Zahlensysteme in der Computermathematik

Univ. Doz. Mag. Dr. Wolfgang Herfort

Institut für Angewandte und
Numerische Mathematik
Technische Universität Wien

Kapitelübersicht

1 Einleitung

Im vorliegenden Bericht soll ein Einblick in Zahlensysteme im Zusammenhang mit Computerarithmetik gegeben werden. Vorallem die Behandlung einiger typischer Fragen im Zusammenhang mit der Implementation von Grundrechnungsarten in *B-adischen Zahlsystemen* steht im Vordergrund, weil danach die Erweiterung der Rechenoperationen auf Gleitkommazahlen bzw. symbolisch dargestellte algebraische Zahlen möglich ist. Es werden durchaus viele Fragen offenbleiben, für deren Beantwortung ich auf entsprechende Fachliteratur hinweisen darf.

Es erschien mir wertvoll, (nicht ganz im Einklang mit dem Titel) auch geschichtliche Bezüge über die bekannteren Zahlensysteme aufzugreifen. Für Ziffernsysteme komplexer Zahlen siehe [Tr94].

2 Grundtypen von Zahl– und Ziffernsystemen

2.1 B–adische Systeme

Die heute übliche Darstellung reeller Zahlen als Dezimalzahlen, etwa der Zahl

$$\pi := 3,14159\ldots$$

im Dezimalsystem wird unter Zuhilfenahme von Reihen rationaler Zahlen in der Form

$$\pi = 3 \times 10^0 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \ldots$$

verstanden, wobei man üblicherweise davon ausgeht, daß:

- π als reelle Zahl aufgefaßt wird und reelle Zahlen als Vervollständigung der rationalen Zahlen bekannt sind;
- die Theorie der unendlichen, konvergenten Reihen zur Verfügung steht;
- ein Algorithmus zur Erzeugung der *Ziffernentwicklung* der Zahl π zur Verfügung steht;
- Rundungskonventionen getroffen wurden, derenthalben das Symbol "..." zum Beispiel dahingehend gelesen wird, daß

  $$0 \leq \pi - 3,14159$$

  gilt, und die letzte Stelle in $3,14159$ eine gesicherte Dezimale ist, also selbst bei Bekanntgabe nachfolgender Dezimalen und Runden auf diese Stelle aus der '9' keine '0' und der davorkommenden '5' keine '6' werden kann.

In Verallgemeinerung des *dekadischen* Zahlsystems ergibt sich der Begriff B–adisches Zahl- und Ziffernsystem. Eine formale Definition lautet folgendermaßen. Ein B–adisches Ziffernsystem ist ein Paar $(B, \mathcal{D})$ wobei:

- B ist eine natürliche Zahl größer als 1, die *Basis*.
- $\mathcal{D}$ ist eine endliche Menge von Symbolen (den *Ziffern*). Häufig ist es üblich, die Zahlen $0, 1, \ldots, B-1$ zu wählen.

- Jedem reellen x wird schließlich eine B-adische Darstellung der Form

$$x = \sum_{j=-\infty}^{k} a_j B^j$$

zugewiesen, wobei $a_j \in D$ und k eine passende natürliche Zahl ist. Man verlangt die Konvergenz dieser Reihe.

Es hat sich im Zusammenhang mit Diskussionen über Zahlsysteme die Kurznotation

$$n = (a_k a_{k-1} \cdots a_0, a_{-1} a_{-2} \ldots)_B$$

eingebürgert, wobei ein Kommazeichen ',' verwendet wird.

Beispiele zur Anwendung dieser Notation sind die folgenden:

1. *Dekadisches System.* Es gelten $B := 10$, $D := \{0,1,2,3,4,5,6,7,8,9\}$.

   Zum Beispiel ist dann $125 = (125)_{10} = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$.

2. Im *Binärsystem* gelten $B := 2$, $D := \{0,1\}$.

   Zum Beispiel ist dann $(125)_{10} = (111101)_2$.

3. Im *Oktalsystem* gelten $B := 8$ und $D := \{0,1,2,3,4,5,6,7\}$.

   Zum Beispiel ist dann $(125)_{10} = (175)_8$

4. Im *Hexadezimalsystem* ist $B = 16$ und

   $D = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$.

   Dann ist zum Beispiel $(447)_{10} = (1BF)_{16}$.

5. Im *Sexagesimalsystem* gilt $B := 60$ und

$$D := \{[0],[1],\ldots,[59]\},$$

   wobei in diesem Beispiel deutlichkeitshalber eckige Klammern um die *Ziffernsymbole* gesetzt wurden.

   Es ist dann etwa $1/3 = (0,[2])_{60}$.

2.2 Polyadische Systeme

Eine strikte Verallgemeinerung der B-adischen Systeme stellen die *polyadischen* Entwicklungen dar, wobei hier nur die Darstellung natürlicher Zahlen beschrieben werden soll:

- Es sei

$$\{b_k\}_{k=0}^{\infty}$$

eine unendliche Folge natürlicher Zahlen, jede größer als 1.

