fakultät (267), Binäre Suche (268)

# A MIPS Assembler Kurzreferenz 273

## A.1 Systemaufrufe 273

## A.2 Assemblerdirektiven 273

## A.3 Befehle 274

# B ASCII-Tabelle 275

# C Index 277

# 1

## Arithmetik

Die meisten Rechner arbeiten digital (von lat. *digitus* = der Finger). Das heißt, dass eine Speicherzelle eine endliche Anzahl von Zuständen annehmen kann. In der Praxis ist eine Speicherzelle **binär**, das heißt, dass sie nur zwei Zustände annehmen kann: Spannung vorhanden, Spannung nicht vorhanden; 0 oder 1. In diesem Kapitel besprechen wir, wie man ganze Zahlen mithilfe von **Bits** repräsentiert und wie man die gewohnten Rechenoperationen (Addition, Subtraktion) durch Bitoperationen ausdrückt.

Wir diskutieren zuerst, wie man mittels **Stellenwertsystemen** die natürlichen Zahlen durch Folgen von Ziffern (aus einem endlichen Ziffernvorrat) darstellt. Im täglichen Leben verwenden wir zehn Ziffern, der Rechner verwendet zwei. Beim Programmieren ist es manchmal praktisch, sechzehn oder nur acht Ziffern zu verwenden. Eine binäre Ziffer nennt man **Bit**. Mit einer Folge von  $n$  Bits kann man dann  $2^n$  Zahlen darstellen. Aus praktischen Gründen rechnet ein Prozessor immer mit Bitfolgen fest vorgegebener Länge. Dies rührt daher, dass die Schaltkreise, die die Bitfolgen verarbeiten eine feste Größe haben und sich nicht ändern lassen. Daher müssen wir verstehen, wie wir die gewohnten arithmetischen Operationen sinnvoll auf diese Zahlen fester Länge anpassen. Dies erreichen wir durch die **Moduloarithmetik**.

### 1.1 Stellenwertsysteme

Zahlen schreibt man gewöhnlich als Folge von **Ziffern**. Dadurch, dass man nur einen endlichen Vorrat von Ziffern hat, muss man Ziffern zu Folgen zusammensetzen, um jede natürliche Zahl darstellen zu können. Da es auf die Gestalt der Ziffer nicht ankommt, sondern allein entscheidend ist, dass man nur endlich viele Ziffern hat, wollen wir im folgenden die Menge der Ziffern mit der Menge  $\{0, \dots, \beta - 1\}$  gleichsetzen, wobei  $\beta > 0$  die Anzahl der vorhandenen Ziffern ist und **Basis** des Stellen-