Vorlesungsmodul Grundlagen der Informatik
- VorlMod GrdlgInf -

Matthias Ansorg

04. Oktober 2001 bis 8. Mai 2002

Zusammenfassung

Studentische Mitschrift zur Vorlesung »Grundlagen der Informatik« bei Prof. Schmitt (Wintersemester 2001/2001) im Studiengang Informatik an der FH Gießen-Friedberg. Die inhaltliche Gliederung dieses Mitschriebes richtet sich streng nach dem offiziellen, von Prof. Schmitt ausgeteilten Skript der Vorlesung [3] und ist als dessen Ergänzung zu verstehen; so enthält diese Mitschrift u.a. den »handschriftlichen Teil« und die Lösungen der offiziellen, stets gleichen Übungen [4]! Diese Mitschrift kann das Vorlesungsskript jedoch noch nicht ersetzen (es fehlen die Kapitel »Technische Grundlagen: Rechnerinterne Informationsdarstellung und Zahlensysteme: Informationsdarstellung im Rechner« und »Technische Grundlagen: Betriebssysteme« und »Anhang«, darüberhinaus viele Details und Grafiken); für eine Klausurvorbereitung sind nötig: diese Mitschrift, das offizielle Skript [3], die offiziellen Übungsblätter [4] und Klausuren aus der Fachschaft. Der Besuch der Vorlesung erübrigt sich.[1]

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit: `http://homepages.fh-giessen.de/~hg12117/index.html`. Wenn sie vollständig ist, kann sie auch über die Skriptsammlung der Fachschaft Informatik der FH Gießen-Friedberg `http://www.fh-giessen.de/FACHSCHAFT/Informatik/cgi-bin/navi01.cgi?skripte` downgeloadet werden.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieses Scriptes liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der verwendeten Quellen zu beachten.
- **Korrekturen:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg, ansis@gmx.de.
- **Format:** Die vorliegende Mitschrift wurde mit dem Programm LyX (graphisches Frontend zu LaTeX) unter Linux erstellt und als `pdf`-Datei exportiert.
- **Dozent:** Prof. Dr. Wolfgang Schmitt.
- **Verwendete Quellen:** [3], [5], [6].
- **ToDo:** Dieses Skript eignet sich als Hilfe zum Verstehen des Stoffes, aber nicht zum schnellen Nachschlagen in der Klausur. Jemand sollte sich also die Mühe machen, eine wirklich gute Schnellreferenz (mit Verfahren und Fakten, ohne Details, zum schnellen Auffinden) als Hilfe für Klausuren zu schreiben und zum Download zur Verfügung zu stellen.

Inhaltsverzeichnis

[1] Ich habe die Klausur im WS 2001/2002 geschrieben und (mit Gebet ...) gut bestanden, trotz dass ich nur das erste Viertel der Vorlesungen und nur die erste Übung in GrdlgInf besucht habe. Ich habe mich aus o.a. Materialien vorbereitet. Abgesehen vom Schreiben dieses Skriptes war mein Zeitaufwand: 15h Lernen aus den Materialien, 18h Lösen der Übungsaufgaben und alten Klausuren. –Matthias

Teil I
Einleitung

1 Grundbegriffe

Prof. Dr. Wolfgang Schmitt, F106, http://homepages.fh-giessen.de/~hg6421/. Anmeldung zu einer Übung: www.fh-giessen.de -> FB MNI -> Klausuranmeldung -> Passwort studi

1.1 Was ist Informatik?

IuK-Syteme: Informations- und Kommunikationssysteme.

1.2 Information, Bit und Byte

1.3 Kerngebiete der Informatik

1.4 Informatik und Gesellschaft

1.5 Algorithmen und Programme

1.5.1 Algorithmus

Definition »Ein Algorithmus ist ein präzise und eindeutig formuliertes Schema, das unter Verwendung von endlich vielen ausführbaren und effektiven elementaren Arbeitsschritten (Aktionen) zur Lösung einer Klasse gleichartiger Probleme auch von einer Maschine schrittweise abgearbeitet werden kann. Den Vorgang der Abarbeitung bezeichnet man als Prozess, die ausführende Maschine als Prozessor«[1]. Diese Definition ist anwendbar für die Programmierung, aber nicht genügend für die theoretische Informatik.

Beschreibung Ein Algorithmus meint also ein Verfahren, mit dem man immer zu einer Lösung kommt. Hier handelt es sich um Algorithmen, die von einer Maschine verarbeitet werden, nicht vom Menschen, der im Gegensatz dazu ein recht fehlertolerantes System ist.

Elementare Arbeitsschritte meint die kleinsten Arbeitsschritte, die eine Maschine ausführen kann; beim Computer ist dies die Ebene einzelner Bits.

Algorithmen sollen auch von Maschinen ausführbar sein: früher wurde Algorithmen auch von Menschen ausgeführt, z.B. zur Berechnung der Nullstellen von Polynomen. Algorithmen sind also im Gegensatz zu Programmen unabhängig vom ausführenden Prozessor; sie haben deshalb eine sehr hohe Lebensdauer (Gauß-Algorithmus etc.).

Arten von Algorithmen

- imperative Algorithmen: bestehend aus einer Folge von Befehlen; darauf basieren alle heutigen Programme, weil die heutige Hardware auf der von-Neumann-Architektur basiert. Bis 1980 war man überzeugt, dass diese Algorithmen von den folgenden beiden Typen abgelöst werden; dies hat im Wesentlichen nicht stattgefunden.
- funktionale Algorithmen: wichtig bei der künstlichen Intelligenz.
- Algorithmen auf Basis der Prädikatenlogik: wichtig bei der künstlichen Intelligenz.

Tabelle 1 stellt die möglichen, in Flussdiagrammen verwendbaren Symbole dar. Flussdiagramme sind eine mögliche Art der Darstellung von Algorithmen.

1.5.2 Programme

Algorithmen können in PseudoCode beschrieben werden (eine sehr stark formalisierte Umgangssprache, z.B. structured english). PseudoCode ist jedoch nicht ausführbar. Ein Programm dagegen ist eine präzise Formulierung eines Algorithmus in einer für den Rechner interpretierbaren künstlichen Sprache. Eine solche Sprache heißt Programmiersprache. Den Prozess der Umsetzung einer algorithmischen Idee in einer Programmiersprache heißt Programmierung.

Es ist sehr zu empfehlen und auch heute noch das Vorgehen in der Industrie, vor der Codierung den Algorithmus beschreibend zu formulieren: erst den Algorithmus schreiben, dann die Umsetzung im Programm.

2 Anwendungsgebiete der Informatik

- kommerzielle, verwaltungstechnische Aufgaben
- Textverarbeitung, Tabellenkalkulation, Grafik
- Datenbanken; (intelligente) Informationssysteme, d.h. Expertensysteme z.B. unter Verwendung von KI-Methoden zur Diagnose- und Therapieerstellung bei Krankheiten.
- technische und wissenschaftliche Berechnungen. Numerisches und symbolisches Rechnen.
- Steuern und Regeln, Prozessautomatisierung.

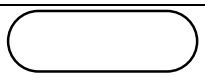
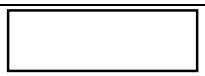
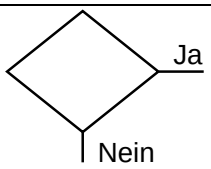
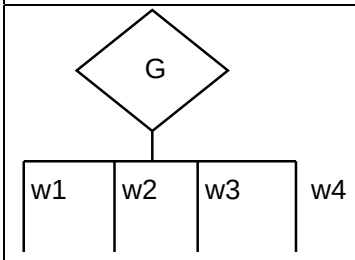
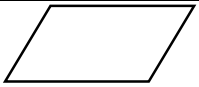
	Start oder Ende des Flussdiagramms. Dieses Symbol enthält »Start«, »Ende« oder eine Beschreibung des ersten bzw. letzten Arbeitsschrittes.
	Prozessschritt, Aktivität: Eine Anweisung
	Verzweigung
	Mehrfachverzweigung
	Dokument
	Daten
	Diese Anweisung (Aktivität oder Prozesschritt) hat ein eigenes Flussdiagramm
	Flussrichtung; verbindet die einzelnen Symbole.
	Fortsetzung des Flussdiagramms bei einem ebensolchen Symbol mit der gleichen Nummer darin.

Tabelle 1: Symbole von Flussdiagrammen

- Rechen- und Kommunikationssysteme. Netzwerke, z.B. Internet.
- Aufgaben in der Medizin und im Gesundheitswesen.

3 Berufsbilder

Für Informatiker sind Berufsbilder und Spezialistenprofile für folgende Tätigkeitsfelder definiert worden:

3.1 Systemanalyse

- Systemanalyse. IT-Systemanalytiker. Die eigentliche Programmierung ist kostenmäßig der geringste Anteil bei der Realisierung eines DV-Projektes. Aufgaben der Systemanalyse vorher:
  - Anforderungsanalyse: das Problem analysieren um herauszufinden, was der Kunde eigentlich will.
  - Entwurferstellung: wie zerlege ich ein Gesamtsystem in sinnvolle Teilkomponenten.
  - Kostenanalyse vor Auftragsbeginn.
  - Testplan vor Codierung
- Softwareentwickler (System- oder Anwendungssoftware).
- Datenbankentwickler
- Nutzerschnittstellenentwickler. Dies ist schwieriger als man meint; man muss sich an genau defineirte Regeln alten. Dazu passt die Vorlesung »Mensch-Maschine-Schnittstelle«.
- Multimediaentwickler. Verarbeitung und Darstellung von meherern Medien auf dem PC.
- IT-Systemplaner
- Netzplaner. Plant die Vernetzung von Rechnern.
- Sicherheitsplaner. Dies ist ein neues Berufsbild, das hauptsächlich durch die globale Vernetzung entstand. Man versucht, Sicherheitsstrategien gegen Angriffe von außen zu entwickeln. Passende Vorlesungen:
  - Datenschutz und Datensicherheit
  - Prof. Ecker: »Kryptologie«
  - Prof. Schmitt: »Sicherheit in der Informationsverarbeitung«

3.2 Koordination

- IT-Projektkoordinator. Es passiert oft, dass IT-Subprojekte in andere Systeme integriert werden müssen, z.B. Steuerungen in Maschinen. Die Übereinstimmung der Teilprojekte muss koordiniert werden.
- IT-Produktkoordinator. Entwicklung eines einheitlichen Benutzerinterfaces für die verschiedenen Produkte einer Firma, um eine Produktfamilie zu erhalten.
- IT-Konfigurationskoordinator
- IT-Sicherheitskoordinator
- IT-Qualitätssicherungskoordinator
- Dokumentationsentwickler. Das Schreiben der Dokumentation zu einem Produkt wird im Allgemeinen unterschätzt. Es ist ein wesentlicher Bestandteil bei der Softwareproduktion.

3.3 Kundenbetreuung

- IT-Vertriebsbeauftragter
- IT-Kundenbetreuer
- IT-Trainer. Heute ein wichtiges und dynamisches Feld, in dem viele Informatiker arbeiten.

3.4 Administration

- Netzwerkadministrator. Diese Aufgabe der Wartung eines bestehenden Netzes ist bei großen Netzen anspruchsvoll.
- Datenbankadministrator
- Web-Administrator
- IT-Systemadministrator
- IT-Anwendungsadministrator

3.5 Lösungsentwicklung/-beratung

- Anwendungssystemberater
- E-Marketingentwickler
- E-Logistikentwickler. Bei Internethandel fällt eine riesige Menge an materieller Logistik an, die hier bewältigt werden muss.
- IT-Wissensorganisationsentwickler. Wissen ist sinnvoll nutzbare Organisation.

4 Geschichte der Informatik

Siehe [2].

Teil II
Technische Grundlagen

5 Einführung in die Rechnertechnik

5.1 Geschichte der Rechnerentwicklung

5.1.1 Rechnergenerationen

Der www-Dienst, entwickelt in Cern in der Schweiz, machte das Internet erst für Normalverbraucher bedienbar. Es war seit seiner Trennung vom militärischen ARPANET Forschungsgegenstand, ist heute eine weitgehend kommerzialisierte und wichtige Infrastruktur.

HP hatte um 1970 den ersten Taschenrechner auf den Markt gebracht, er kostete 1800 DM.

5.1.2 Mikroprozessorgenerationen

Erst die Möglichkeit, Schaltungen zu integrieren (mehere Transistoren in einem Schaltkreis) statt diskret, d.i. aus Einzelbauteilen aufzubauen, gab die Möglichkeit zur Produktion von Mikroprozessoren, das sind Prozessoren auf einem Chip.

Die Entwicklung der 16bit 80x86-Prozessorfamilie begann 1979 mit dem Intel 8086. 1982 erschien der 80286, 1988 der 80386, 1994 der Intel Pentium.

Der Transistor wurde 1948 in den Bell Laboratories entwickelt, den berühmte Entwicklungslabors der damaligen Firma AT&T.

Das Gesetz von Moore, dass sich die Komplexität von ICs jedes Jahr verdoppeln wird, wurde von ihm 1964 aufgestellt!

Die Klausur wird keine speziellen Fragen zu Persönlichkeiten der Computergeschichte enthalten.

5.2 Aufbau eines Rechnersystems

5.2.1 Bausteine eines Rechnersystems

1. Rechner
   (a) CPU
   (b) Controller. Dies ist ein »Steuerbaustein«, keine »Kontrollierbaustein«.
   (c) Programme
2. Eingabegeräte. Hauptsächlich Tastatur und Maus, früher noch Lochkarten- und Lochstreifenlesegeräte. Außerdem Scanner.
3. Ausgabegeräte. Bildschirm und Drucker. Spezialgeräte: CD-Brenner, Lochstreifenschreiber. Die Bildschirmtechnik basiert auf Kathodenstrahltechnik wie im Fernseher (durch elektrische Felder abgelenkte Elektronenstrahlen) oder auf LCD-Technik.
4. Massenspeicher. Dazu zählen wechselbare Speichermedien wie Diskette, CD und Magnetbänder.
5. Netzwerkverbindungen. Vernetzung ist typisch für heutige Rechner. LAN: »local area network«.

5.2.2 Hardwarekomponenten eines PCs

An den Mikroprozessor (μP, die CPU) sind angeschlossen:

- RAM
- Controller. Diese ermöglichen die Anpassung beliebiger Hardware an einen Mikroprozessor, so dass sich eine Komponententechnologie ergibt.
- Grafikadapter
- parallele Schnittstelle bzw. »parallel interface (IF)«. Bei einer parallelen Übertragung werden mehrere Bit gleichzeitig übertragen, eine entsprechende Anzahl paralleler Leitungen ist dazu Voraussetzung. Die Reichweite paralleler Schnittstellenleitungen liegt unter 10m.
- serielle Schnittstelle. Es kann nur ein Bit gleichzeitig bertragen werden. In einem Schieberegister (LIFO-Speicher) werden während des Empfangs so lange Bits zwischengespeichert, bis dem Prozessor vollständige Bytes übergeben werden können. Parallele Schnittstellen sind für große Entfernungen ungeeignet, da die Verkabelung zu teuer wäre. Dazu werden serielle Schnittstellen verwendet, trotz der langsameren Übertragungsgeschwindigkeit. Die normale serielle Schnittstelle ist nach RSC232 bzw. V24 genormt, außerdem gibt es noch USB, was die serielle Schnittstelle in absehbarer Zeit ersetzen wird. Zur Normung einer Schnittstelle gehört die physikalische, elektrotechnische Normung.
- Netzwerkadapter. Modems (MODulator-DEModulator) sind Geräte zur Übertragung digitaler Daten über analoge (Telefon-)Netze. Die Notwendigkeit, im Telefonnetz auch Datenübertragungsverbindungen nach Zeit zu tarifieren, liegt an der Optimierung des Telefonnetzes für Sprachübertragung: Ressourcen werden nach Zeit, nicht nach Datenmenge belegt. Es gibt dagegen auch auf Datenübertragung optimierte Netze, z.B. LANs, WANs. LANs wiederum sind ungeeignet für Weitverkehrsverbindungen und Sprachübertragung.

5.2.3 Systembus

Ein Systembus besteht aus einigen parallelen Leitungen (was bitparellele Übertragung ermöglicht), die die Busteilnehmer (CPU, RAM, Festplatte usw.) miteinander verbindet. Ein Busverwalter (ein Mikroprozessor) legt fest, welcher Teilnehmer wann senden darf. In einem Bus gibt es eine Normung der Zeit durch eine Taktung (siehe Abbildung im Skript [3] zu »Aufbau eines Rechnersystems: Systembus - Was ist das?«). Der Busverwalter (auch: Arbiter) legt nach einem Busprotokoll fest, wer wann senden darf.

Bussysteme sind lineare Leitungen, die nicht zu einem Ring geschlossen sind. Ein Ring ist auch eine mögliche Kommunikationsstruktur, aber auf Systemebene nicht zu verwirklichen.