- Es sei

$$\{D_k\}_{k=0}^{\infty}$$

eine unendliche Folge gewisser endlicher Mengen natürlicher Zahlen (*Ziffernmengen*), wobei meist

$$D_k := \{0, \ldots, b_k - 1\}$$

gewählt wird.

- Einer natürlichen Zahl n wird die Darstellung

$$n = a_0 + a_1 b_0 + a_2 b_0 b_1 + a_3 b_0 b_1 b_3 + a_{m-1} b_0 b_1 \cdots b_m$$

zugewiesen, wobei m eine geeignete natürliche Zahl ist und $a_j \in D_j$ für $j \in \mathbb{N}$.

Das B-adische System erscheint als Spezialfall, wenn man die Folge $b_k := B$ und $D_k := D$ setzt.

Als Beispiele polyadischer Systeme werden in diesem Artikel Ziffernsysteme der Babylonier, Ägypter und Griechen behandelt. Viele Maß und Gewichtssysteme, aber auch Währungseinheiten lassen polyadische Struktur erkennen, so auch das österreichische.

2.3 Gleitkommazahlen

Im naturwissenschaftlich, technischen Gebrauch hat sich die Darstellung reeller Zahlen in der Form

$$m_P \approx 1{,}673 \times 10^{-24} Gramm$$

(Masse des Protons) herausgebildet. Diese Zahl wird somit durch die *Mantisse* 1,673 und den *Exponenten* -24 beschrieben, und stellt ein Beispiel einer *Gleitkommazahl* dar. Freilich wird das dekadische Zahlsystem mitbenützt.

Der Grund für diese Darstellungsart ist in seiner besseren Einsichtigkeit der Größenordnung der entsprechenden Zahl zu suchen. Als großer Nutzen hat sich jedoch die Verwendbarkeit von Logarithmen zur besseren Implementation der Multipliakation in Rechenautomaten erwiesen (*skalierte Multiplikation*), ein Gedanke, der schon in der babylonischen Mathematik zu finden ist.

2.4 Symbolische Darstellungen

In der Computer-Algebra werden Elemente in $\mathbb{Q}$ als geordnete Paare *(Zähler,Nenner)* geschrieben. Diese symbolische Schreibweise faßt rationale Zahlen als durch Verhältnisse ganzer Zahlen gegeben auf, genau wie man es in jeder Grundvorlesung hört. Üblicherweise werden solche Zahlenpaare *normiert* (teilerfremd, Nenner positiv), um die Eindeutigkeit der Symbole zu gewährleisten. Die Schreibweisen $\sqrt{2}$, $\sqrt[3]{7}$, etc. stellen eine *symbolische* Beschreibung algebraischer Irrationalitäten dar. Das symbolische Rechnen in algebraischen Körpern ist ein Zweig der Computeralgebra.

3 Einige Zahlensysteme der Antike

3.1 Sexagesimalsystem in Babylon um 3000 v. Chr.

Es soll die grundlegend *polyadische* Struktur dieses Ziffernsystems aufgezeigt werden. Die in der babylonischen Mathematik gepflegte Schreibweise des Sexagesimalsystems (Basis $B := 60$) benützt für die Zahlen $0, 1, \ldots, 59$ das Dezimalsystem (die Tatsache, daß die Null lediglich als Lücke, bzw. Lückenzeichen benützt wurde, soll hier nicht interessieren). Nun wähle man die oben zitierten Folgen:

$$b_0 := 10, b_1 := 6, b_2 := 10, b_3 := 6, \ldots,$$

allgemein

$$b_k := \begin{cases} 10, & \text{falls } k \text{ gerade ist} \\ 6, & \text{falls } k \text{ ungerade ist.} \end{cases}$$

Als Folge der Ziffernmengen wählt man:

1. Falls k gerade ist, so wählt man

$$Z_k := \{0, 1, \ldots, 9\}.$$

(Es wurden mit einem flach gehaltenen dreikantigen Hölzchen senkrechte Kerben dieser Anzahl, bzw. keine Kerbe für 0 in die Tontafel gedrückt).

2. Falls k ungerade ist, so wählt man etwa (deutlichkeitshalber überstrichen)

$$Z_k := \{\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}\}.$$

(Es wurden mit dem fast stehenden Hölzchen bis zu fünf Winkelhaken eingedrückt, jeder mit dem *Wert* 10. Die Null wurde nicht berücksichtigt.)

3. In moderner Form (unter Benützung von Nullen) steht dann eine Zahl der Form

$$n := (\bar{5}\bar{9}\bar{3}1\bar{0}\bar{0}\bar{2}1)$$

für

$$\begin{aligned} n = 1 &+ \bar{2} \times (10) \\ &+ 0 \times (10 \times 6) \\ &+ \bar{0} \times (10 \times 6 \times 10) \\ &+ 1 \times (10 \times 6 \times 10 \times 6) \\ &+ \bar{3} \times (10 \times 6 \times 10 \times 6 \times 10) \\ &+ 9 \times (10 \times 6 \times 10 \times 6 \times 10 \times 6) \\ &+ \bar{5} \times (10 \times 6 \times 10 \times 6 \times 10 \times 6 \times 10). \end{aligned}$$

Die hier vorgestellte Schreibweise zeigt, daß der Begriff *polyadisches Zahlsystem* der babylonischen Schreibweise gerecht wird. Dennoch wird in der Literatur bequemlichkeitshalber von einem Hexadezimalsystem (Basis $B = 60$, $D := \{0, \ldots, 59\}$ gesprochen.

Die oben dargestellte Zahl ist dann (im Sexagesimal- bzw. Dezimalsystem)

$$n = (59, 31, 0, 21)_{60} = 59 \times 60^3 + 31 \times 60^2 + 21 = (12\,855\,621)_{10}.$$

Der Beistrich wurde diesmal lediglich als Trennzeichen zwischen den gelegentlich zweistelligen *Ziffern* benützt.

3.2 Hieroglyphenzahlen in Ägypten um 3000 v. Chr.

Sehr deutlich wird die *polyadische* Struktur des zunächst scheinbar dekadischen Ziffernsystems in der Hieroglyphenschrift (Ägypten, ca 3000 v. Chr.). Dabei sollen nur Zahlen bis $(9\,999\,999)_{10}$ dargestellt werden. Man wählt:

1. $b_k := 10$. Insofern gibt es keine Abweichung vom dekadischen System.

2. Es gab jedoch *verschiedene* Ziffernmengen, welche von der jeweiligen Zehnerpotenz abhängig waren.

$$
\begin{aligned}
D_0 &:= \{|, ||, \ldots, ||| \; ||| \; |||\} \\
D_1 &:= \{\cap, \cap\cap, \ldots, \cap\cap\cap \; \cap\cap\cap \; \cap\cap\cap\} \\
D_2 &:= \{\rho, \rho\rho, \ldots, \rho\rho\rho \; \rho\rho\rho \; \rho\rho\rho\} \\
D_3 &:= \{L, LL, \ldots, LLL \; LLL \; LLL\} \\
D_4 &:= \{F, FF, \ldots, FFF \; FFF \; FFF\} \\
D_5 &:= \{K, KK, \ldots, KKK \; KKK \; KKK\} \\
D_6 &:= \{G, GG, \ldots, GGG \; GGG \; GGG\}
\end{aligned}
$$

Dabei waren | (Strich), $\cap$ (Fessel), ρ (Strick),L (Lotusblume), F (stehender Finger), K (Kaulquappe) und G (Gott) spezielle Hieroglyphen.

3. Die Zahl $n = (1\,302\,120)_{10}$ wurde somit als

$$n = GKKKLL\rho\cap\cap$$

wiedergegeben.

3.3 Zahlen und Ziffernsystem bei den Griechen um 300 v. Chr.

Bezüglich der Wahl der Folge $b_k := 10$ liegt ein dekadisches System vor. Jedoch ähnlich wie in der Hieroglyphendarstellung verwendet man pro Zehnerpotenz unterschiedliche Ziffernmengen:

$$
\begin{aligned}
D_0 &:= \{\alpha, \beta, \gamma, \varepsilon, \digamma, \varsigma, \eta, \theta\} \\
D_1 &:= \{\iota, \kappa, \lambda, \mu, \nu, \xi, o, \pi, \qoppa\} \\
D_2 &:= \{\rho, \sigma, \tau, \upsilon, \varphi, \xi, \psi, \omega, \sampi\} \\
D_3 &:= \{'\alpha, '\beta, '\gamma, '\varepsilon, '\digamma, '\varsigma, '\eta, '\theta\} \\
D_4 &:= D_0 \\
D_5 &:= D_1 \\
D_6 &:= D_2 \\
D_7 &:= D_3
\end{aligned}
$$

$$D_8 := D_4$$
$$D_9 := D_5.$$

Die Symbole $\digamma$ (Digamma), ς (Koppa) und λ waren einem älteren Alphabet entnommen und sind drucktechnisch in diesem Artikel nur andeutungsweise wiedergegeben worden. Die Symbole $'\alpha, \ldots$ wurden eigentlich in der Form $,\alpha, \ldots$ geschrieben, sodaß im Rahmen dieses Vortrags der Strich hochgezogen wurde, um die Beistriche als Trennzeichen der listenmäßigen Beschreibung der Ziffernmengen benützen zu können. Beim folgenden Zahlenbeispiel soll jedoch die Originalkonvention benützt werden.