Es gibt Baugruppen, die den Bus selbständig belegen können und einen Datenaustausch durchführen, dies sind sog. Busmaster: Baugruppen, die »eine gewissen Intelligenz haben«. Sie haben eine Talker-Funktion,

d.h. sie können senden. Beispiele sind Mikroprozessoren und Netzwerkkarten. Nur bei mehreren Busmastern braucht man einen Arbiter, um die Sendereihenfolge festzulegen. Ein Adressat wird als Slave bezeichnet. Die Rolle von Master und Slave kann wechseln, z.B. in Mehrprozessorsystemen.

Die Busleitungen (z.B. eines ISA- oder PCI-Systembusses) bestehen nicht nur aus Datenleitungen, sondern aus Datenbus, Adressbus, Steuerbus und Versorgungsbus (Takt- und Spannungsversorgung), denn zu einem Kommuniktionsablauf gehören neben der eigentlichen Datenübertragung (als Busmaster oder -slave) Interrupt-Handling und Bushandling.

Die Busslots in einem PC sind Zusatzanschlüsse des Busses, in die beliebige zusätzliche E/A-Module eingesteckt werden können, wie z.B. Netzwerkkarten.

Ladezyklus einer Variablen aus dem RAM: die CPU schreibt die Adresse, deren Wert er aus dem RAM anfordert, auf den Adressbus; der RAM schreibt den Wert auf den Datenbus, wo ihn der Prozessor abholt. In der von-Neumann-Architektur werden auch Programmteile über den Datenbus gelesen.

Aufgaben der Teilbusse Im Skript [3] in der Abbildung »Aufbau eines Rechnersystems: Systembus - Datenflüsse« bedeuten schwarze Pfeile Steuersignale, und weiße Pfeile Daten oder Adressen.

Steuerbus Er implementiert die Steuerungsfunktionen zur Realisierung der genannten Busfunktionen. Er unterteilt sich in Leitungsgruppen für die Steuerung von: Busverwaltung, Interruptverwaltung, Datenübertragung.

Adressbus Um Daten an bestimmte Stellen des RAM transportieren oder von bestimmten Stellen im RAM holen zu können, müssen die Speicherbereiche adressiert sein. Der Adressbus transportiert diese Adrsessen.

Datenbus Die Busbreite gibt die Zahl der Leitungen an, die gleichzeitig zum Datentransport verwendet werden können. Gängige Busbreiten sind 8, 16, 32 der 64 Bit, also immer $2^n \cdot 8Bit = 2^n Byte$. In einem Taktzyklus können bei einer Busbreite von 64 Bit also gleichzeitig 64 Bit transportiert werden. Die Daten stellen in der von-Neumann-Architektur Programmteile oder Programmdaten dar.

5.2.4 Softwarekomponenten eines PCs

Betriebssysteme Jeder Computer benötigt ein Betriebssystem, das alle folgenden Softwarekomponenten verwaltet. Es gibt neben den Windows- und *iX-Systemen auch Betriebssysteme für Chipkarten und solche für Großrechner mit realtime-Anwendungen usw.

Anwendungsprogramme Sie sollen Probleme aus der Anwendungswelt lösen. Beim Programmieren von Anwendungsprogrammen darf man beim Benutzer keinerlei Kenntnisse der Informatik voraussetzen.

Systemprogramme Sie sollen dem Benutzer ermöglichen, den Rechner zu verwalten. Beispiele: Taskmanager, Auslastungsanzeiger, log-Programme.

Treiber Die unterste Ebene über dem Betriebssystem. Sie sind eine Schnittstelle zwischen Hardwarekomponenten und Softwarekomponenten.

6 Das Grundkonzept der von-Neumann-Architektur

6.1 Historie

Umfassende Darstellung siehe das offizielle Vorlesungsskript [3] im Kapitel »Technische Grundlagen: Das Grundkonzept der von-Neumann-Architektur: Historie«.

Johann von Neumann definierte als erster die prinzipielle Arbeitsweise einer universellen, speicherprogrammierbaren Rechenmaschine. Universell heißt: sie ist nicht nur zur Lösung eines Problems, sondern universell zur Lösung von Problemen geeignet. Soeicherprogrammierbar heißt: der Algorithmus zur Lösung eines Problems wird in den Speicher geladen, um aktiviert zu werden; er kann ausgewechselt werden und ist nicht fest verdrahtet, was natürlich vorteilhaft ist.

Der allergrößte Teil der heutigen Computer basieren auf einer Weiterentwicklung der von-Neumann-Architektur.

6.2 Konstruktionsprinzipien

6.2.1 Funktionseinheiten

- Steuerwerk (auch: Leitwerk). Es organisiert den Datenfluss über den Verlauf der Zeit.
- Rechenwerk. Führt die eigentlichen Berechnungen durch.
- Speicher. Dient der Aufnahme von Daten, Algorithmen und den Ergebnissen der Berechnung.
- Eingabewerk.
- Ausgabewerk.

Die Zusammenfassung von Steuerwerk und Rechenwerk bildet den Prozessor, bei Unterbringung in einem Chip den Mikroprozessor.

6.2.2 Programme

Bei der von-Neumann-Architektur hängt die Rechnerstruktur nicht von der Problemstellung ab, sondern ist universell. Im Gegensatz dazu gibt es auch heute noch Spezielrechner, die z.B. darauf optimiert sind, Produkte von Zahlen aufzuaddieren (Signalprozessoren). Von Neumanns Idee war, eine auswechselbare Bearbeitungsvorschrift (einen Algorithmus) zu formulieren und diesen in einer für den Prozessor verständlichen Sprache (d.h. als Programm) im Speicher abzulegen. Weil hier Befehle gegeben werden, verlangt die von-Neumann-Architektur einen imperativen Programmierstil. Im Gegensatz dazu wäre auch ein funktionaler Programmierstil unter Verwendung mathematischer Gleichungen möglich (wo Zuweisungen $a = a + b$ nicht existieren, da mathematisch nicht korrekt).

6.2.3 Arbeitsspeicher

Sowohl das Programm als auch alle Daten, Zwischenergebnisse und Ergebnisse werden in einem einzigen Arbeitsspeicher abgelegt (früher rotierende Trommelspeicher). Heute wurde dies weiterentwickelt zu einer Speicherhierachie, wo es kleine schnelle und große langsame Speicher gibt. Solch eine Hierchie wurde aus Kostengründen entwicklelt und auch aus Gründen der Arbeitsorganisation: ein Cache enthält nur die gerade zur Bearbeitung nötigen Daten (working set). Es gibt flüchtige und nichtflüchtige Speicher.

Der Speicher ist gegliedert in gleich große Speicherzellen (im RAM typischerweise 1 Byte), die über den Adressbus adressiert und über den Datenbus gelesen und geschrieben werden können.

6.2.4 Befehle

Befehle einer Sequenz werden typischerweise auch sequentiell im Arbeitsspeicher abgelegt. Die Adressierung des nächsten Befehls geschieht dann durch Erhöhen des Befehlszählers im Steuerwerk um eine Einheit. Neben Programmsequenzen gibt es aber auch Bedingungen (bedingte Sprünge) und Schleifen. In solchen Fällen muss der Befehlszähler nicht um eine Einheit erhöht, sondern auf die Sprungadresse gesetzt werden.

6.2.5 Befehlsformat

Ein Befehl enthält:

- den Opcode, der die auszuführende Operation codiert, z.B. Addition, Multiplikation usw.
- Adressen der Operanden und die Adressierungsart, sofern der Befehl überhaupt Operanden braucht. Die Adresse kann absolut sein oder relativ; relative Adressen beziehen sich auf einen Startpunkt im RAM, an den das Programm geladen wurde.
- eine Adresse, auf die das Ergebnis der Operation geschrieben werden soll.

Beispiel 1

```
MOV AX, 0x2F //80x86 Assembler
```

Der Befehl meint semantisch: Lade das (allgemeine) Register `AX` mit dem Wert `0x2F`. `0x2F` ist dabei die Hexadezimaldarstellung der Zahl 47. Sie bezeichet keine Speicheradresse, sondern eine Zahl, und wird deshalb unmittelbarer Operand (immediate) genannt. Der Opcode des Befehls `MOV` ist `0xB8`. Damit ist der Maschinen code dieses Befehls:

```
0xB8 0x2F
```

bzw. in Binärdarstellung:

```
1011 1000 0010 1111
  Opcode | Operand
```

Beispiel 2

```
MOV 0x2F, AX //80x86 Assembler
```

Semantisch bedeutet der Befehl: Hole den Wert, der unter der Adresse `DS:0x2F` im RAM steht und lade ihn in das Register `AX`. Der RAM ist in verschiedene Speicherbereiche unterteilt, nämlich in Reihenfolge in Heap, Stack, Codesegment (CS), Datensegment (DS) und Systemprogramme. Die Adresse `DS:0x2F` ist nun ein indirekter Speicheroperand, der die Speicheradresse relativ zur Grundadresse des Datensegments (die im Register DS gespeichert ist) bezeichnet. `0x2F` ist also ein Offset (oder: Displacement) von der Grundadresse des Datensegments aus.

Der Maschinencode des obigen Befehls ist nun:

```
0xA3 0x2F
```

bzw. in Binärdarstellung:

```
1010 0011 0010 1111
  Opcode | Operand
```

6.2.6 Befehlszyklus (Mikrozyklus)

1. Adressiere den Speicher über den Befehlszähler, d.h. hole den nächsten Befehl. Der Befehlszähler enthält stets die Adresse des nächsten abzuarbeitenden Befehls. Die Übertragung der Adressen geschieht über den Adressbus.
2. Inkrementiere den Befehlszähler um die Anzahl der gelesenen Bytes (ein Befehl kann nämlich unterschiedlich lang sein, je nachdem wieviele Operanden er hat).
3. Führe den Befehl aus.
4. Wiederhole ab Schritt 1.

Diese sog. SISD-Struktur (single instruction, single data) hat den Nachteil, dass sie streng seriell ist. In der Praxis wird daher die Adressierung des nächsten Befehls und die Ausführung des aktuellen Befehls parallelisiert.

6.2.7 Befehlskategorien

Die Menge aller Befehle ist der Befehlsastz eines Mikroprozessors. Sie kann untergliedert werden in folgende Kategorien:

- Datenverarbeitung (Addieren, Multiplizieren, ...)
- Logik (relationale und boolesche Operationen)
- Datentransfer (Daten transferieren zwischen Arbeitsspeicher, Registern und Ausgabeeinheiten). Beispiel: der oben behandelte Befehl MOV.
- Sprünge

  - bedingte Sprünge: z.B. bei if-Anweisungen
  - unbedingte Sprünge: GOTO. Sie sind in der strukturierten Programmierung verboten, weil sie unübersichtlichen Spaghetticode produzieren: der Kontrollfluss im Programm ist dann nicht mehr nachvollziehbar. Es gibt wenige Ausnahmen, wo unbedingte Sprünge benötigt werden (Automatenprogrammierung, Fehlerbehandlung). Auf Maschinencodeebene werden jedoch stets unbedingte Sprünge verwendet.
  - Sprünge mit Rückkehrabsicht: Werden zur Realisierung von Unterprogrammen (Prozeduren) verwendet, die helfen, redundanten Code zu vermeiden: Der Code mehrmals aufgerufener Funktionen muss dann nicht doppelt geschrieben werden, sondern wird doppelt aufgerufen. Der Sprung zu der Speicherstelle, an der der Code dieser Funktion liegt, ist verbunden mit der Rückkehrabsicht zum Hauptprogramm.
    Im Unterschied dazu gibt es inline-Funktionen, d.h. der Compiler ersetzt die Inline-Funktion durch den entsprechenden Sourcecode. Zu gebrauchen ist dies, um bei sehr kurzen Funktionen den Aufwand des Sprungs zum Unterprogramm und zurück zu sparen und also die Geschwindigkeit des Programms zu erhöhen.

- in modernen Prozessoren

  - Stackbefehle. Ein Stack (bzw. Stapel / LIFO-Speicher / Kellerspeicher) ist ein Speicher, der so organisiert ist dass das zuletzt abgelegte Element zuerst wieder entfernt wird. Im Gegensatz zu diesem LIFO-Speicher (last in, first out) gibt es den FIFO-Speicher (first in, first out). Der Stack dient dazu, rekursiv in Unterprogramme absteigen zu können (und die Variablen der aufrufenden Funktion auf den Stack zu legen) und danach wieder zur jeweils aufrufenden Funktion zurückzukehren (und die abgelegten Variablen dann wieder vom Stack zu holen).
    Lebensdauer von Variablen: Die Lebensdauer lokaler Variablen ist auf die Dauer der Ausfürhung der entsprechenden Funktion begrenzt, danach sind sie nicht mehr ansprechbar.
  - Systembefehle (zur Verwaltung des Prozessors selbst).

6.2.8 Datendarstellung

In heutigen Computern werden alle Daten, Befehle und Adressen binär kodiert, denn die verwendeten Schaltelemente unterscheiden nur zwei Zustände. Für die Zukunft ist es vorstellbar, dass Schaltelemente (wie Elektronen auf unterschiedlichen Energieniveaus, wie sie Quantencomputer verwenden sollen) mehr als zwei Zustände unterscheiden und also auch andere Zahlensysteme verwendet werden können.

Die einheitliche binäre Kodierung bedeutet, dass aus dem Inhalt einer Speicherzelle nicht auf die Bedeutung ihres Inhalts (Daten oder Befehle) geschlossen werden kann. Diese Unterscheidung trifft die CPU durch Einrichtungen des Steuerwerks.

Außer der Aufteilung des Hauptspeichers in Segmente, die entweder Daten (Datensegment, DS) oder Code (Codesegment, CS) oder Adressen enthalten, werden in Programmiersprachen Datentypen eingeführt, um die Semantik von Bitmustern festzulegen. Dabei ist jedes Datenobjekt ein Tripel aus Adresse (bzw. Name), Typ und Inhalt. Das Typkonzept in Programmiersprachen ist notwendig, damit der Computer die Daten richtig interpretieren kann, aber auch sehr nützlich zur Modellierung von realen Konzepten mit den Mitteln der Programmiersprache (z.B. die Datentypen Felder und Strukturen (records)).

6.3 Zusammenwirken der Funktionseinheiten

6.3.1 Informationsflüsse

In der entsprechenden Abbildung im Skript [3] stellen die schwarzen Pfeile Steuerrichtungen dar, die weißen Pfeile Datenflüsse.

Die CPU liedert sich hauptächlich in Steuer- und Rechenwerk.

Das Steuerwerk der CPU gliedert sich wiederum in:

- Befehlsregister (ein Speicher)
- Befehlszähler (ein Speicher)
- Adressregister (ein Speicher)

- Ablaufsteuerung
- Befehlsdekoder. Dies ist eine logische Schaltung, die den Opcode eines Befehls interpretiert.

Das Rechenwerk der CPU gliedert sich wiederum in:

- zwei Datenregister. Es enthält die Operanden einer durchzuführenden Operation.
- mehrere Zusatzregister. Das Flagregister enthält z.B. ein Flag, wenn verbotene Operationen wie die Division durch 0 durchgeführt werden sollen. Zusatzregister helfen auch zur Realisierung von Sprungbedingungen.
- die ALU (arithmetic logic unit). Die Vorlesung Netz- und Schaltwerke erklärt ihre grundlegenden Funktionsprinzipien.

6.3.2 Register

Register sind Speicherplätze in der CPU selbst. Sie untergliedern sich in mehrere Typen mit jeweils spezifischen Aufgaben:

Datenregister / Adressregister Datenregister wie AX, BX speichern Daten, Adressregister wie DS, CS speichern Adressen von Daten, die zur Ausführung in der ALU bestimt sind.

Zusatzregister

[. . .]

6.3.3 Steuer- und Rechenwerk

6.3.4 ALU

Der Akkumulator ist ein besonders ausgezeichnetes Datenregister; es enthält vor einer Operation einen Operanden, wie die anderen Datenregister auch, nach der Operation aber das Ergebnis dieser Operation.

6.4 Interrupts

Interrupts dienen dazu, die Abarbeitung eines Programms (aufgrund bestimmter Ereignisse) zu unterbrechen und anstelle davon ein anderes Programm (eine Interruptbehandlungsroutine) abzuarbeiten. Die Adressen der Interruptbehandlungsroutinen stehen in einem definierten Speicherbereich (dem sog. Interruptvektor). Nach deren Ende wird das unterbrochene Programm, wenn möglich, fortgesetzt. Ein Interrupt-Zyklus entspricht damit einer besonderen Form der Unterprogramm-Technik, wobei der Aufruf zum Sprung (d.h. der Interrupt) durch zwei Arten von ausgelöst werden kann:

Software-Interrupts werden ausgelöst durch einen INT-Befehl des Betriebssystems. Ein Beispiel ist die Unterbechnung eines Programms zum Ausdruck des Bildschirms, wenn die Taste »print screen« gedrückt wurde. Der eigentliche Tastendruck impliziert außerdem einen Hardware- Interrupt.