Die Zahl $n = (51\,406\,789)_{10}$ wurde etwa in der Form

$$n = M, \digamma\rho\mu \cdot, \varsigma\psi\pi\theta$$

geschrieben. Es war M ein Zeichen für Zehntausender (10,000 = 1 Myriade) und $\cdot$ ein Trennzeichen.

4 Die Grundrechnungsarten für natürliche Zahlen in B–adischen Systemen

In diesem Abschnitt werde lediglich mit natürlichen Zahlen gearbeitet. Die Implementation von Routinen für ganze Zahlen sollte nach den Diskussionen dieses Abschnittes verstehbar sein. Konversionsroutinen zwischen verschiedenen B-adischen Systemen werden etwa in [Kn81,GR87] beschrieben.

Es werde vom B–adischen System mit Basis B und Ziffernmenge $D := \{0, 1, \ldots, B-1\}$ ausgegangen. Es sei MAXINT die größte im Rechner darstellbare *Integerzahl*. (Bei 32-bit Rechnern ist dies $2^{32} - 1$).

Es wird im Folgenden vorausgesetzt, daß *Maschinenprozeduren*

$$\text{ADD}, \text{SUB}, \text{MUL}, \text{DIV}$$

zur Verfügung stehen, welche bei Eingabe von Elementen der Ziffernmenge deren Summe, Differenz, Produkt, Quotienten (ganzzahliger Teil) ausgeben. Genauere Forderungen sollen bei der nachstehenden Diskussion erörtert werden.

Wie sich dabei herausstellt, ergibt sich eine maschinenabhängige Begrenzung für B, welches meist als möglichst große Potenz von 2 gewählt wird. Diese Begrenzung wird bei der Diskussion der Grundrechnungsarten sichtbar.

Im vorliegenden Bericht sollen ganze Zahlen mit dem Typ WORD dargestellt werden (unsignierte Integer-Zahlen). Die Handhabung für den Datentyp SIGNED – INTEGER ist ähnlich.

Zunächst seien natürliche Zahlen

$$x := (x_m x_{m-1} \dots x_0)_B, y := (y_n y_{n-1} \dots y_0)_B$$

als *Ziffernfolgen* gegeben. (Wie gesagt, alle x_j, y_j sind dann vom Typ WORD).

Es mag vermerkt werden, daß in den meisten, auf LISP basierenden Computeralgebrasystemen solche Ziffernfolgen als *lineare Listen* gespeichert werden.

Ist $\oplus$ eine der vier Rechenoperationen $+, -, \times, \div$ so ist zunächst ein Algorithmus gefragt, welcher

$$x \oplus y := (c_p c_{p-1} \dots c_0)_B$$

berechnet, sofern das sinnvoll ist. Es folgt eine Diskussion für die einzelnen Grundrechnungsarten.

4.1 Addition

Als implementiert wird eine *Maschinenaddition* ADD angesehen.
INPUT : Zwei Maschinenzahlen $0 \leq x, y \leq B-1$.
OUTPUT: Ein Paar (z, C), wobei $0 \leq z \leq B-1$ und $C \in 0, 1$. Genauer:

$$(z, C) = \begin{cases} (x+y, 0) & \text{falls } x+y \leq B-1, \\ (x+y-B, 1) & \text{falls } x+y > B-1. \end{cases}$$

In etwas salopper Manier würde man schreiben:

$$ADD(x, y) = (z, C).$$

Die Tatsache, daß im Gegensatz zur händisch ausgeführten Addition zweier Zahlen das Ergebnis von ADD ein *Paar* von Zahlen ist, ist zwar gewöhnungsbedürftig, jedoch unumgänglich zur Darstellung des *Übertrags* beim Addieren (C steht hier für *Carry*, das englische Wort für Übertrag).

Die Papier- und Bleistiftmethode steht Pate bei der Formulierung des nachstehenden Additionsalgorithmus ADDEXT:
INPUT : Die Ziffernfolgen für x und y. Dabei geht man davon aus, daß $m, n \leq$ MAXINT gelten.
OUTPUT: Ein Paar (s, S). Falls $x + y > B^{\text{MAXINT}}$ so wird $(s, S) := (0, 1)$ ausgegeben. Andernfalls wird für s die Ziffernfolge für $x+y$ und $S := 0$ ausgegeben.

Anmerkung: Wo es bequem ist, wird $x_j = 0, y_j = 0$ für $j > m$, bzw. $j > n$ gesetzt.

Die einfachste Form des Algorithmus lautet:

$p := \max\{m, n\}$
$C_{-1} := 0$
for $j := 0$ to p do begin (j-loop)
(c_j, C_j) :=ADD(x_j, y_j)
(c_j, C_j) :=ADD(c_j, C_{j-1})
end (j-loop)
if $p =$ MAXINT and $C_p = 1$ then $(s, S) := (0, 1)$
else $(s, S) := ((c_p c_{p-1} \cdots c_0)_B, 0)$
ADDEXT:= (s, S)

Anmerkungen:

1. Die zweite Komponente S mag als *Signal, Flagge* gedeutet werden. Sie benötigt ein einziges Bit und dokumentiert das Fehlerverhalten von ADDEXT.

2. Es läßt sich zeigen, daß $B \leq$ MAXINT $+ 1$ gewählt werden kann.

4.2 Multiplikation

Als implementiert wird eine *Maschinenmultiplikation* MUL angesehen. Die übliche Multiplikation von WORDs führt zu OVERFLOW, weil die Multiplikation zweier etwa 32-Bit-Zahlen natürlich mehr als 32 Bit zur Darstellung verbrauchen kann. Das führt zu dem Problem, daß man $B < \sqrt{\text{MAXINT} + 1}$ wählen müßte. Um dies zu vermeiden, werde mittels der Maschinenroutine MUL eine Hilfsfunktion MUL0 geschrieben. Zunächst jedoch die Spezifikation von MUL:
INPUT : Zwei Maschinenzahlen $0 \leq x < B, 0 \leq y < B$.
OUTPUT: Ein Paar MUL$(x, y) := (a, b)$, wobei $0 \leq a < B$ und $b \in \{0, 1\}$. Genauer:

$$(a, b) = \begin{cases} (x * y, 0) & \text{falls } x * y < B, \\ (0, 1) & \text{falls } x * y \geq B. \end{cases}$$

Die Implementation solch einer Routine hängt von der jeweiligen Maschinensprache am jeweiligen Prozessor ab.

Unter der Annahme, daß $B+1$ durch 4 teilbar und $B \leq \text{MAXINT}+1$ gilt, werde nun mit Hilfe von MUL die Prozedur MUL0 definiert. Dazu gehen wir von der eindeutigen Darstellbarkeit der nichtnegativen Zahlen

$$x = x_0 + x_1\frac{B}{2}, \quad x_0, x_1 \in \{0,1\},$$
$$y = y_0 + y_1\frac{B}{2}, \quad y_0, y_1 \in \{0,1\},$$

und der Multiplikationsformel

$$x * y = x_0y_0 + (x_0y_1 + x_1y_0)\frac{B}{2} + x_1y_1\frac{B}{4}B$$

aus. Unter den gemachten Annahmen ist der Koeffizient von B ganzzahlig und der Koeffizient von $B/2$ kann nur die Werte 0, 1, oder 2 annehmen. Falls mindestens eine der Zahlen x_0, x_1, y_0, y_1 verschwindet, ist die Summe der ersten beiden Terme kleiner als $B-1$. Falls im Gegensatz dazu $x_0 = x_1 = y_0 = y_1 = 1$ gilt, so ist $x = y = 1 + B/2$ und $xy = 1 + (1 + B/4)B$. Unter den gemachten Annahmen ist $1 + B/4 < B$ eine ganze Zahl.

All diese Überlegungen einbeziehend, könnte die Routine MUL0 wie folgt definiert werden.

INPUT : $0 \leq x < B, 0 \leq y < B$.

OUTPUT: Ein Paar MUL0$(x,y) := (a,b)$ mit $0 \leq a < B$, $0 \leq b < B$, sodaß $x * y = a + b * B$.

Das Programm sieht etwa wie folgt aus:

if $(x_0 = x_1 = y_0 = y_1)$ then
$a := 1$
$(b,0)$:=ADD$(1, B/4)$
else
$(s1,0)$:=MUL(x_0, y_0)
$(s2,0)$:=MUL(x_0, y_1)
$(s3,0)$:=MUL(x_1, y_0)
$(s4,0)$:=ADD$(s2, s3)$
$(s5,0)$:=MUL$(s4, B/2)$
$(a,0)$:=ADD$(s1, s5)$
$(b1,0)$:=MUL(x_1, y_1)
$(b,0)$:=MUL$(b1, B/4)$
MUL0:= (a,b)

Als Vorstufe der Implementation der Multiplikation zweier nichtnegativer ganzer Zahlen (MULTEX) wird nun eine Routine MULEX definiert. Hier die Spezifikation:

INPUT : (a, x), wobei $0 \leq a < B$ und $x = (x_m x_{m-1} \cdots x_0)_B$ ist.

OUTPUT: Ein Paar MULEX$(a, x) := (p, S)$. Falls $a * x < B^{\text{MAXINT}}$ so wird $p := a * x$ und $S := 0$ ausgegeben, andernfalls $p := 0$ und $S := 1$.