Hardware-Interrupt Jeder Druck auf eine Taste der Tastatur löst einen Hardware-Interrupt aus, d.i. einen elektrischen Impuls, der über einen logischen Baustein, der alle Interrupts verwaltet, der CPU mitgeteilt wird, die dann die entsprechende Interruptbehandlungsroutine ausführt.

Interrupts eignen sich also speziell zur Datenein- und Ausgabe. Weiter sind die sog. exceptions eine bestimmte Form der Interrupts: sie werden durch Fehlersituationen ausgelöst, wie z.B. eine Division durch Null oder einen ungültigen Opcode.

6.5 Befehlsabarbeitung

Interrupts können keinen einzelnen Befehlszyklus unterbrechen, sondern nach der Abarbeitung jedes Befehls wird geprüft, ob ein Interruptsignal anliegt und ggf. der Interrupt abgearbeitet.

6.6 Kritik

Rechner nach der Von-Neumann-Architektur stellen nur sehr elementare Befehle hardwaremäßig zur Verfügung. Dadurch kann sie zur universellen Problemlösung dienen, muss aber aufwendig programmiert werden. Heute werden Klassenbibliotheken verwendet, um den Programmieraufwand vertretbar zu halten.

Der sog. »Von Neumann Flaschenhals« ist das Bussystem, denn hierüber müssen alle zur Befehlsabarbeitung notwendigen Daten verschoben werden. Durch Optimierung des Bussystems und Verwendung schneller Zwischenspeicher (Cache) wird versucht, dieses Problem zu beheben.

Zur Verbesserung der Von-Neumann-Architektur wird gefordert, zur nichtimprativen Programmierung überzugehen (was bisher nicht stattfand) oder eben Daten parallel zu verarbeiten. Probleme bei der Parallelisierung von Datenverarbeitung ist die geforderte Universalität. Einzelne Probleme lassen sich sehr gut parallel lösen (wie bei Signalprozessoren, die nach der Harward-Architektur aufgebaut sind, oder parallel arbeitende Rechner zur Wetterdatenverarbeitung oder Strömungssimulation).

Im PC-Bereich werden jedoch nur bewährte Architekturen verbessert. Auch der Bestand an großen Programmsystemen wird dazu beitragen, dass die von-Neumann-Architektur noch lange als Standard erhalten bleibt.

7 Rechnerinterne Informationsdarstellung und Zahlensysteme

7.1 Informationsdarstellung im Rechner

7.1.1 Informationen, Daten und Kodes

Daten Es sind Informationen, die im Rechner verarbeitet werden sollen und nach eindeutigen Vorschriften verarbeitungsgerecht formuliert sind. Entsprechend den jeweiligen Aufgaben unterscheidet man:

Verarbeitungsdaten

Programmdaten

Steuerdaten Sie sind nach außen nicht sichtbar. Zum Beispiel: das vom Betriebssystem verwaltete und interpretierte Statuswort eines Prozesses (ruht, aktiv, wartet usw.).

Bit, Byte, Maschinenwort Heute bestehen Wörter (die von einem Prozessor adressierbare Bitgruppe) stets aus Vielfachen von 8 Bit. Dies war jedoch in der Geschichte der Informatik nicht immer so. Nach der Länge des Wortes richtet sich die Länge der Datentypen in Programmiersprachen, deshalb werden die Länge von Integern in Bibliotheken teilweise in Worteinheiten definiert, so dass Anpassungen an neue Generationen vopn Mikroprozessoren sehr leicht möglich sind.

Kodes Ein Kode f ist eine Abbildung $f : A \longrightarrow B$; ist diese Abbildung bijektiv[2], so hat der Kode die »Fano-Eigenschaft«. Die Elemente der Menge A sind die Maschinenwörter, die Elemente der Menge B heißen »Kodewörter des Kodes f«. Kode haben in der Praxis unterschiedlichste Aufgaben. Beispiele:

- Verschlüsselung
- Umsetzung realer Daten (wie Buchstaben) auf binäre Daten.
- Abbildung binärer Daten auf Leitungsdaten (elektrische Signale)
- Kompression von Dateien durch Entfernen von Redundanz

Kodes können in Kodetabellen dargestellt werden.

[2]Eine Abbildung $f : A \longrightarrow B$ ist bijekiv, wenn sie sowohl injektiv ist (jedes von f getroffene $b \in B$ ist nur einem $a \in A$ zugeordnet: Eineindeutigkeit) als auch surjektiv ist (jedes $b \in B$ wird von f getroffen).

7.1.2 Kodes für die Zeichendarstellung

ASCII-Zeichensatz ITU (»internationl telecommunication union«, früher CCITT) ist ein wichtiges Standardisierungsgremium der Telekommunikation. Ausgehend vom Internationalen Telegraphenalphabet Nr. 5 entstand der 7-Bit-ASCII-Code. Ein achtes Bit wurde eingespart.

In C sind die Datentypen char und int intern identisch. Allein die Formatierung bei der Ausgabe unterscheidet sich. So hat das ASCII-Zeichen »A«: $A = 1000001_2 = 65_{10} = 41_{16} = 0x41$. Der Vorteil des Hexadezimalsystems ist, dass eine Hexadezimalziffer stets ein Halbbyte codiert.

Da der ASCII-Code aus der Zeit der elektromechanischen Datenübertragung stammt, enthält er im Sinne der zeichenorientierten Protokolle etliche Steuerzeichen, die ursprünglich zur Kontrolle der Datenübertragung (Telegrafie) dienen: CR zum Wagenrücklauf der »ferngesteuerten Schreibmaschine«, LF zum Zeilenvorschub und etliche andere (wie ESC).

IBM-PC-Zeichensatz Der IBM-Zeichensatz ist eine Erweiterung des ASCII-Zeichensatzes, eingeführt mit der PC-Generation IBM PC AT (»advanced technology«). Dabei wurde der ASCII-Zeichensatz auf 8 Bit erweitert und enthielt nun Sonderzeichen, Blockgrafikzeichen usw. Auch der IBM-Zeichensatz hat noch Steuerzeichen, nur werden die auf einem PC nicht sinnvoll zu interpretierenden Steuerzeichen durch gafische Äquivalente dargestellt. In der entsprechenden Code-Tabelle sind die noch vorhandenen Steuerzeichen durch weiße Flächen dargestellt. Der IBM-PC-Zeichensatz ist auch heute noch der im PC-Bereich verwendete Standard-Zeichensatz.

EBCDIC-Zeichensatz Eine weitere Codierung von Zeichen, die auf Großrechnern verwendet wurde. Vergleiche die Codetabelle im Skript [3, Anhang 1].

BTX BTX war der in den 80er Jahren erstellte Vorläufer des Internet. Dabei wurden Grafiken komplett zeichenorientiert dargestellt, wofür eine eigener Zeichensatz verwendet wurde.

Probleme der Zeichenkodes, Unicode Da der Datenraum von 8 Bit für internationale Zeichen nicht ausreichte, entstanden viele Ländervarianten des IBM-PC-Zeichensatzes (mit Doppelbyte-Zeichensätzen für Sprachen mit mehr als 256 Symbolen) und schließlich 1988 der Unicode. Derzeit sind erst etwa 30.000 der 65536 Unicode-Zeichen definiert. Der Unicode beginnt mit dem ASCII-Code, der also seine Positionen behalten hat. Unicode macht die Internationalisierung von Programmen weit einfacher.

7.1.3 Darstellung ganzer Zahlen

7.1.4 Darstellung von Gleitkommazahlen

bias $:= 2^{ne-1} - 1$ bedeutet (ne ist die Zahl der Binärstellen im Exponentenfeld CH): »bias« ist die Hälfte der mit ne Bit unterscheidbaren Kombinationen (d.h. $\frac{1}{2}2^{ne} = 2^{-1} \cdot 2^{ne} = 2^{ne-1}$) weniger 1. Für $ne = 8$ wie im Format `short real` ist bias $:= 2^7 - 1 = 127$ und für $ne = 11$ wie nach IEEE-Format `double` ist bias $:= 2^{10} - 1 = 1023$. Die Addition von »bias« zum Exponenten wirkt also als ein Offset, das alle dargestellten Zahlen um einen Faktor 2^{bias} auf der Zahlengeraden nach rechts und damit vollständig in den positiven Berech verschiebt. Hier wurde also nicht ein Vorzeichenbit zur Darstellung negativer Zahlen reserviert, wie bei Integern. Trotzdem erfolgen alle Rechnungen im positiven Bereich.

Berechnung des Exponenten aus der Charakteristik:

$$\begin{aligned} CH &:= e + bias \\ \Leftrightarrow e &= CH - bias \\ \Rightarrow 2^e &= 2^{CH-bias} \end{aligned}$$

Die Bitmuster $00\ldots0$ und $11\ldots1$ im Feld CH bedeuten nicht die Exponenten $-bias$ bzw. $bias + 1$, sondern sind reserviert für Sonderfälle:

- Wenn $CH = 00\ldots0$:
  - wenn $m = 00\ldots0$, so bedeutet dies den Float-Wert 0
  - wenn $m \neq 00\ldots0$, so liegt eine denormalisierte Zahl vor (d.h. die Mantisse ist keine normierte Zahl, deren Vorkommastelle 1 nicht gespeichert wird, sondern jetzt gibt das erste Bit der Mantisse die Vorkommastelle an).

- Wenn $CH = 11\ldots1$:
  - wenn $m = 00\ldots0$, so bedeutet dies den Float-Wert ∞.
  - wenn $m \neq 00\ldots0$, so bedeutet das »der Wert ist NaN (not a number)«: der enthaltene Wert kann nicht als Zahl dargestellt werden. Das Mantissenfeld ist dann stets $m = 11\ldots1$.

7.1.5 Binär kodierte Dezimalzahlen

7.2 Zahlendarstellung und Zahlenumwandlung

7.2.1 Zahlendarstellung

Stellenwertverfahren

Natürliche Zahlen. Natürliche Zahlen können durch Stellenwertverfahren zu verschiedenen Basen dargestellt werden. Die Anwendung des Stellenwertverfahrens ist gleichzeitig die einfachste Möglichkeit, von einem beliebigen anderen Zahlensystem ins Dezimalsystem umzuwandeln. Beispiele:

$$\begin{aligned} N_{10} = 1024_{10} &= 1\cdot10^3 + 0\cdot10^2 + 2\cdot10^1 + 4\cdot10^0 \\ N_8 = 43217_8 &= 4\cdot8^4 + 3\cdot8^3 + 2\cdot8^2 + 1\cdot8^1 + 7\cdot8^0 \\ &= 16384 + 1536 + 128 + 8 + 7 = 18063_{10} \\ N_2 = 110101_2 &= 1\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 53_{10} \end{aligned}$$

Allgemein: Der Wert einer natürlichen Zahl N_b (»N zur Basis b«, $N \in \mathbb{N}$) ist:

$$N_b = \sum_{i=0}^{n-1} Z_i \cdot b^i \tag{1}$$

b: Basis des Zahlensystems

N_b: Zahl im System mit der Basis b

n: Anzahl der Stellen der Zahl

Z_i: die i-te Ziffer von N_b; $i = 0, 1, \ldots, n-1$ - die Anzahl der Stellen ist also 1 größer als der größte Index i.

rationale Zahlen. Hierzu muss das Stellenwertverfahren erweitert werden. Beispiele:

$$\begin{aligned} Q_{10} = 53,789_{10} &= 5\cdot10^1 + 3\cdot10^0 + 7\cdot10^{-1} + 8\cdot10^{-2} + 9\cdot10^{-3} \\ Q_{16} = F4,573 &= F\cdot16^1 + 4\cdot16^0 + 5\cdot16^{-1} + 7\cdot16^{-2} + 3\cdot16^{-3} \\ Q_2 = 110,0101 &= 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 \\ &\quad +0\cdot2^{-1} + 1\cdot2^{-2} + 0\cdot2^{-3} + 1\cdot2^{-4} \end{aligned}$$

Allgemein: Der Wert einer rationalen Zahl Q_b (»Q zur Basis b«, $Q \in \mathbb{Q}$) ist nach DIN 44300:

$$\begin{aligned} Q_b &= \sum_{i=0}^{n-1} Z_i \cdot b^i + \sum_{i=-1}^{-m} Z_i \cdot \omega^i \\ &= \sum_{i=-m}^{n-1} Z_i \cdot b^i \end{aligned}$$

b: Basis des Zahlensystems

N_b: Zahl im System mit der Basis b

n: Anzahl der Stellen der Zahl vor dem Komma

m: Anzahl der Stellen der Zahl nach dem Komma

Z_i: die i-te Ziffer von N_b; $i = -m, \ldots, -1, 0, 1, \ldots, n-1$ - die Anzahl der Vorkommastellen ist also 1 größer als der größte Index i. Beispiel für die Indizierung:

$$\begin{aligned} Q_{16} = F4,573 &= \sum_{i=-3}^{1} Z_i \cdot 16^i \\ &= Z_1\cdot16^1 + Z_0\cdot16^0 + Z_{-1}\cdot16^{-1} + Z_{-2}\cdot16^{-2} + Z_{-3}\cdot16^{-3} \\ &= F\cdot16^1 + 4\cdot16^0 + 5\cdot16^{-1} + 7\cdot16^{-2} + 3\cdot16^{-3} \end{aligned}$$

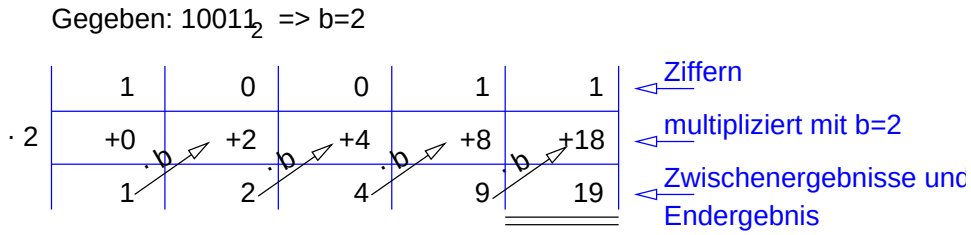


Abbildung 1: Auswertung von $10011_2 = 19_{10}$ nach dem Hornerschema

Hornerschema Diese iterative Darstellung des Stellenwertverfahrens hat folgende Vorteile: eine iterative Lösung »von innen heraus« ist möglich, d.h. unter Verwendung von bisherigen Teilergebnissen; die Rechnung wird auf Multiplikation und Addition reduziert, es müssen keine Potenzen mehr gebildet werden.