Die Beschreibung von MULEX benützt die Formel

$$a * x = \sum_{j=0}^{m} (a x_j) B^j.$$

Da ax_j sich in der Form

$$ax_j = y_j + z_j B,$$

mit $0 \leq y_j < B, 0 \leq z_j < B$ und $(y_j, z_j) = \text{MUL0}(a, x_j)$ gilt, ergibt sich

$$ax_j = \sum_{j=0}^{m} y_j B^j + \sum_{j=1}^{m+1} z_{j-1} B^j.$$

Nun zur algorithmischen Beschreibung von MULEX.

for $j := 0$ to m do

(y_j, z_j) :=MUL0(a, x_j)

$y := (y_m y_{m-1} \cdots y_0)_B$

if $m + 1 >$ MAXINT and $z_m \neq 0$ then $(M, S) := (0, 1)$

else

$t_0 := 0$

If $m =$ MAXINT then $n := m - 1$ else $n := m$

for $j := 0$ to n do $t_{j+1} := z_j$

$t := (t_{n+1} t_n \cdots t_0)_B$

(M, S) :=ADDEXT(x, t)

MULEX:= (M, S)

Nun ist es klar, wie man einen Algorithmus MULTEXT zur Multiplikation zweier Zahlen x, y spezifiziert und beschreibt.

INPUT : $x := (x_m x_{m-1} \cdots x_0)$, $y := (y_m y_{m-1} \cdots y_0)$

OUTPUT: Ein Paar MULTEXT:= (z, S), wobei $(z, S) = (0, 1)$ angibt, daß OVERFLOW stattfindet. Im Falle, daß kein OVERFLOW stattfindet, wird MULTEXT:= $(x * y, 0)$ ausgegeben.

Die Implementation von MULTEX geht von der Formel

$$x * y = \sum_{j=0}^{m}(\sum_{l=0}^{n} x_j y_l B^l)B^n$$

aus und benützt neben der eben definierten Prozedur MULEX der Einfachheit halber eine Shiftroutine SHIFTRIGHT (Multiplikation mit B^k):

INPUT : $x := (x_m, x_{m-1} \cdots x_0)_B$ und $k \geq 0$.

OUTPUT:

$$\text{SHIFTRIGHT}(x,k) := \begin{cases} (0,1) & \text{falls } k+m > \text{MAXINT} \\ ((z_{k+m}z_{k+m-1}\cdots z_0)_B, 0) & \\ z_{k+j} = x_j, j = 0,\ldots,m & \\ z_l = 0, l := 0, k-1 & \text{sonst} \end{cases}$$

Die Implementation einer solchen Prozedur ist offensichtlich, sodaß nur die Implementation von MULTEXT angegegeben werde:

$S := 0,\ M := 0$
begin
for $j := 0$ to m do begin
    (M_j, S_j) :=MULEX(x_j, y)
    If $S_j = 1$ or $S = 1$ then $(M,S) := (0,1)$
    else begin
        (M_j, S_j) :=SHIFTRIGHT(M_j, j)
        If $S_j = 0$ then (M,S) :=ADDEXT(M, M_j)
MULTEXT:= (M,S)

4.3 Subtraktion nichtnegativer Zahlen mit nichtnegativem Ergebnis

Will man dem Prinzip, mit vorzeichenlosen ganzen Zahlen zu operieren, treu bleiben, so wird zunächst die Subtraktion nur gültig ausführbar wenn der Minuend nicht kleiner als der Subtrahend ist. Man benötigt Vergleichsroutinen ISGREATER, ISEQUAL für B-adische Zahlen. Kurz zur Spezifikation.

INPUT : x, y zwei B-adische Zahlen.

OUTPUT: ISGREATER:= TRUE, falls $x > y$. Andernfalls werde FALSE ausgegeben.

In ähnlicher Manier gebe es eine Funktion ISEQUAL welche TRUE ausgibt, genau dann, wenn $x = y$.

Danach wird eine Prozedur geschrieben, welche $x - y$ unter der Annahme $x \geq y$ berechnet. Für eine solche Prozedur muß kein OVERFLOW mitberücksichtigt werden. Der Einfachheit halber werde $+, -$ geschrieben, wo eigentlich die oben genannten Maschinenprozeduren ADD, SUB in geigneterweise benützt werden müßten.

$c_0 := 0$
for $j := 0$ to m do begin
$c_j := c_j + x_j - y_j$
If $c_j < 0$ then begin
$c_{j+1} := -1$
$c_j := B - c_j$
SUBEXT:= $(c_m c_{m-1} \cdots c_0)_B$

4.4 Division nicht negativer Zahlen

Ausgegangen werde von einer Maschinenprozedur DIVREM mit folgender Spezifikation:
INPUT : Ein Paar von Zahlen $0 \leq x < B, 0 \leq y < B$.
OUTPUT: Ein Paar DIVREM$(x,y) := (q,r)$ wobei $x = qy + r$ mit $0 \leq q$ und $0 \leq r \leq y - 1$.

Auch die Division kann unter Zuhilfenahme der Ideen, welche hinter der Papier- und Bleistiftmethode stehen, implementiert werden. Will man dies einigermaßen effizient gestalten, so entstehen eine Reihe von Fragen.