Natürliche Zahlen. Das Stellenwertverfahren kann mathematisch vereinfacht werden zu einem geklammerten Ausdruck, bei dem das jeweils letzte Zwischenergebnis mit der Basis multipliziert und darauf die nächste Stelle addiert wird. Allgemein:

$$N_b = ((((Z_{n-1} \cdot b + Z_{n-2}) \cdot b + Z_{n-3}) \ldots) \cdot b + Z_1) \cdot b + Z_0$$

b**:** Basis des Zahlensystems

N_b**:** Zahl im System mit der Basis b

n**:** Anzahl der Stellen der Zahl vor dem Komma

Z_i**:** die i-te Ziffer von N_b; $i = 0, 1, \ldots, n-1$

Beispiele für die Entstehung des Hornerschemas aus dem Stellenwertverfahren:

$$\begin{aligned} N_{10} = 1024 &= 1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 \\ &= \left(1 \cdot 10^2 + 0 \cdot 10^1 + 2\right) \cdot 10^1 + 4 \\ &= \left(\left(1 \cdot 10^1 + 0\right) \cdot 10^1 + 2\right) \cdot 10^1 + 4 \end{aligned}$$

Beispiel für die Berechnung nach dem Hornerschema (in Abbildung 1 ist für dieses Beispiel gezeigt, wie das Hornerschema übersichtlich in Tabellenform abgearbeitet wird):

$$10011_2 = (((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1 = 19$$

rationale Zahlen. Beispiel:

$$\begin{aligned} Q_{10} = 110,758_{10} &= 1 \cdot 10^{+2} + 1 \cdot 10^{+1} + 0 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2} + 8 \cdot 10^{-3} \\ &= \left(1 \cdot 10^{+1} + 1\right) \cdot 10^{+1} + 0 \cdot 10^0 + \left(7 + 5 \cdot 10^{-1} + 8 \cdot 10^{-2}\right) \cdot 10^{-1} \\ &= \left(1 \cdot 10^{+1} + 1\right) \cdot 10^{+1} + 0 \cdot 10^0 + \left(7 + \left(5 + 8 \cdot 10^{-1}\right) \cdot 10^{-1}\right) \cdot 10^{-1} \end{aligned}$$

Daraus leitet sich das allgemeine Verfahren ab, getrennt in einen Vorkommateil (Gleichung 4) und einen Nachkommateil (Gleichung 4):

$$\begin{aligned} Q_b &= ((((Z_{n-1} \cdot b + Z_{n-2}) \cdot b + Z_{n-3}) \ldots) \cdot b + Z_1) \cdot b + Z_0 && (2) \\ &+ \left(Z_{-1} + \left(Z_{-2} + \left(\ldots \left(Z_{-(m-1)} + Z_{-m} \cdot b^{-1}\right) \ldots\right) \cdot b^{-1}\right) \cdot b^{-1}\right) \cdot b^{-1} && (3) \end{aligned}$$

oder umgeordnet als

$$\begin{aligned} Q_b &= ((((Z_{n-1} \cdot b + Z_{n-2}) \cdot b + Z_{n-3}) \ldots) \cdot b + Z_1) \cdot b + Z_0 \\ &+ \left(\left(\left(\left(Z_{-m} \cdot b^{-1} + Z_{-(m-1)}\right) \cdot b^{-1} + Z_{-(m-2)}\right) \ldots\right) \cdot b^{-1} + Z_{-1}\right) \cdot b^{-1} \end{aligned}$$

b**:** Basis des Zahlensystems

N_b**:** Zahl im System mit der Basis b

n**:** Anzahl der Stellen der Zahl vor dem Komma

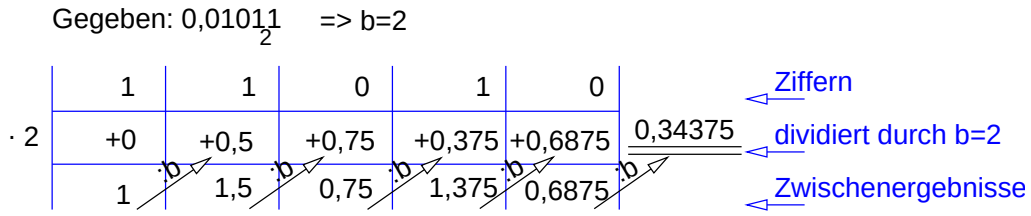


Abbildung 2: Auswertung $0,01011_2 = 0,34375_{10}$ nach dem Hornerschema

m**:** Anzahl der Stellen nach dem Komma

Z_i**:** die i-te Ziffer von N_b; $i = -m, \ldots -1, 0, 1, \ldots, n-1$

Weiteres Beispiel (in Abbildung 2 ist für den Nachkommateil gezeigt, wie das Hornerschema übersichtlich in Tabellenform abgearbeitet wird; man beachte dabei, dass die Reihenfolge der Nachkommastellen umgekehrt werden muss!):

$$\begin{aligned} 100,01011_2 &= ((1 \cdot 2 + 0) \cdot 2 + 0) \\ &+ \left(0 + \left(1 + \left(0 + \left(1 + 1 \cdot 2^{-1}\right) \cdot 2^{-1}\right) \cdot 2^{-1}\right) \cdot 2^{-1}\right) \cdot 2^{-1} \end{aligned}$$

Wichtige Zahlensysteme in der Informatik

Name	Basis	Ziffernvorrat
Dualsystem	2	$\{0; 1\}$
Oktalsystem	8	$\{0; 1; 2; 3; 4; 5; 6; 7\}$
Dezimalsystem	10	$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9\}$
Hexadezimalsystem	16	$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A; B; C; D; E; F\}$

7.2.2 Konvertierung ganzer Zahlen

Zahlenkonvertierung meint die Umwandlung einer Zahl im Quellzahlensystem in eine Zahl im Zielzahlensystem. Die jeweils letzte Ziffer im Zielzahlensystem ist der Rest bei Division durch die Basis b des Zielzahlensystems, denn der Wert der restlichen Summanden ist ohne Rest durch b teilbar, weil jeder dieser Summanden einen Faktor b^j, $j > 0$ enthält (bezogen auf Gleichung 1).

Wie kann 1027_{10} im Oktalsystem dargestellt werden?

$$\begin{aligned} 1027_{10} &= 8 \cdot 128 + 3 \\ &= 8 \cdot (8 \cdot 16 + 0) + 3 \\ &= 8 \cdot (8 \cdot (8 \cdot 2 + 0) + 0) + 3 \\ &= 8 \cdot (8 \cdot (8 \cdot (8 \cdot 0 + 2) + 0) + 0) + 3 \\ &= 2003_8 \end{aligned}$$

Diese sukzessive Division modulo 8 wird in einem Schema übersichtlich durchgeführt. 8 ist dabei die Basis des Zielzahlensystems, die Reste ergeben von unten nach oben gelesen die Ziffernfolge im Zielzahlensystem (hier im Oktalsystem; $1027_{10} = 2003_8$):

$$\begin{aligned} 1027 : 8 &= 128 \text{ Rest } 3 \\ 128 : 8 &= 16\text{Rest } 0 \\ 16 : 8 &= 2 \text{ Rest } 0 \\ 2 : 8 &= 0 \text{ Rest } 2 \end{aligned}$$

Weiteres Beispiel. Umwandlung von 201_{10} ins Dualsystem:

Hex	Dual	Hex	Dual
0	0000	8	1000
1	0001	9	1001
2	0010	*A*	1010
3	0011	*B*	1011
4	0100	*C*	1100
5	0101	*D*	1101
6	0110	*E*	1110
7	0111	*F*	1111

Tabelle 2:

$$\begin{array}{rcll} 201:2 & = & 100 & \text{Rest } 1 \\ 100:2 & = & 50 & \text{Rest } 0 \\ 50:2 & = & 25 & \text{Rest } 0 \\ 25:2 & = & 12 & \text{Rest } 1 \\ 12:2 & = & 6 & \text{Rest } 0 \\ 6:2 & = & 3 & \text{Rest } 0 \\ 3:2 & = & 1 & \text{Rest } 1 \\ 1:2 & = & 0 & \text{Rest } 1 \end{array}$$

Die Ziffernfolge im Dualsystem ist also 11001001_2. Soll eine Zahl ins Dualsystem umgewandelt werden, so kann die Anzahl der nötigen Rechenschritte reduziert werden, wenn man vorher ins Hexadezimalsystem umwandelt:

$$\begin{array}{rcll} 201:16 & = & 12 & \text{Rest } 9 \hat{=} 9_{16} \\ 12:16 & = & 0 & \text{Rest } 12 \hat{=} C_{16} \end{array}$$

Die Hexadezimalzahl $C9_{16}$ kann dann nach Tabelle 2 leicht zeichenweise in eine Binärzahl 1100 1001 umgesetzt werden.

Die Umwandlung zwischen Zahlensystemen, deren Basis eine Potenz von 2 ist (z.B. Oktalsystem, Hexadezimalsystem) erfolgt geschickterweise ziffernweise über das Dualsystem. Beispiel: Oktal- in Hexadezimalzahlen; jede Ziffer im Oktalsystem kann in 3 Bits des Dualsystems ausgedrückt werden, so dass die Dualzahl schnell gefunden werden kann. Dann beginnt man, diese Dualzahl von hinten beginnend zu je 4 Bits zu einer Hexadezimalziffer zusammenzufassen (bzw. zu jeweils so vielen Bits, wie nötig sind zur Darstellung einer Ziffer im Zielzahlensystem). Beispiele:

$$\begin{array}{rcl} 6351_8 & = & 110\,011\,101\,001_2 \\ & = & 1100\,1110\,1001_2 \\ & = & CE9_{16} \end{array}$$

$$\begin{array}{rcl} 25_8 & = & 010\,101_2 \\ & = & 01\,0101_2 \\ & = & 15_{16} \end{array}$$

7.2.3 Konvertierung gebrochener Zahlen

Die Umwandlung gebrochener Zahlen ist im Allgemeinen nicht exakt möglich, da aus einem endlichen, nichtperiodischen Bruch des Quellsystems ein nicht endender, periodischer Bruch im Zielsystem entstehen kann. Aufgrund des unten angewandten Verfahrens ist darum Q_Z um $\varepsilon \geq 0$ kleiner als Q_Q:

$$Q_Q = Q_Z + \varepsilon$$

mit

Q**:** eine gebrochene Zahl

Q_Q**:** gebrochene Zahl im Quellzahlensystem

Q_Z**:** gebrochene Zahl im Zielzahlensystem

ε**:** Wertdifferenz bzw. Ungenauigkeit; ihre Größe hängt von der Genauigkeit des verwandten Datenformats ab, d.h. von der Anzahl der darin darstellbaren Dezimalstellen.

a priori-Berechnung der Nachkommastellen n von Q_z bei vorgegebener dezimaler Genauigkeit[3]
Sei $Q_n < 1$ der Nachkommateil einer gebrochenen Zahl, berechnet bis zur n-ten Nachkommastelle:

$$Q_n = Z_{-1} \cdot 2^{-1} + Z_{-2} \cdot 2^{-2} + \ldots + Z_{-n} \cdot 2^{-n} + \varepsilon$$

Mathematische Grundlagen:

- Die endliche geometrische Reihe

$$s_n = \sum_{i=0}^{n} a \cdot q^i = a \cdot \frac{1-q^n}{1-q} \text{ für } |q| < 1 \tag{4}$$

- Und die zugehörige unendliche geometrische Reihe

$$s = \sum_{i=0}^{\infty} a \cdot q^i = \frac{a}{1-q} \text{ für } |q| < 1 \tag{5}$$

  Diese geht aus Gleichung 4 hervor unter Verwendung von

$$\lim_{i \to \infty} q^i = 0 \text{ für } |q| < 1$$

Verwendet man nun in Gleichung 5 $a = 1$, $q = \frac{1}{2}$, so erhält man die größtmögliche Summe der Nachkommastellen einer Binärzahl:

$$s = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{1-\frac{1}{2}} = 2$$

Die Ungenauigkeit ε, d.i. der Wert der vernachläsigten Nachkommastellen, ist nun maximal die Differenz zwischen dieser maximalen Summe der Nachkommastellen s und dem Summenwert bis zur Stelle n, d.i. s_n:

$$\begin{aligned} \varepsilon &\leq \sum_{i=n+1}^{\infty} \left(\frac{1}{2}\right)^i \\ &= s - s_n = 2 - \frac{1-\left(\frac{1}{2}\right)^n}{1-\frac{1}{2}} = 2 - 2 \cdot \left(1 - \frac{1}{2^n}\right) \\ &= \frac{2}{2^n} = 2^{-n+1} \\ \Rightarrow \varepsilon &\leq 2^{-n+1} \end{aligned}$$

Beispiel siehe Aufgabenlösungen in Kapitel 19.3.

Beispiele

- $Q_Q = 0,19_{10}$, $Q_Z = x_2$

$$\begin{aligned} 0,19_{10} &= z_{-1} \cdot 2^{-1} + z_{-2} \cdot 2^{-2} + z_{-3} \cdot 2^{-3} + \ldots \quad | \cdot 2 \\ \Leftrightarrow 0 \cdot 2^0 + 0,38 = 0,38 &= z_{-1} \cdot 2^0 + z_{-2} \cdot 2^{-1} + z_{-3} \cdot 2^{-2} + \ldots \quad | \cdot 2 \end{aligned}$$

  Durch Koeffizientenvergleich (zwei Polynome $a_0x^0 + a_1x^1 + \ldots$ und $b_0x^0 + b_1x^1 + \ldots$ sind gleich, wenn die Koeffizienten der sich entsprechenden Potenzen der Unbekannten gleich sind: $a_0 = b_0$, $a_1 = b_1$ usw.)

[3] Vergleiche [4] im Teil »Zahlendarstellung und Zahlenumwandlung« Aufgabe 3a.

ergibt sich hier: $0 \cdot 2^0 = z_{-1} \cdot 2^0 \Rightarrow z_{-1} = 0$. Dieses Verfahren wird schrittweise weiter angewandt, bis die ganze Zahl im Zielzahlensystem vorliegt:

$$\begin{aligned}
\Leftrightarrow 0 \cdot 2^1 + 0 \cdot 2^0 + 0,76 = 0,76 &= z_{-1} \cdot 2^1 + z_{-2} \cdot 2^0 + z_{-3} \cdot 2^{-1} + \ldots \quad | \cdot 2 \\
&\Rightarrow z_{-2} = 0 \\
\Leftrightarrow 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0,52 = 1,52 &= z_{-1} \cdot 2^2 + z_{-2} \cdot 2^1 + z_{-3} \cdot 2^0 + \ldots \quad | \cdot 2 \\
&\Rightarrow z_{-3} = 1 \\
\Leftrightarrow 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0,04 = 3,04 &= z_{-1} \cdot 2^3 + z_{-2} \cdot 2^2 + z_{-3} \cdot 2^1 + z_{-4} \cdot 2^0 + \ldots \\
&\Rightarrow z_{-4} = 1
\end{aligned}$$

Dabei ist zu beachten: Tritt auf der linken Seite der Gleichung bei Multiplikation mit der Basis des Zielzahlensystems (hier 2) ein Rest ≥ 1 auf, so ist dieser (wie im Beispiel geschehen) zu schreiben als:

$$a_0, a_{-1}a_{-2} \ldots = a_0 \cdot b^0 + 0, a_{-1}a_{-2} \ldots$$

Damit kann für die Umwandlung das folgende Schema verwendet werden, wobei sich die Ziffernfolge im Zielzahlensystem dann von oben nach unten ergibt:

Schritt	Operation	Zwischenergebnis	Ziffer im Zielsystem
1	$0,19 \cdot 2$	$0,38$	0
2	$0,38 \cdot 2$	$0,76$	0
3	$0,76 \cdot 2$	$1,52$	1
4	$0,52 \cdot 2$	$1,04$	1
5	$0,04 \cdot 2$	$0,08$	0
6	$0,08 \cdot 2$	$0,16$	0
7	$0,16 \cdot 2$	$0,32$	0
8	$0,32 \cdot 2$	$0,64$	0
9	$0,64 \cdot 2$	$1,28$	1
10	$0,28 \cdot 2$	$0,56$	0
$\vdots$			

- $Q_Q = 2629,5586_{10}$, $Q_Z = x_2$
  - Vorkommastellen (über das Hexadezimalsystem)

$$\begin{aligned}
2629 : 16 &= 164 \quad \text{Rest } 5 \hat{=} 5_{16} \\
164 : 16 &= 10 \quad \text{Rest } 4 \hat{=} 4_{16} \\
10 : 16 &= 0 \quad \text{Rest } 10 \hat{=} A_{16}
\end{aligned}$$

  - Nachkommastellen (über das Hexadezimalsystem)

Schritt	Operation	Zwischenergebnis	Ziffer im Zielsystem
1	$0,5586 \cdot 16 =$	$8,9376$	8_{16}
2	$0,9376 \cdot 16 =$	$15,0016$	F_{16}
3	$0,0016 \cdot 16 =$	$0,0256$	0_{16}
4	$0,0256 \cdot 16 =$	$0,4096$	0_{16}
$\vdots$			$\vdots$

Damit ergibt sich für die Umwandlung der gesamten Zahl nach Tabelle 2:

$$\begin{aligned}
2629,5586_{10} &= A45,8F_{16} + \varepsilon \\
&= 1010\,0100\,0101,\ 1000\,1111_2 + \varepsilon
\end{aligned}$$

mit

$$\begin{aligned}
\varepsilon &= 2629,5586_{10} - A45,8F_{16} \\
&= 6,25 \cdot 10^{-6}
\end{aligned}$$

$$
\begin{array}{l}
1010_2 \cdot 1100_2 \\
\hline
\qquad 0000_2 \\
+ \quad 1100_2 \\
+ \quad 0000_2 \\
+ \quad 1100_2 \\
\hline
\quad 1111000_2
\end{array}
$$

Abbildung 3: Multiplikation $10 \cdot 12 = 120$ im Dualsystem

7.3 Dualzahlenarithmetik

7.3.1 Grundoperationen

Wie unter Kapitel 7.2 gezeigt, können ganze und gebrochene Zahlen beliebig zwischen verschiedenen Zahlensystemen konvertiert werden. Die Grundoperationen folgen in allen Zahlensystemen dem gleichen Schema, nämlich dem aus dem Dezimalsystem bekannten:

Addition

$$
\begin{aligned}
1010_2 + 1100_2 &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
&\quad +1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\
&= 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
&= 10110_2
\end{aligned}
$$

Die Addition erfolgt also analog zur schriftlichen Addition mit Übertrag im Dezimalsystem, denn $1_2 + 1_2 = 10_2$. Vergleiche die Tabelle »Addition von Dualzahlen« in Quelle [3, Anhang 2].

Subtraktion

$$
\begin{aligned}
1100_2 - 1010_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\
&\quad - \left(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0\right) \\
&= 1 \cdot 2^2 - 1 \cdot 2^1 = 1 \cdot 2^1 \\
&= 10_2
\end{aligned}
$$

Entsprechend lautet die »Operationstafel für die Subtraktion von Ziffern im Dualsystem« in Ergänzung zu Quelle [3, Anhang 2]:

-	0	1
0	0	1
1	1-B	0

»B« ist das sog. »Borgbit«, das Gegenstück zum »Übertrag« der Addition.

Beachte: Das Schema zur schriftlichen Subtraktion ist nicht geeignet für Subtraktionen mit negativen Ergebnissen; ggf. vertausche man also Minuend und Subtrahend und nehme die Gegenzahl des Ergebnisses dieser Rechnung als Endergebnis.

Multiplikation

$$
\begin{aligned}
1010_2 \cdot 1100_2 &= \left(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0\right) \cdot \left(1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0\right) \\
&= \left(1 \cdot 2^3 + 1 \cdot 2^1\right) \cdot \left(1 \cdot 2^3 + 1 \cdot 2^2\right) \\
&= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 \\
&= 1111000_2
\end{aligned}
$$

Das Schema zur schriftlichen Multiplikation für diese Aufgabe ist in Abbildung 3 dargestellt. Die schriftliche Multiplikation kann natürlich auch mit Kommazahlen durchgeführt werden (Beispiel siehe Abbildung 4). Verfahren:

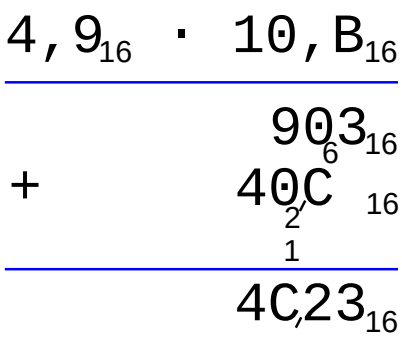


Abbildung 4: Multiplikation $4,5625 \cdot 16,6875 = 76,136\ldots$ im Hexadezimalsystem

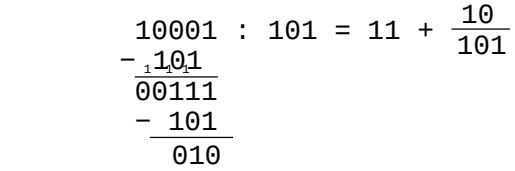


Abbildung 5: Division $17 : 5 = 3 + \frac{2}{5}$ im Dualsystem

1. Wiederhole von der letzten bis zur ersten Stelle des ersten Faktors:
   (a) Beginne eine neue Zeile für eine Teilsumme und lege die Beginnposition zum Aufschreiben der Teilsumme von hinten nach vorne fest. In der ersten Zeile ist diese Position unter der letzten Stelle des zweiten Faktors, in jeder folgenden Zeile eine Stelle weiter nach links gerückt.
   (b) Wiederhole von der letzten bis zur ersten Stelle des zweiten Faktors:
      i. Multipliziere die aktuellen Stellen beider Faktoren miteinander
      ii. Schreibe das Ergebnis von rechts nach links auf. Eine ggf. existierende zweite Stelle muss als Index eine Stelle weiter links geschrieben werden!
2. Ziehe Strich zur Summenbildung, dabei eine Zeile für weitere Indizes freilassen.
3. Summiere alle Teilsummen zum Endergebnis, dabei alle Indizes mit berücksichtigend.
4. Setze das Komma in der Teilsumme, in der die Einer des ersten Faktors mit dem zweiten Faktor multipliziert wurden: diese Teilsumme hat genausoviele Nachkommastellen wie der zweite Faktor.
5. Setze das Komma im Endergebnis an derselben Stelle.

Division Die Division im Dualsystem verläuft ganz analog zur Division im Dezimalsystem und ist eigentlich sogar einfacher, da man nur entscheiden muss, ob die nächste Ziffer des Ergebnisses eine 0 oder eine 1 ist, d.h. ob der Divisor 0mal oder 1mal in den aktuellen Dividenden hineinpasst. Beispiel: Abbildung 5. Entsprechend lautet die »Operationstafel für die Division von Ziffern im Dualsystem« in Ergänzung zu Quelle [3, Anhang 2]:

:	0	1
0	Fehler	Fehler
1	0	1

7.3.2 Komplementbildung[4]

Das Komplement einer Zahl ist die Ergänzung zu einer anderen Zahl.

[4] Hier werden mathematische Grundlagen dargestellt, die in den Kapitel 7.3.3 und 7.3.4 benötigt werden.

Zehnerkomplement Sei Q_{10} eine Zahl im Dezimalsystem und n die Anzahl ihrer Vorkommastellen. Dann heißt die Zahl $K_{10} = 10^n - |Q_{10}|$ das »Zehnerkomplement von Q_{10}«. Es ist die Ergänzung zur Zahl 10^n, d.h. zur nächtgrößeren Zehnerpotenz, denn $K_{10} + Q_{10} = 10^n$. Beispiele:

$$K_{10}(15) = 100 - |15| = 85 = K_{10}(-15)$$

$$K_{10}(187,375) = 1000 - |187,375| = 812,625 = K_{10}(-187,375)$$

Zweierkomplement Sei Q_2 eine Zahl im Dualsystem und n die Anzahl ihrer Vorkommastellen. Dann heißt die Zahl $K_2 = 2^n - |Q_2|$ das »Zweierkomplement von Q_{10}«. Es ist die Ergänzung zur Zahl 2^n, d.h. zur nächtgrößeren Zweierpotenz.

Stellenkomplement Das Stellenkomplement ist die Ergänzung zu der größten Zahl, die mit der aktuellen Anzahl Ziffern im jeweiligen Zahlensystem darstellbar ist. Man erhält es also stellenweise, indem man für jede Ziffer einer Zahl die Ziffer sucht, die sie zum höchsten Ziffernwert ergänzt. (Beispiel: das Stellenkomplement zu 49 ist 50, denn $4+5=9$ und $9+0=9$; Probe: $49+50=99$). Im Dualsystem entspricht die Stellenkomplementbildung gerade der Invertierung[5] einer Dualzahl, denn 1 wird durch 0 zu 1 ergänzt und 0 wird durch 1 zu 1 ergänzt.
Um das Komplement zur nächsten Stufenzahl[6] aus dem Stellenkomplement zu erhalten, muss man nur noch 1 addieren, denn dadurch wird (am Beispiel von oben) die 99 zur 100. Beispiele:

Zahl	Stellenkomplement	Komplement zur nächsten Stufenzahl
48	$K_9 = 51$ **Probe:** $48 + 51 = 99$	$K_{10} = K_9 + 1 = 52$ **Probe:** $48 + 52 = 100$
0100_2	$K_1 = 1011_2$ **Probe:** $0100_2 + 1011_2 = 1111_2$	$K_2 = K_1 + 1 = 1100_2$ **Probe:** $0100_2 + 1100_2 = 10000_2$
N_2	$K_1 = \overline{N_2}$ **Probe:** $N_2 + \overline{N_2}$	$K_2 = \overline{N_2} + 1$ **Probe:** $N_2 + \overline{N_2} + 1$

Für die Bildung des Zweierkomplementes aus dem Stellenkomplement gilt also (vgl. letzte Zeile in obiger Tabelle):

$$K_2(N_2) = \overline{N_2} + 1$$

denn die Probe ergibt stets (sei n die Zahl der Vorkommastellen von N_2):

$$\begin{aligned} N_2 + K_2(N_2) &= N_2 + \overline{N_2} + 1 \quad (6) \\ &= 1 \cdot 2^n \quad (7) \end{aligned}$$

Komplement als Rest bei modulo-Division Vorangestellt sei die Definition des Restes bei Ganzzahldivision: »Seien $m \in \mathbb{N}$ und $z \in \mathbb{Z}$. Dann existieren eindeutig bestimmte Zahlen $q, r \in \mathbb{Z}$ mit $z = qm + r$ und $0 \leq r < m$. Die Zahl r heißt »Rest von z bei Division durch m« oder kürzer »Rest von z modulo m«. Man findet den Rest von z modulo m wie folgt: Bestimme die größte Zahl $q \in Z$ mit $q \cdot m \leq z$ und bilde $r = z - qm$.« Einige Beispiele: Sei $m = 5$. Dann können Zahlen z in die Darstellung $z = qm + r$ zerlegt werden. Diese Darstellung ist eindeutig:

- $9 = 1 \cdot 5 + 4$
- $13 = 2 \cdot 5 + 3$
- $-4 = (-1)\,5 + 1$
- $-18 = (-4)\,5 + 2$

Die Bildung des Restes modulo m ist also der Abstand $r \geq 0$ von z zur Zahl qm, $q \in \mathbb{Z}$, die das gleiche Vorzeichen hat wie z. Nimmt man für m eine Stufenzahl an, so entspricht das genau der Definition des Komplements, allerdings nur für negative Zahlen:

Für den Rest r bei Ganzzahldivision gilt ja:

$$z = qm + r \quad (8)$$

[5] Invertierung einer Dualzahl ist der Tausch der Ziffern 0 bzw. 1 mit ihren komplementären Ziffern 1 bzw. 0. Sie kann schaltungstechnisch mit einem Feld von Invertern (logisches NOT) erfolgen.

[6] d.i. das Zehnerkomplement im Dezimalsystem, das Zweierkomplement im Dualsystem usw.

mit $0 \leq r < m$ und $q \in \mathbb{Z}$, $m \in \mathbb{N}$. Übertragen auf die Definition des Komplements heißt das:

$$r = m - |z| \tag{9}$$

Können beide Gleichungen zugleich gelten, d.h. sind Komplementbildung und Modulo-Division äquivalent? Ja, aber nur für negative Zahlen. Denn: Setze Gleichung 9 in Gleichung 8 ein:

$$\begin{aligned} z &= (q+1)\,m - |z| \\ \Leftrightarrow z + |z| &= (q+1)\,m \end{aligned}$$

Für $z < 0 \Rightarrow z + |z| = 0$ ist diese Gleichung erfüllt:

$$\begin{aligned} 0 &= (q+1)\,m \\ \Leftrightarrow q &= -1 \end{aligned}$$

7.3.3 Ganzzahlige Arithmetik

Das Vorzeichen ganzer Zahlen wird im Integer-Datentyp durch eine führende 1 (für »−«) bzw. eine führende 0 (für »+«) kodiert. Gibt es eine Möglichkeit zur einfachen Addition und Subtraktion ganzer Zahlen trotz dieser »Sonderrolle« des führenden Bits? Ja, denn eigentlich entstand obige Definition des Integer-Datentyps aus der für Computer sehr günstigen Tatsache, dass die Subtraktion auf die Addition mit dem Komplement zur nächsten Stufenzahl zurückgeführt werden kann. Und die Darstellung aller negativen Zahlen durch ihr Zweierkomplement ergibt nun gerade die obige Definition des Integer-Datentyps.

- eine positive Zahl wird wie bisher als binär kodierte Dezimalzahl dargestellt
- eine negative Zahl wird durch ihr Zweierkomplement dargestellt. Es hat stets eine führende 1, das also das Vorzeichen − definiert.

Rückführung der Subtraktion auf die Addition: Im Dezimalsystem Aus der Definition des Zehnerkomplements folgt als Möglichkeit zur Darstellung einer Zahl $Q_{10} > 0$:

$$\begin{aligned} K_{10} &= 10^n - |Q_{10}| \\ \Leftrightarrow -|Q_{10}| &= K_{10} - 10^n \\ \Leftrightarrow -Q_{10} &= K_{10} - 10^n \end{aligned}$$

Damit kann jede Subtraktion durch eine Addition und die anschließende Subtraktion der zum Minuenden gehörenden nächstgrößeren Stufenzahl dargestellt werden:

$$R_{10} - Q_{10} = R_{10} + K_{10}\,(Q_{10}) - 10^n$$

$$17 - 15 = 17 + 85 - 100 = 2 \tag{10}$$

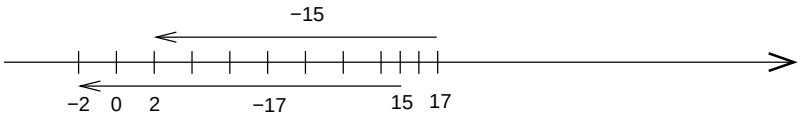
$$15 - 17 = 15 + 83 - 100 = -2 \tag{11}$$

Die anschließende Subtraktion der Stufenzahl 100 kann wie folgt ersetzt werden:

- Ist $a \geq b$ wie in Gleichung 10, so vernachlässige man den letzten Übertrag der durchgeführten Addition.[7] Dadurch ist deren Ergebnis 2 statt 102.
- Ist $a < b$ wie in Gleichung 11, so interpretiere man das Ergebnis der Addition (98) als Zehnerkomplement einer negativen Zahl (nämlich als $98 = K_{10}\,(-2)$). Das gewünschte Ergebnis erhält man dann durch eine weitere, aufhebende Zehnerkomplementbildung und Voranstellen eines Minuszeichens: $-K_{10}\,(98) = -2$.

[7] In Computern geschieht dies einfach dadurch, dass der zur verfügung stehende Speicher mit n Stellen nicht zur Aufnahme des Übertrags in die $n+1$-te Stelle ausreicht.

Herkömmliche Subtraktion:



Subtraktion unter Verwendung des Zehnerkomplements:

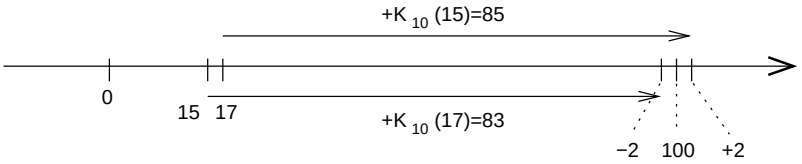


Abbildung 6: Subtraktion durch Zehnerkomplementbildung

Diese Interpretationen $102\hat{=}2$ bzw. $98\hat{=}-2$ entsprechen der Festsetzung von 100 als neuem Nullpunkt (siehe Abbildung 6). Wir haben damit die Subtraktion auf eine Addition und Komplementbildung zurückgeführt, nach folgender Definition ($a, b \in \mathbb{R}$, Zahl der Vorkommastellen $n = n_a = n_b$):

$$a - b = \begin{cases} a + K_{10}(b) - 10^n & \text{für } a \geq b \\ -K_{10}(a + K_{10}(b)) & \text{für } a < b \end{cases} \tag{12}$$

Man kommt also für Addition und Subtraktion mit positiven Zahlen aus. Für die Zahlendarstellung im Computer hat das natürlich den Vorteil, dass es Datentypen wie `unsigned int` und `unsigned float` geben kann, die nicht zwischen positiven und negativen Zahlen unterscheiden.

Für $a = b$ ergibt sich nach 12: $a - a = 0 = a + K_{10}(a) - 10^n$. Wenn man nun wie oben vereinbart die Subtraktion -10^n durch die Vernachlässigung des letzten Übertrags von $a + K_{10}(a)$ ersetzt, so ergibt sich, dass das Zehnerkomplement als Gegenzahl von a interpretiert werden kann:

$$\begin{aligned} a + K_{10}(a) &\cong 0 \\ \Leftrightarrow K_{10}(a) &\cong -a \end{aligned}$$

Genau das ist die Art, wie negative Zahlen im Computer dargestellt werden: durch das Komplement der positiven Zahl.

Da nun das Komplement einer negativen Zahl z zu einer Stufenzahl m gerade der Bildung des Restes bei Ganzzahldivision entspricht (siehe Kapitel 7.3.2) und weiterhin die positiven Zahlen ihren Resten bei Ganzzahldivision durch m entsprechen ($z \bmod m = z$), ergibt sich die Tatsache, dass Ganzzahlen im Computer immer durch ihre Reste mod m dargestellt sind. Mathematisch geschehen also alle Rechnungen im Restklassenring $Z_m = \mathbb{Z}/_{(m)} = \{[0], [1], \ldots, [m-1]\}$ - in einem solchen Restklassenring kann man tatsächlich rechnen[8].

Rückführung der Subtraktion auf die Addition: Im Dualsystem Im obigen Abschnitt wurde gezeigt, dass im Computer alle Zahlen durch ihre Reste mod m (die Reste bei Ganzzahldivision durch die folgende Stufenzahl) dargestellt werden. Der Wertebereich der so dargestellten Zahlen muss so eingegrenzt werden, dass diese Abbildung eindeutig bleibt. Für 4-Bit Binärzahlen (Komplementbildung zu $2^4 = 16$) sieht diese Abbildung wie folgt aus:

[8] Dies ist Stoff der Vorlesung Mathematik 2. Der hier dargestellte mathematische Hintergrund ist die Verallgemeinerung des im Abschnitt »Rückführung der Subtraktion auf die Addition: Im Dezimalsystem« dargestellten Verfahrens und korrigiert die Abschnitte »Rechnerinterne Informationsdarstellung ... Darstellung ganzer Zahlen (2)« und »Rechnerinterne Informationsdarstellung ... Darstellung ganzer Zahlen (3)« in [3, S. 26].