Zunächst soll eine Routine DIV0 mit folgender Spezifikation implementiert werden:
INPUT : Ein Zahlenpaar (x, d_0) mit $x := (x_1, x_0)_B$ und $0 < d_0 < B$.
OUTPUT: Eine Zahlenpaar DIV0$(x, d_0) := (q(x), r(x))$ mit $q(x) = (q_1(x), q_0(x))_B$ und $0 \leq r(x) < d_0$ mit

$$x = q(x)d_0 + r(x).$$

Die Implementation dieser Routine könnte etwa wie folgt aussehen:

$(q(x_1), r(x_1))$:=DIVREM(x_1, d_0)
$(q(x_0), r(x_0))$:=DIVREM(x_0, d_0)
$q(x) := q(x_1)B + q(x_0)$
$x := x - q(x)d_0$

$(q(B), r(B))$ sei durch $B = q(B)d_0 + r(B)$, $q(B) \geq 0, 0 \leq r(B) < d_0$ definiert
(* Wie könnte die Implementation dieses Schrittes aussehen ? *)
repeat
$q(x) := q(x) + x_1 q(B)$
$x := x - q(x)d_0$
until $x < B$
(y, r) :=DIVREM(x, d_0)
DIVREM0:= $(q(x) + y, r)$

Diese Routine gibt eine *Schätzung* für die erste(n) Stelle(n) des Quotienten in einer Division an.

Den Nachweis der Korrektheit erlaube ich mir, dem Leser zu überlassen.

Nun zur Prozedur DIVREMEXT.

INPUT : $x = (x_m x_{m-1} \cdots x_0)_B, y = (y_n y_{n-1} \cdots y_0)_B$
OUTPUT: DIVREMEXT$(x, y) := (q, r)$, wobei

$$x = yq + r, \quad 0 \leq r < y$$

gilt.

Eine Rohform des Algorithmus läßt sich sofort hinschreiben:

$X := x$, $q := 0$
repeat
if $X < y$ then $R := X$
else begin
(*) Bestimme $0 \leq Q < B$ mit $X - Qy \geq 0, X - (Q+1)y < 0$
$q := B * q + Q$
$X := X - Qy$
until $X < y$
DIVREMEXT:= (q, R)

Nur die mit (*) bezeichnete Zeile bedarf eines Hinweises zur Implementation. Man hat demnach zwei Zahlen x, y gegeben und möchte Q finden. Dabei kann man wie folgt vorgehen:

- Es werde in einem ersten Schritt x, y mit einer passenden Zahl k_0 (üblicherweise eine Potenz von 2) multipliziert, sodaß (nach eventuellem Bezeichungswechsel) $y_n \geq B/2$ gilt.

- Für das so modifizierte x, y ergibt sich mit Hilfe der Routine DIV0 aus $x_m B + x_{m-1}$ und y_n eine Schätzung $\hat{Q}$ von Q. Dabei wird grundsätzlich im Falle $\hat{Q} \geq B$ $\hat{Q} := B - 1$ gesetzt. Für dieses $\hat{Q}$ kann $\hat{Q} - 2 \leq Q \leq \hat{Q}$) gezeigt werden. Durch Multiplikation (MULTEXT) und Subtraktion (SUBEXT) können die drei Fälle entschieden werden, wobei, nun wieder das ursprüngliche x, y verwendend, $x - (\hat{Q} - 2)y, x - (\hat{Q} - 1)y, x - \hat{Q}$ auf erstmalig auftretendes nichtnegatives Vorzeichen untersucht werden.

- Dieses Q ist, mit der richtigen Potenz von B multipliziert, die erste *Stelle* des Quotienten.

Eine genaue Darstellung der Argumente findet man in [Kn81].

5 Gleitkommazahlenspeicherung bei 32-bit Prozessoren

Der Standard für die Implementation der Arithmetik von Gleitkommazahlen wurde 1985 im IEEE-754 festgelegt ([IEEE-754]). Rundungsmethoden und Fehlerverhalten sind weitgehend vorgegeben. Dem Standard entsprechende Multiprezisionsarithmetik, ausgehend von den Grundrechnungsarten bezüglich eines B–adischen Systems zu implementieren, stellt kein großes Problem dar. Um der Länge des Artikels willen sei lediglich die Darstellung von Gleitkommazahlen beschrieben. Für die allgemeine Theorie siehe etwa [SB92]. Die oben angeführten Prozeduren für die Grundrechungsarten können dabei verwendet werden, allerdings bedarf es sorgfältiger Untersuchungen über das Verhalten der Nachkommastellen. Vorallem die Addition von G. ist subtil.