Zahl	Abbildung	Binärdarstellung
-8	$-8 \bmod 16 = 8$	1000
$\vdots$		$\vdots$
-1	$-1 \bmod 16 = 15$	1111
0	$0 \bmod 16 = 0$	0000
1	$1 \bmod 16 = 1$	0001
$\vdots$		$\vdots$
7	$7 \bmod 16 = 7$	0111

Dies ist die einfachste Möglichkeit, die Ganzzahldarstellung im Computer zu verstehen: Die Zahlen werden auf ihre Reste mod m abgebildet, und es ist eine mathematische Tatsache, dass man im entstehenden Restklassenring wie mit den Zahlen selbst rechnen kann. Nur dass dabei eben nur positive Zahlen verwendet werden.

In einem Mikroprozessor wird die Operation $a - b$ unabhängig davon, ob das Ergebnis negativ oder postiv ist, wie folgt in einem Befehl SUB durchgeführt, durch Rückführung auf die Addition $a + (-b)$.

1. Bilde die Argumente a, $-b$ durch $a \bmod m = a$ und $-b \bmod m = K_m(-b) = K_m(b)$ auf die Darstelleung im Restklassenring Z_m ab. Beispiel:

$$\begin{aligned} 4_{10} - 11_{10} &= 0000\,0100_2 - 0000\,1011_2 \\ 0000\,0100_2 \bmod 256 &= 0000\,0100_2 \\ -0000\,1011_2 \bmod 256 &= 1111\,0101_2 \end{aligned}$$

2. Addiere $a + (-b)$ in dieser Darstellung. Beispiel:

$$0000\,0100_2 + 1111\,0101_2 = 0\,1111\,1001_2$$

3. Bilde das Ergebnis mit einer inversen Abbildung vom Restklassenring auf die Zahlen ab. Dabei ist ein Übertrag in die $n+1$-te Stelle stets völlig zu ignorieren[9]:

$$\begin{array}{rr} 4 & 0100 \\ \underline{-3} & \underline{+1101} \\ 1 & 10001 \end{array}$$

   Dieser Übertrag m hat nämlich keinen Einfluss darauf, in welcher Restklasse aus Z_m das Ergebnis enthalten ist, also bei bijektiver Abbildung zischen Zahlen und Restklassen auch keinen Einfluss auf das Ergebnis: Das Ergebnis $10001_2 = 17_{10}$ ist in der Restklasse [1] enthalten, also ist das Ergebnis 1. So vereinfacht ist die Rückabbildung von den Restklassen auf die Zahlen also:

   - ist das führende Bit (an der n-ten Stelle!) 0, so ist das Ergebnis die durch n Stellen dargestellte positive Dualzahl.
   - ist das führende Bit (an der n-ten Stelle!) 1, so ist das Ergebnis als Zwischenergebnis anzusehen, nämlich als Zweierkomplement eines negativen Ergebnisses. Man erhält dieses Ergebnis durch erneute, aufhebende Komplementbildung und Voranstellen eines negativen Vorzeichens. Beispiel:

$$-K_2(1111\,1001_2) = -0000\,0111_2 = -7_{10}$$

     denn $-7_{10} \bmod 256 = 1111\,1011_2$.

Zusammenfassung und Regeln

Allgemeine Prinzipien

- Negative Zahlen werden durch ihr Zweierkomplement dargestellt
- Die Subtraktion wird auf die Addition nach einfachen Regeln wie oben beschrieben zurückgeführt
- Die Division wird auf die Multiplikation nach einfachen Regeln zurückgeführt

[9] In Computern geschieht das automatisch, weil keine weiteren Binärstellen zur Zahldarstellung vorhanden sind.

Addition: Voraussetzungen

- Das Zahlenformat wurde so gewählt, dass keine Bereichsüberschreitungen (»Überlauf«) auftreten können.
- Negative Zahlen sind durch ihr Zweierkomplement dargestellt
- Die Anzahl der Binärstellen eines ganzzahligen Operanden sei n.

Addition: Regeln

- Ein Übertrag in die $n+1$-te Stelle wird stets ignoriert. Er hat keinen Einfluss auf das Ergebnis, denn das Zahlenformat wurde ja so gewählt, dass ein Überlauf ausgeschlossen ist.
- Für die Interpretation der Summe $s = a + b$, $a, b \in \mathbb{Z}$ gilt:
  - Die n-te Stelle gibt Auskunft über das Vorzeichen ($0\hat{=}+$; $1\hat{=}-$)
  - `Falls` $s > 0$ `dann` Interpretation als positive Dualzahl
    `sonst` Interpretation als negative Dualzahl im Zweierkomplement

Addition: Beispiele

- $$\begin{array}{rr} 4 & 0100 \\ \underline{+3} & \underline{+0011} \\ 7 & 0111 \end{array}$$
- $$\begin{array}{rr} 4 & 0100 \\ \underline{-3} & \underline{+1101} \\ 1 & \not{1}0001 \\ & = 0001 \end{array}$$

  Zur Erinnerung: Subtraktion erfolgt durch Addition mit dem Zweierkomplement; die $n+1$-te Stelle wird stets ignoriert
- $$\begin{array}{rr} 4 & 0100 \\ \underline{-4} & \underline{+1100} \\ 0 & \not{1}0000 \\ & = 0 \end{array}$$
- $$\begin{array}{rr} 4 & 0011 \\ \underline{-4} & \underline{+1100} \\ -1 & 1111 \end{array}$$
- $$\begin{array}{rr} -4 & 1100 \\ \underline{-4} & \underline{+1100} \\ -8 & \not{1}1000 \end{array}$$
- $$\begin{array}{rr} -4 & 1100 \\ \underline{-7} & \underline{+1001} \\ -11 & \not{1}0101 \\ & \hat{=}5_{10} \end{array}$$

  Hier hat ein Wertüberlauf stattgefunden, deshalb liefert die Addition der Dualzahlen das falsche Ergebnis. -11 ist mit 4 Bit breiten Dualzahlen (Wertebereich $-8 \ldots +7$) nicht darstellbar!

7.3.4 Gleitkommaarithmetik

Berechnungen mit gebrochenen Zahlen erfolgen üblicherweise in halblogarithmischer Darstellung im IEEE754-Format. Es gelten folgende Rechenregeln:

Sei $a = m_a \cdot 2^{e_a}$, $b = m_b \cdot 2^{e_b}$; seien die Exponenten bei Addition und Subtraltion identisch ($e_a = e_b = e$), was durch Kommaverschiebung immer erreicht werden kann.

Addition

$$\begin{aligned} a + b &= m_a \cdot 2^e + m_b \cdot 2^e \\ &= (m_a + m_b) \cdot 2^e \end{aligned}$$

Subtraktion

$$\begin{aligned} a - b &= m_a \cdot 2^e - m_b \cdot 2^e \\ &= (m_a - m_b) \cdot 2^e \end{aligned}$$

Multiplikation

$$\begin{aligned} a \cdot b &= m_a \cdot 2^{e_a} \cdot m_b \cdot 2^{e_b} \\ &= m_a \cdot m_b \cdot 2^{(e_a + e_b)} \end{aligned}$$

Division

$$\begin{aligned} a : b &= \frac{m_a \cdot 2^{e_a}}{m_b \cdot 2^{e_b}} \\ &= \frac{m_a}{m_b} \cdot 2^{(e_a - e_b)} \end{aligned}$$

Wie bei der Dualzahlenarithmetik mit Ganzzahlen kann die Subtraktion auf die Addition zurückgeführt werden, wenn negative Zahlen im Zweierkomplement dargestellt werden (vgl. Kapitel 7.3.2). Beispiel für Zweierkomplementbildung bei Gleitkommazahlen:

$$\begin{aligned} 4,5_{10} &= 0100,1_2 \\ K_2\,(0100,1_2) &= 1011,0_2 + 0000,1_2 \\ &= 1011,1_2 \end{aligned}$$

denn

$$\begin{aligned} 4,5_{10} + K_2\,(4,5_{10}) &= 0100,1_2 + 1011,1_2 \\ &= 10000,0_2 \end{aligned}$$

$$\begin{aligned} 2,5_{10} &= 0010,1_2 \\ K_2\,(0010,1_2) &= 1101,0_2 + 0000,1_2 \\ &= 1101,1_2 \end{aligned}$$

Beispiele zu Addition und Subtraktion von Gleitkommazahlen

- $$\begin{array}{rr} 4,5_{10} & 0100,1_2 \\ \underline{+2,5_{10}} & \underline{+0010,1_2} \\ 7,0_{10} & 0111,0_2 \end{array}$$
- $$\begin{array}{rr} 4,5_{10} & 0100,1_2 \\ \underline{-2,5_{10}} & \underline{+1101,1_2} \\ 2,0_{10} & \not{1}0010,0_2 \end{array}$$

- $$\begin{array}{rr} 2,5_{10} & 0010,1_2 \\ \underline{-4,5_{10}} & \underline{+1011,1_2} \\ -2,0_{10} & 1110,0_2 \end{array}$$

  An der 1 an der n-ten Stelle erkennt man, dass es sich um das Komplement einer negativen Zahl, des Ergebnisses, handelt. Also muss man wiederum das Zweierkomplement bilden und ein negatives Vorzeichen voranstellen:

  $$\begin{aligned} -K_2\left(1110,0_2\right) &= -\left(0001,1_2 + 000,1_2\right) \\ &= -0010,0_2 = -2_{10} \end{aligned}$$

- $$\begin{array}{rr} -2,5 & 1101,1 \\ \underline{-4,5} & \underline{+1011,1} \\ -7,0 & \not{1}1001,0 \end{array}$$

  An der 1 an der n-ten Stelle erkennt man, dass es sich um das Komplement einer negativen Zahl, des Ergebnisses, handelt. Also muss man wiederum das Zweierkomplement bilden und ein negatives Vorzeichen voranstellen:

  $$\begin{aligned} -K_2\left(1001,0_2\right) &= -\left(0110,1_2 + 0000,1_2\right) \\ &= -0111,0_2 = -7_{10} \end{aligned}$$

8 Betriebssysteme

8.1 Was ist ein Betriebssystem?

8.2 Grundbegriffe

Prozess. Da mehrere Prozesse (wie Eltern- und Kindprozesse) gleichzeitig laufen können, müssen sie in folgenden Dingen synchronisiert werden:

- Was soll passieren, wenn zwei Prozesse dasselbe Betriebsmittel belegen wollen? Die Synchronisation erfolgt durch gegenseitigen Ausschluss (mutual exclusion): der Prozess, der das Betriebsmittel bis zur Anforderung durch den zweiten Prozess belegt hatte, darf es behalten und schließt damit die Zuweisung des Betriebsmittels an den zweiten Prozess aus.
- Was soll passieren, wenn die weitere Ausführung eines Prozesses vom Zustand eines anderen Prozesses abhängig ist? Hier erfolgt Zustands-Synchronisation (condition synchronisation) durch das Betriebssystem, so dass die gegenseitige Abhängigkeit in der Ausführung von Aktionen im parent process und child process erfüllt wird.

Teil III
Programmiersprachen und -systeme

9 Grundbegriffe

9.1 Generationen von Programmiersprachen

Man unterscheidet verschiedene Generationen der Entwicklung von Programmiersprachen, gleichzeitig verschiedene Abstraktionsebenen. Die Programmiersprachen werden nach ihrem Abstraktionsgrad unterschieden, d.h. wie weit richtet sich die Sprache nach Maschinenstruktur oder menschlichem Denken. Je höher abstrahiert eine Sprache ist, desto besser ist sie vom Menschen verstehbar und anwendbar.

9.1.1 Maschinensprachen (Sprachen der ersten Generation)

Der Rechner basiert auf binären Schaltelementen (0 oder 1), mit denen all unsere Informationen kodiert werden können. Maschinensprachen sind vom Rechner direkt umsetzbar; es ist das Sprachniveau, das vom Mikroprozessor direkt verstanden wird. Vorteil: von der Maschine verstehbar; Nachteil: vom Menschen nur sehr schwer verstehbar. Maschinensprachen verwenden die direkten Adressen von Speicherzellen / Daten; es sind nur elementare Operationen möglich, Schleifen zum Beispiel sind Kombinationen vieler Maschinenbefehle. Maschinensprachen sind heute auf Spezialgebiete beschränkt, werden heute i. A. nur noch von Leuten gelernt, die Mikroprozessoren bauen. Deshalb kam man früh auf die Idee, die Bitmuster für den Menschen verständlicher darzustellen:

9.1.2 Assemblersprachen (Sprachen der zweiten Generation)

Die schlechte Lesbarkeit der Maschinensprachen wurde in Assemblersprachen dadurch verbessert, dass wiederkehrende Bitmuster (z.B. Multiplikation von zwei Zahlen in Registern) zu einem Befehl zusammengefasst wurden. Assemblersprachen sind wie Maschinensprachen sehr strukturäquivalent bzw. transparent zur Hardeware, deshalb auch abhängig von einer bestimmten Hardware (für jeden Prozessor gibt es eine Assemblersprache). Assemblerprogramme sind deshalb nicht portierbar, im Gegensatz zu Hochsprachen. Direkte Speicheradressierung ist auch für Assemblersprachen typisch; neu ist die Verwendung von leichter zu merkenden Mnemonics (Abkürzungen; z.B. LOAD, JUMP, ADD) statt Bitmustern für Befehle.

Assemblersprachen müssen vor der Ausführung in Maschinensprache bersetzt werden; der Übersetzer heißt auch Assembler. Heute hat Assembler auch schon hochentwickelte, hochsprachenähnliche Konstrukte.

Zur Assemblerprogrammierung passt an der FH die Vorlesung »maschinennahe Programmierung«. Jedoch nimmt dort wie in der Industrie der Anteil von Assemblerprogrammierung immer weiter ab und bleibt Spezialanwendungen vorbehalten.

9.1.3 Problemorientierte Sprachen (Sprachen der dritten Generation)

Diese Sprachen orientieren sich zum ersten Mal nicht an den Problemen und der Funktion der Maschine, sondern an Problemwelt und Denken des Menschen. Die Sprache abstrahiert immer mehr vom Aufbau der Maschine: problembezogene Hochsprachen.

Problemorientierte Sprachen sind solche, die geeignet sind, Probleme aus einem bestimmten Anwendungsbereich hardwareunabhängig zu formulieren. Man unterschied zwischen Sprachen für wirtschaftliche Anwendungen (COBOL) und für technisch-wissenschaftliche Anwendungen (FORTRAN). Typisch für Sprachen der dritten Generation sind die Möglichkeit zur Benennung von Variablen, statt Prozessorregister zu verwenden,

und zusammengesetze Datentypen (struct, array usw.). Programme in einer problemorientierten Sprache sollen auf beliebiger Hardware lauffähig sein (durch kompilieren); um in einer solchen Sprache zu schreiben, benötigt man keine Kenntnisse des Aufbaus des Computersystems. In der Praxis sind allgemeine Kenntnisse der Computerfunktion jedoch sehr hilfreich.

Liste der wichtigsten imperativen Progammiersprachen (daneben gibt es PROLOG für Prädikatenlogik und LISP als funktionale Programmiersprache für KI). Imperative Programmiersprachen sind solche, die zur Ausführung auf von-Neumann-Architektur geeignet sind.

FORTRAN Beginn der Entwicklung 1954 in den USA. Früher dauerte es 10-15 Jahre, bis sich eine Programmiersprache vom Entwurf bis zum flächendeckenden Einsatz durchsetzte. Diese Sprache ist geeignet für technisch-wissenschaftliche Berechnungen, z.B. Grafiksoftware. Fortran bietet die Möglichkeit zum Rechnen in komplexen Zahlen, eine Funktion, die in Assembler o.ä. natürlich überhaupt nicht angelegt ist.

COBOL Beginn der Entwicklung 1957 in den USA. COBOL hat sich bis heute als Programmiersprache für wirtschaftliche Anwendungen gehalten, v.a. für Großrechneranlagen.

ALGOL-60 Erster Ansatz der Europäer zur Entwicklung einer Programmiersprache, 1957 an der ETH in Zürich von Prof. Niklaus Wirth spezifiziert. Er hat die Sprachentwicklung wesentlich beeinflusst. ALGOL-60 war konzeptionell besser als FORTRAN / COBOL, hat sich jedoch nur etwas verbreitet, im Umfeld von Hochschule und Forschung.

PL/1 IBM begann 1960 mit der Entwicklung dieser Sprache, um FORTRAN und COBOL zu kombinieren. Diese Sprache hatte jedoch nie eine große praktische Bedeutung.

BASIC Ursprünglich gedacht zur Programmierung von Taschenrechnern. Heute noch eine sehr verbreitete Sprache.

Pascal 1960 von Prof. Wirth spezifiziert mit dem Ziel, einer Sprache, die von Anfängern leicht erlernbar sein sollte. Dem entspricht, dass Pascal heute noch viel in Schulen gelehrt wird und von vielen fachfremden Personen gelernt wird (E-Techniker für Steuerungen usw.). TURBO-Pascal ist eine objektorientierte Erweiterung von Pascal, hauptsächlich gedacht für PCs. Objekte kombinieren Methoden mit den zu diesem Anwendungsfall gehörenden Datentypen; man betrachtet Daten immer zusammen mit den darauf möglichen Operationen. Delphi basiert auf Pascal und wird heute verbreitet eingesetzt.

CHILL Eine Sprache, die im Bereich der Telekommunikation eingesetzt wurde und wird. Sie wurde vom CCITT spezifiziert als einem Standardisierungsgremium der TK-Industrie, abgelöst 1990 durch ITU-T. Grund der Entwicklung waren Schwächen in bisherigen Sprachen.

Pearl (nicht die Skriptsprache Perl) Eine Sprache, die für Automatisierungstechnik entwickelt wurde (SPS-Technik).

C Mit der Entwicklung wurde 1970 in den Bell Labs als Nachfolger der Sprache B begonnen. C hat eine Sonderstellung: von FORTRAN bis ADA wurde das Abstraktionsniveau immer mehr erhöht, C aber machte hier einen Rückschritt, indem es Zugriffe auf Prozessorregister usw. ermöglichte. Begründet war dies darin, dass mit C erstmals ein Betriebssystem nicht in Assembler programmiert werden sollte. C entstand quasi nebenbei, die erste Version war das noch nicht standardisierte Kerningham&Ritchie-C. Heute ist der Sprachstandard das ANSI-C. C war eine Sprache, die erstmals nicht mehr an eine bestimmte Anwendung gebunden ist.

ADA Als ein Raumfahrtprojekt gestoppt werden musste aufgrund eines Fehlers in einer Software, die in FORTRAN geschrieben wurde, begründet in syntaktischen Schwächen von FORTRAN, wurde in den USA 1975 mit der Entwicklung von ADA begonnen. Hauptsächlich unterstützt vom Militär und gedacht als Standardsprache für alle großen Projekte. ADA war eine gute, aber komplexe Sprache, so dass es lange dauerte, bis nach der Spezifikation überhaupt Compiler vorhanden waren. ADA wird heute noch eingesetzt bei Projekten, die große Anforderungen an Sicherheit stellen (Luftfahrt usw.).

SMALLTALK

SIMULA Eigentlich die erste objektorientierte Sprache, heute ohne Bedeutung.

C++ Objektorientierte Weiterentwicklung von C, ab 1980. Erhöhte des Abstraktionsniveau gegenüber C. Zur Zeit sind C und C++ die Sprachen, in denen am meisten neue Software entwickelt wird. Java hat zunehmende Bedeutung. In Zukunft wird wohl hardwarnahe Software (Treiber usw.) in C++ geschrieben werden, Anwendungssoftware in Java.

JAVA Interessant ist hierbei, dass Java auf einer virtuellen Maschine läuft.

9.1.4 Programmiersprachen der vierten Generation

Diese Definition ist nicht mehr konsitent. Manche fassen LISP als funktionale Programmiersprache der KI darunter, manche anwendungsorientiert abstrahierte »Sprachen« wie Excel und Datenbankabfragesprachen wie SQL.

10 Syntax und Semantik von Programmiersprachen

10.1 Syntax

Die Grammatik einer Programmiersprache - Regeln zur Kombination der lexikalischen Elemente, d.h. der Menge der Terminalsymbole. Programmiersprachen sind »künstliche Sprachen«; natürliche Sprachen sind zur Ausführung als Programm zu ungenau; oder anders: der Mensch ist in seiner Kommunikation sehr fehlertolerant, die Maschine beherrscht dies nicht.

10.2 Lexik

Sie bestimmt den Aufbau der Terminalsymbole (Grundelemente) der Sprachen aus Einzelzeichen und legt damit fest, welche Symbole (z.B.: `+`, `-`, `&&`) und Wortsymbole (Schlüsselwörter, z.B. `do`, `while`, `for`) in einer Programmiersprache verwendet werden dürfen.

10.3 Semantik

Die Semantik definiert, wie der Compiler die einzelnen Sprachelemente umzusetzen hat. Heute festgelegt in einem Sprachstandard bzw. eine Sprachbeschreibung, z.B. ANSI-C, um zu gewährleisten, dass die Compiler auch nach der Sprachdefinition gebaut werden, d.h. standardkonform sind. Die Definition von Sprachstandards soll die Entwicklung von Dialekten verhindern.

Die Semantik legt also die Bedeutung der Sprachkonstrukte fest. Z.B. bedeutet die syntaktisch richtige Formulierung (C-Code) `c=a+b;`: lade den Wert von `a`; lade den Wert von `b`; addiere `a` und `b`; weise den Wert des Ergebnisses `c` zu. Die Semantik kann mittlerweile auch ansatzweise (mathematisch) formal beschrieben werden statt wie hier in der Umgengssprache.

10.4 Pragmatik

Beschreibt die Eigenschaften einer Programmiersprache aus der subjektiven Sicht des Benutzers, wie Erlernbarkeit, Benutzbarkeit, Schwierigkeit, Lesbarkeit. Also können die Einschätzungen verschiedener Benutzer zum selben Objekt differieren.

11 Beschreibung der Syntax

Wie kann die Syntax einer Programmiersprache formal dargestellt werden? Die Syntax kann mit formalen Methoden definiert werden: BNF (Backus-Naur-Form), EBNF (erweiterte Backus-Naur-Form) oder Syntaxdiagramme. Pascal war die erste Sprache, die vollständig so definiert wurde. BNF und EBNF sind nicht hart standardisiert.

Eine Grammatik G ist ein 4-Tupel $G = (T, N, P, S)$ aus:

Terminalsymbole. Menge der Terminalsymbole T. Sie bilden das Alphabet der Programmiersprache, bestehend aus allen Zeichen und Schlüsselwörtern. Die Lexik definiert diese Menge T. Terminalsymbole werden unterstrichen oder fett geschrieben oder in " " gesetzt, z.B. um die Terminalsymbole im Pseudocode zu definieren.

Nichtterminalsymbole. Menge der Nichtterminalsymbole N. Die Konstrukte, die sich aus Terminalsymbolen zusammensetzen. Die Vereinigungsmenge von Nichtterminalsymbolen und Schlüsselwörtern[10] heißt »Vokabular« einer Sprache.

Produktionen. Regeln der Grammatik, wie aus Elementen von T und N neue Elemente von N gebildet werden können; genannt »Produktionen P«.

Startsymbol S. Das größtmögliche Nichtterminalsymbol, von dem ausgehend in einer Hierachie die Zusammensetzung aus jeweils einfacheren Symbolen gemäß den Produktionen P erklärt wird. In einer Programmiersprache ist das Startsymbol S das Programm.

Es gibt verschiedene Typen von Grammatiken; hier wird nur eine kontextfreie Grammatik behandelt.

11.1 Erweiterte Backus-Naur-Form

EBNF ist eine Metasprache, d.h. eine Sprache, die geeignet ist, eine andere Sprache zu beschrieben (wie auch XML). Die Sprachelemente einer Metasprache heißen Metasymbole. Jede Programmiersprache hat eine Definition in EBNF; die Nichtterminalsymbole größerer Grammatiken werden sowohl hierachisch entsprechend der Definition in EBNF als auch alphabetisch geordnet, um die Navigation zu erleichtern. Um die Beschreibung der Grammatik durch EBNF übersichtlich zu halten, verwendet man neu eingeführte Nichtterminalsymbole, die dann in einem weiteren Schritt definiert werden, statt die ganze Grammatik in einer Zeile darzustellen. Die EBNF ist hauptsächlich zur Auswertung per EDV gedacht, im Unterschied zur eher menschenlesbaren Notation der Grammatik in Syntaxdiagrammen.

Startymbol Ein Nichtterminalsymbol, das auch entsprechend dargestellt wird.

Terminalsymbole Werden unterstrichen oder in Anführungszeichen dargestellt.

Nichtterminalsymbole Werden in `< >` dargestellt, z.B. `<programm>`.

Produktion Auf der linken Seite steht das durch die Produktion zu erklärende Nichtterminalsymbol, dann folgt ein Gleichheitsszeichen `=` und rechts ein EBNF-Ausdruck (auch »Erzeugung«) und abschließend ein Punkt. Dass in der Produktion links nur ein Nichtterminalsymbol stehen darf, klassifiziert die Grammatik nach Chomsky-Klassifikation als Typ-2-Grammatik (d.h. kontextfreie Grammatik). Typ-2-Grammatiken können durch Kellermaschinen ausgeführt werden, Typ-3-Grammatiken (reguläre Ausdrücke) durch endliche Automaten. Mit Typ-3-Grammatiken kann die Lexik einer Grammatik (Zusammensetzung der Schlüsselwörter) definiert werden. Die Symbole `=`, `.`, `< >` sind Beispiele für terminale Metasymbole der EBNF, `<EBNF-Ausdruck>` und `<Metabezeichner>` sind Beispiele für ein nichtterminale Metaysmbole der EBNF. Liste der terminalen Metasymbole in EBNF[11]:

Metasymbol	Bedeutung
`=`	**»ist definiert als»**
`|`	**Logisches ODER**
`.`	**Ende der Produktion**
`[x]`	**optionales (0- oder 1-maliges) Auftreten des Symbols `x` der beschriebenen Sprache**
`{x}`	**n-malige Wiederholung von `x` ($n \geq 0$)**
`(x|y)`	**Zusammenfassung zu einer Einheit wie () in der Mathematik** **Hier: »genau eine Alternative `x` oder `y`«**
`"XYZ"` **oder** <u>**XYZ**</u>	**das Terminalsymbol <u>XYZ</u> der beschriebenen Sprache**
`<XYZ>`	**das Nichtterminalsymbol `<XYZ>` der beschriebenen Sprache**

[10] Schlüsselwörter: Die Menge der Terminalsymbole ohne die Menge der Zeichen.

[11] Die Zeichen x, y, XYZ in der Tabelle gehören nicht zu den terminalen Metasymbolen der EBNF, sondern sind (Nicht)-Terminalsymbole der beschriebenen Sprache, durch die die Metasymbole zu Beispielen ergänzt werden. Die Bedeutung dieser Beispiele wird in der folgenden Spalte erklärt.

Beispiel: EBNF eines vereinfachten deutschen Satzes Es ergibt sich eine absteigende, immer genauer definierende Produktionenhierachie; so enthält die Produktion von `<satz>` das Nichtterminalsymbol `<subjekt>`, dessen Produktion wiederum das Nichtterminalsymbol, `<eigenname>` usw.

```
<satz> = <subjekt> <prädikat> [<objekt>].
// <satz> ist also das Startsymbol
<subjekt> = <eigenname> | <artikel_und_substantiv>.
<prädikat> = <verb>.
<objekt> = <artikel_und_substantiv>.

<eigenname> = "Adam" | "Eva".
// genau eine Alternative verwenden!
<artikel_und_substantiv> = <artikel><substantiv>.
<artikel> = "der" | "die" | "den".
<substantiv> = "Frucht" | "Schlange".
<verb> = "isst" | "beißt".
```

Beispiel: Beschreibung der EBNF-Syntax durch EBNF Dieser Abschnitt korrigiert den entsprechenden Abschnitt in [3]: Teil »Programmiersprachen und -systeme«, Kapitel 3.1, S.7, Beispiel 3.

```
<Syntax>=<Produktion>{<Produktion>}.
<Produktion>=<Nichtterminalsymbol>"="<Ausdruck>".".
<Ausdruck>=<Alternative>{"|"<Alternative}.
<Alternative>=<syntaktischesKonstrukt>{<syntaktischesKonstrukt>}.
<syntaktischesKonstrukt>=<Nichtterminalsymbol>|<Terminalsymbol>|
   "("<Ausdruck>")"|"["<Ausdruck>"]"|"{"<Ausdruck>"}".
// gemeint sind hier Terminalsymbole der im Einsatz
// mit EBNF beschriebenen Programmiersprache, nicht die
// von EBNF selbst.
<Terminalsymbol>="""<Zeichen>{<Zeichen>}""".
<Zeichen>=<ZeichenAusDemAlphabetDerProgrammiersprache>.
// dieses Nichtterminalsymbol muss für jede Programmiersprache
// separat definiert werden, nicht allgemein möglich.
<Nichtterminalsymbol>="<"<Buchstabe>{<Buchstabe>|<Ziffer>}">".
<Buchstabe>="a"|"A"|...|"z"|"Z".
<Ziffer>="0"|"1"|...|"9".
```

In diesem Beispiel wurde `<Ausdruck>` rekursiv definiert, d.h. letztlich mit sich selbst erklärt, vgl. die Produktion `<syntaktischesKonstrukt>`. In einer Klausur kann z.B. eine Aufgabe sein, herauszufinden, ob gegebene Konstrukte nach gegebenen Produktionen syntaktisch korrekt gebildet wurden.

Beispiel: das Metasymbol `(a|b)` der EBNF am Beispiel der PASCAL-Syntax

```
<Anweisung>=[<Marke>](<einfacheAnweisung>|<strukturierteAnweisung>).
```

Die runden Klammern der EBNF werden genauso wie mathematische Klammern verwendet, einfach zur Zusammenfassung verschiedener Elemente. Der Inhalt dieser Klammern ist ein beliebiger EBNF-Ausdruck (vgl. Definition der EBNF-Syntax oben), zum Beispiel auch `(<a>|<b>|<c>)`. Gemeint ist in obigem Beispiel: einer optionalen Marke folgt entweder eine einfache oder eine strukturierte Anweisung, im Unterschied zu folgender Syntax, wo eine Anweidung nicht aus einer Marke und einer strukturierten Anweisung bestehen könnte:

```
<Anweisung>=[<Marke>]<einfacheAnweisung>|<strukturierteAnweisung>.
```

11.2 Syntaxdiagramme

Eine Liste der Metasymbole in Syntaxdiagrammen mit ihren Äquivalenten in EBNF findet sich in [3] im Teil »Programmiersprachen und Programmiersysteme«, Kapitel »3.2 Syntaxdiagramme«, S. 7.

Syntaxdiagramme sind mathematisch betrachtet gerichtete Graphen. Sie dürfen nur in Pfeilrichtung durchlaufen werden. In der Klausur kann zum Beispiel eine Aufgabe sein, von EBNF zu Syntaxdiagrammen zu

übersetzen: beide Beschreibungsformen sind gleichwertig. Das in einer EBNF-Produktion definierte Nichtterminalsymbol wird in einem Syntaxdiagramm zum Titel des Diagramms. In Syntaxdiagrammen enthalten Kreise oder Ovale jeweils Terminalsymbole, die in EBNF durch Anführungszeichen oder Unterstreichen dargestellt wurden.

12 Werkzeuge zur Programmierung[12]

Die meisten Algorithmen muss man sich nicht ausdenken, sondern kann in einem Buch über Standardalgorithmen nachschlagen (Lösung eines Gleichungssystems etc.). Die Übersetzung aus einer problemorientierten Sprache in Maschinensprache geschieht maschinell. Dazu gibt es zwei Möglichkeiten:

12.1 Übersetzung mittels Interpreter

Das Interpreterprogramm wiederholt folgenden Zyklus bis zum Programmende: übersetze die nächste Anweisung aus der höheren Programmiersprache in Maschinensprache; veranlasse die Ausführung dieser Anweisung durch das Rechensystem. Interpreter sind vor allem in der Testphase eines Programms interessant, als eine Art maschinell durchgeführter Schreibtischtest. Nachteilig ist die langsame Ausführungsgeschwindigkeit beim Einsatz des Programms.

12.2 Übersetzung mittels Compiler

Das Compilerprogramm übersetzt ein Programm aus einer höheren Programmiersprache (über den Zwischenschritt der Assemblersprache) einmalig in Maschinensprache, die vom Mikroprozessor interpretiert und ausgeführt wird. Für jeden Mikroprozessor benötigt man einen Compiler, der eine bestimmte Hochsprache an den jeweiligen Mikroprozessor anpasst, d.h. die richtige Maschinensprache kompiliert.

Java hat demgegenüber ein anderes Konzept: Der Compiler erzeugt den Byte-Code, eine Art Assembler für einen nichtexistenten Mikroprozessor. Auf dem Mikroprozessor gibt es nun eine virtuelle Maschine (die für jeden Mikroprozessortyp verschieden ist), d.i. ein Interpreter, der den Byte-Code in Maschinensprache übersetzt. Somit sind Java-Programme im Byte-Code-Format auf jedem Prozessor ausführbar, können also in diesem Format ausgetauscht werden (deshalb funktionieren Java-Anwendungen im Internet auf jedem System!). Bei einer klassischen Hochsprache dagegen muss man ein Programm für jeden Mikroprozessor extra kompilieren, so dass ein Austausch im Binärformat zwischen verschiedenen Prozessortypen nicht möglich ist, also es keine Alternative zu Java-Applets gibt. Der Nachteil des Konzepts von Java ist die Geschwindigkeit, weil der Byte-Code eben durch einen Interpreter umgesetzt wird.

Programme, die in Maschinensprache vorliegen, heißen Objektprogramm.

12.3 Verfahren der Softwareproduktion von der Problembeschreibung zum Zielprogramm

1. Problembeschreibung.
   - Ziel: Programmspezifikation.
2. Modellbildung.
   - Ziel: Modell der Problemlösung in Form von Algorithmen und Datenstrukturen. Zur Notation der Algorithmen werden dabei Pseudocode oder Flussdiagramme verwendet.
3. Programmieren und Editieren.
   - Werkzeug: Editor
   - Ziel: Quellprogramm.
4. Einfügen von Quellmodulen, Übersetzen. Quellmodule sind Bibliotheken oder andere fremde Software, um nicht alles selbst programmieren zu müssen.

[12] Im WS 2001/2002 wurde in der Vorlesung aus [3], Teil »Progrmmiersprachen und -systeme«, Kapitel »Werkzeuge zur Programmierung« nur das Diagramm »Bild 3 Arbeitsschritte von der Problembeschriebung zum Zielprogramm« besprochen. Mehr wird hiervon also auch in der Klausur nicht vorausgesetzt.

- Werkzeug: Compiler
- Führt zum Objektprogramm

5. Einfügen von Objektmodulen, Binden der Objektprogramme.
   - Werkzeug: Binder (engl. linker)
   - Ziel: Lademodul (ein ausführbares Programm)
6. Programmausführung und Test
   - Werkzeuge: Lader, Debugger
7. Zielprogramm

13 Kontrollfragen

Teil IV
Anhang

14 Kodetabellen

15 Operationstafeln für Dual-, Oktal- und Hexadezimalsystem

Teil V
Übungen

16 Algorithmen

Dies sind die Lösungen zu den offiziellen Übungsaufgaben [4].

16.1 Aufgabe 1

16.2 Aufgabe 2

16.3 Aufgabe 3

16.4 Aufgabe 4

16.5 Aufgabe 5

```
min:=a;
max:=a;
Falls b>max   // b>a
dann  max:=b;
Falls c>max   // c>max(a,b)
dann  max:=c;
Falls d>max   // d>max(a,b,c)
dann  max:=d;
Falls b<min   // b<a
dann  min:=b;
Falls c<min   // c<min(a,b)
dann  min:=c;
Falls d<min   // d<min(a,b,c)
dann  min:=d;
```

16.6 Aufgabe 6

16.7 Aufgabe 7

16.8 Aufgabe 8

16.9 Aufgabe 9

16.10 Aufgabe 10

17 EBNF und Syntaxdiagramme

Dies sind die Lösungen zu den offiziellen Übungsaufgaben [4].

17.1 Aufgabe 1

17.2 Aufgabe 2

17.3 Aufgabe 3

18 Informationsdarstellung im Rechner

Dies sind die Lösungen zu den offiziellen Übungsaufgaben [4].

18.1 Aufgabe 1

BSC »binary synchronous communication«, zeichenorientiertes synchrones Kommunikationsprotokoll

Bedeutung der gegebenen Zeichenfolge im BSC-Protokoll:

DEL PAD; Rahmenbeginn

SYN SYN

SOH Start of Header

Kopf der Text im Header

DLE Data Link of Escape

STX Start of Text

Text der übertragene Text

DLE Data Link of Escape

ETX End of Text

BCC sog. block check character

DEL PAD; Rahmenende

18.2 Aufgabe 2

18.3 Aufgabe 3

18.4 Aufgabe 4

Die Durchführung der Addition im Binärsystem ist hier nicht gefordert, sondern es soll nur entsprechend der Darstellung im Rechner, aber im Dezimalsystem, gerechnet werden. Das heißt: alle Zahlen z müssen durch $z' = z \bmod 256$ in ihre Äquivalente im Restklassenring Z_{256} umgewandelt werden (siehe Kapitel 7.3.3) und es wird in diesem Restklassenring gerechnet. Beispiele:

$$\begin{aligned} 100 \bmod 256 + 27 \bmod 256 &= 127 \bmod 256 \\ \Rightarrow 100 + 27 &= 127 \end{aligned}$$

$$\begin{aligned} 100 \bmod 256 + 127 \bmod 256 &= 227 \bmod 256 \\ \Rightarrow 100 + 127 &= 227 \end{aligned}$$

Die Rückabbildung der Restklasse auf die Zahlen $-128 \ldots 127$ liefert hier -29 als Ergebnis, denn $-29 \bmod 256 = 227$.

18.5 Aufgabe 5

19 Zahlendarstellung und Zahlenumwandlung

Dies sind die Lösungen zu den offiziellen Übungsaufgaben [4].

19.1 Aufgabe 1

19.2 Aufgabe 2

19.3 Aufgabe 3

Diese Aufgabe war einmal eine Klausuraufgabe! Stoffliche Grundlagen, die zur Bearbeitung dieser Aufgabe nötig sind, stehen in Kapitel 7.2.3. Welche Werte nun darf eine Zahl haben, damit nach Rundung ihrer fünften Nachkommastelle[13] die Zahl $0,3281$ (geforderte Genauigkeit von vier dezimalen Nachkommastellen) entsteht?

- minimal $Z_{min} = 0,32805 \approx 0,3281$
- maximal $Z_{max} = 0,32814 \approx 0,3281$

Die Differenz beträgt[14]:

$$\Delta = Z_{max} - Z_{min} = 0,00009 \geq \varepsilon \leq 2^{-n+1}$$

Der maximale Wert der vernachlässigten Nachkommastellen $(n+1) \ldots \infty$ der Binärzahl (nämlich $\varepsilon = 2^{-n+1}$) muss nun kleiner oder gleich Δ werden. Wieviele Nachkommastellen $n \in \mathbb{N}$ müssen dazu berechnet werden?

$$\begin{aligned} 2^{-n+1} &\leq 0,00009 \quad |: 2 \\ \Leftrightarrow 2^{-n} &\leq 0,0000045 \quad | \, ld\,() \\ \Leftrightarrow -n &\leq -14,43 \quad | \cdot (-1) \\ \Leftrightarrow n &\geq 14,43 \\ \Rightarrow n &= 15 \text{ da } n \in \mathbb{N} \end{aligned}$$

Die Berechnung dieser 15 Nachkommastellen erfolgt am schnellsten über den Zwischenschritt des Hexadezimalsystems.

19.4 Aufgabe 4

Diese Aufgabe war einmal eine Klausuraufgabe!

19.5 Aufgabe 5

Welcher Wert ergibt sich, wenn man versucht, die Zahl 355 in einer Variable vom Typ `unsigned char` (1 Byte Breite) zu speichern?

Da im Computer alle Zahlen durch ihre Äquivalente im Restklassenring Z_m (hier mit $m = 2^8 = 256$) gespeichert werden (siehe Kapitel 7.3.3), ergibt sich der Wert

$$355_{10} \bmod 256_{10} = 99_{10}$$

Anschaulich entspricht dies dem Abschneiden von $355_{10} = 101100011_2$ auf 8 Bit Länge:

$$01100011_2 = 99_{10}$$

[13] wodurch die fünfte und alle folgenden Nachkommastellen wegfallen

[14] Eigentlich: $Q_Q = Q_Z + \Delta \Leftrightarrow \Delta = Q_Q - Q_Z$, hier wäre das $\Delta = Q_Q - Q_{Z_{min}}$. In der Vorlesung wurde jedoch die im Text folgende Formel angewandt, was genaugenommen falsch ist.

19.6 Aufgabe 6

19.7 Aufgabe 7

19.8 Aufgabe 8

Teil VI
Die Klausur

20 Allgemeines

Die Klausur gliedert sich in:

Teil A: Dieser ist ähnlich wie die Übungen aufgebaut. Stoff:

- Struktugramme
- Flussdiagramme
- EBNF
- Zahlendarstellung

Es dürfen alle Hilfsmittel (eigene Unterlagen, Taschenrechner, Skript, alte Klausuren mit Lösungen usw.) benutzt werden, jedoch mit diesen Einschränkungen:

- nur ein Lehrbuch
- kein Handy

Teil B: Der reprodukive Teil, in dem keine Hilfsmittel benutzt werden dürfen.

20.1 Tipps zur Klausur

1. Besonders wenn Zahlen nach der Konvertierung für weitere Rechnungen verwendet werden, sollte man die Konvertierung mit dem Taschenrechner durchführen oder mindestens damit nachrechnen, sonst ergeben sich viele Folgefehler.
2. Wissen, wie man die Funktionen zur Konvertierung zwischen Zahlensystemen auf dem Taschenrechner bedient.
3. Die Klausur enthält oft im Teil B eine Aufgabe, in der eine gegebene und in der Vorlesung besprochene Grafik zu beschriften ist, z.B. die Prinzipaufteilung eines Prozessors oder die Funktionsweise eines Bussystems. Man sollte sich daher die Grafiken des Skriptes gut einprägen!
4. Um die Richtigkeit einer Aufgabe wie »Führen sie die Multiplikation $2A,5E_{16} \cdot 1,C_{16}$ aus.« zu überprüfen, wenn der Taschenrechner nur ganzzahlige Multiplikation im Hexadezimal-System unterstützt, multipliziere man einfach unter Vernachlässigung des Kommas. Die Zahlen des Ergebnisses müssen identisch sein, man muss nur noch die Kommaposition richtig setzen.

21 Klausur 2002-01-18

Diese Angabe des Stoffes der Klausur in »Grundlagen der Informatik« wurde am Tag der Klausur nach eigener Erinnerung geschrieben. Die Lösungen durften nur auf die ausgeteilten Aufgabenblätter und deren Rückseiten geschrieben werden. Diese waren zusammengeheftet und durften nicht getrennt werden; so war der Übertrag von Teilergebnissen auf das Folgeblatt zeitraubend. Diese Klausur unterscheidet sich wesentlich von den in der Fachschaft erhältlichen alten Klausuren: der Teil A (Rechnen) zählt nun $\frac{2}{3}$ statt $\frac{1}{2}$ und die Zeit ist knapper bemessen.

21.1 Teil A: Rechnen

Zeit: 60 Minuten. Dies war recht knapp bemessen. Wertung: ca. $\frac{2}{3}$.

1. Es waren zu zwei Kontrollkonstrukten (zweistufige `If`-Schachtelunen) Struktugramme zu zeichnen und es musste über Zusicherunegn begründet und dann entschieden werden, ob Vereinfachungen möglich sind.
2. Es war für vier einfache Konstrukte entweder die EBNF oder das Syntaxdiaramm zu finden.
3. Es war das Syntaxdiagramm für eine »vorzeichenloseZahl« in Pascal in EBNF umzusetzen. Die Lösung war ungefähr:

   ```
   <vorzeichenloseZahl>=<vorzeichenloseIntZahl> [<Nachkommateil>] [<Exponentteil>].
   <Nachkommateil>= "." <vorzeichenloseIntZahl>.
   <Exponentteil>= "e" [("+" | "-")] <vorzeichenloseIntZahl>.
   ```

4. Es waren zwei Hexadezimalzahlen a und b als 16-stellige Binärzahlen darzustellen als a, b, $-a$, $-b$ und anschließend die Operationen $a + b$, $a + (-b)$, $(-a) + b$ und $(-a) - (-b)$ im Dualsystem durchzuführen. Die Rechnung war in vorgedruckten Kästchen auszuführen. Bei Überläufen und negativen Ergebnissen war mit kurzer Begründung anzugeben, welches dezimale Ergebnis der Rechner ausgibt.
5. Es war ein gegebenes Bitmuster im Litte Endian Format als short real zu interpretieren. Der Wert sollte in der Form »Vorzeichen, Vorkommastellen, Nachkommastellen« im Dualsystem, Hexadezimalsystem und Dezimalsystem angegeben werden. Die Umwandlung des Nachkommateils ins Dezimalsystem sollte nach dem Horner-Schema durchgeführt werden.
6. Es war anzugeben, welche dezimale Genauigkeit mit den 23 Nachkommastellen im Format short real erreicht werden kann.
7. Es war eine Multiplikation von zwei Hexadezimalzahlen (etwas $CA,16_{16} \cdot 1,C_{16}$) schriftlich nachvollziehbar durchzuführen.

21.2 Teil B: Reproduktion

Zeit: 30 Minuten. Die meisten waren nach 20 Minuten fertig. Wertung: ca. $\frac{1}{3}$.

1. Zeichnen Sie die Pfeile für Datenflüsse zwischen den Busteilnehmern und den Teilbussen.
   Dies entspricht der Vervollständigung der Abbildung in [3], Teil: »Technische Grundlagen | Einführung in die Rechnertechnik | Systembus | Datenflüsse«.
2. Erläutern Sie kurz die Aufgaben von Adress- und Datenbus.
3. Die folgende Abbilduing zeigt den schematischen Aufbau der Prozessverwaltung durch ein Betriebssystem (es folgte die Abbildung aus [3], Teil »Technische Grundlagen | Betriebssysteme | Grundbegriffe | Prozessverwaltung«. Erläutern Sie die Aufgaben von:

   - Dispatcher
   - Scheduler
   - Auftragswarteschlange
   - Prozesswarteschlange
   - Zeituhr

   Hier war eine umfangreichere Bearbeitung verlangt.
4. Ergänzen Sie die Lücken in folgenden Sätzen:

   - Das Fachgebiet »Programmiersprachen und Übersetzer« gehört zur (ergänze: Praktischen) Informatik.
   - Die Fachgebiete »Netz- und Schaltwerksentwurf« und »Rechnernetze« gehören zur (ergänze: Technischen) Informatik.
   - Über die Zuteilung der Busleitungen an einen sendewilligen Teilnehmer entscheidet der: (ergänze: Arbiter).
   - Die Codierung der auszuführenden Operation in einem Maschinenbefehl heißt: (ergänze: Opcode).

Literatur

[1] »Lehr- und Übungsbuch Informatik Bd.1 - Grundlagen und Überblick«.; 59,80DM; Prof. Dr. Wolfgang Schmitt hat in diesem Buch ein Kapitel geschrieben; dieses Buch passt auch am besten zu dieser Vorlesung. Wer dieses Buch erwerben möchte, kann mit einem von Prof. Schmitt ausgestellten Hörereschein zur Vorlesung 20% Rabatt bekommen.

[2] »Lehr- und Übungsbuch Informatik Bd.1 - Grundlagen und Überblick«. Kap.10: Die Entwicklung der Computer.

[3] Prof. Dr. W. Schmitt: »Hilfsblätter zur Vorlesung Grundlagen der Informatik WS 01/02«. Dies ist das offizielles Skript zur Vorlesung »Grundlagen der Informatik« im WS 2001/2002 bei Professor Schmitt, zu dem die vorliegende studentische Mitschrift eine Ergänzung ist. Es wurde in der ersten Übung zur Vorlesung ausgeteilt, zusammen mit Quelle [4]. Dieses Skript ist leider nicht digital erhältlich und darf aus urheberrechtlichen Gründen auch nicht im Internet veröffentlicht werden, weil es ein Manusskript zu [1] enthält. gestellt werden.

[4] Prof. Dr. W. Schmitt: Übungsblätter zur Vorlesung »Grundlagen der Informatik« an der FH Gießen-Friedberg, Studiengang Informatik, erstes Fachsemester. Datenstand 2000-01-04 (neuester Teil). Bestehend aus folgenden Teilen: »Algorithmen«, »EBNF und Syntaxdiagramme«, »Informationsdarstellung im Rechner«, »Zahlendarstellung und Zahlenumwandlung«.

[5] Ernst Otto Schäfer: Studentischer Mitschrieb zur Vorlesung »Grundlagen der Informatik« bei Prof. Dr. W. Schmitt an der FH Gießen-Friedberg, zu Kapitel . Aus dem Wintersemester 2000/2001. Vielen Dank!

[6] Benjamin Hebbel: Studentischer Mitschrieb zur Vorlesung und Übung »Grundlagen der Informatik« bei Prof. Dr. W. Schmitt an der FH Gießen-Friedberg im Wintersemester 2001/2002 bis inkl. (Do) 2001-12-13. Vielen Dank!