5.1 Speicherung von Gleitkommazahlen

Zunächst stehen 32 Bit zur Verfügung. Zur Darstellung einer Gleitkommazahl x_M steht das Format

$$x_M := (-1)^s \times (b_0.b_1 a_2 \ldots b_{p-1}) \times 2^E$$

zur Verfügung, wobei

s = 0 oder 1.

b_i sind 0 oder 1

E ist eine Zahl zwischen 127 und - 126

p = 24, die Anzahl der *signifikanten* Stellen

Die 32 Bits werden nun wie folgt belegt:

1	:	Vorzeichen s
2 - 9	:	e:=E + 127 (verschobener Exponent)
10 - 32	:	Eine noch zu erklärende Binärzahl f

Neben den Gleitkommazahlen gibt es noch Größen, welche als NaN (not a number) bezeichnet werden. Sie werden als Signale bei Fehlern benützt. Die folgende Tabelle beschreibt die Zahlbereiche der Gleitkommazahlen:

NaN	:	falls e = 255, und f$\neq$ 0
$(-1)^s\infty$	:	falls e = 255, und f = 0
$(-1)^s 2^{e-127}(1.f)$	:	falls $0 < e < 255$
$(-1)^s 2^{-126}(0.f)$	:	falls e = 0 und f$\neq$ 0
$(-1)^s 0$	:	falls e = f = 0
$(-1)^s 2^{E-127}(1.f)$	:	falls $0 < e < 255$

Zum Datentyp DOUBLE siehe auch [Sch94].

6 Diverses zu den häufigsten Zahlsystemen

6.1 Binärsystem

6.1.1 Ägypten, ca 1200 v. Chr.

Die Multiplikationsmethode enthält in versteckter Form Binärdarstellungen. Die übliche Form der Erklärung für etwa das Multiplizieren von 8 mit 13 in heutigem Zifferngebrauch lautet:

1. Es sei 8 der Multiplikand und 13 der Multiplikator.
2. Den folgenden Anweisungen gemäß lege man eine zweispaltige Tabelle an.
3. In der ersten Zeile schreibe man die Zahlen 1 und den Multiplikanden (hier 8).

4. Nun schreibe man unter die beiden Zahlen jeweils das Doppelte, also in der zweiten Zeile die Zahlen 2 und 16.

5. Ist der k-te Schritt getan, so schreibe man im $(k+1)$-ten Schritt unter jede der beiden Zahlen der k-ten Zeile das jeweils Doppelte.

6. Abbruchbedingung: Wird in der ersten Spalte durch Verdoppeln eine Zahl gewonnen, welche größer als der Multiplikator (hier 13) ist, so breche man den Vorgang ab.

7. In der angegebenen Liste streiche man jene Zeilen an, in welchen sich durch Summieren von Zahlen der ersten Spalte (entsprechender Potenzen von 2) der Multiplikator (die Zahl 13) ergibt.

8. Man addiere die entsprechenden rechten Seiten.

Hier das Beispiel, in welchem moch eine Spalte mit Zeilennummern und eine mit Kommentaren hinzugefügt wurden.

n	Strich	2^{n-1}	$2^{n-1} \times 8$	Kommentar
1	/	1	8	Initialisierung
2		2	16	Verdoppeln von Zeile 1
3	/	4	32	Verdoppeln von Zeile 2
4	/	8	64	Verdoppeln von Zeile 3
5		16	—	Abbruch, weil $16 > 13$

Somit ergibt sich $8 \times 13 = 1 \times 8 + 4 \times 8 + 8 \times 8 =$ 4.Spalte/Zeile 1 + 4.Spalte/Zeile 3 + 4.Spalte/Zeile $4 = 8 + 32 + 64 = 102$, wie es sein soll.

Betrachtet man die Binärentwicklung $(13)_{10} = (1101)_2$, so sieht man natürlich sofort, daß man die Zeilen mit den Nummern 1,3 und 4 braucht (indem man nachsieht, an welchen Stellen von rechts aus gelesen eine '1' vorkommt).

6.1.2 Englische Hohlmaße im 13. Jhdt.

Es gab eine vollständige binäre Skala von Hohlmaßen.

1	gill
2	chopin
3	pint
4	quart
5	pottle
6	gallon
7	peck

8	demibushel
9	firkin
10	kilderkin
11	barrel
12	hogshead
13	pipe
14	tun

Es waren 2 gills = 1 chopin, 2 chopin = 1 pint, etc. Insbesondere ist 1 tun = 2^{13} gills = $(8,192)_{10}$ gills.

Es war damals auch üblich, die entsprechenden Mengen (etwa Bier) in einer Null-Eins-Schreibweise zu notieren. Tatsächlich gelangte man in dieser Form zur Binärentwicklung für die Gesamtmenge (etwa konsumierten Biers).

6.2 Das Oktalsystem

- 1717 bemühte sich Karl XII von Schweden, das Dezimalsystem durch das O. in Schweden zu ersetzen. (Auch Basis $B = 64$ war für ihn interessant).
- 1745 Vorschlag von Rev. Hugh Jones, das O. in Maryland zu benützen.
- 1845 J.D.Colonne schlägt in Frankreich das O. vor.
- Um 1930 war im Englischen noch der Begriff *octonal system* üblich.

6.3 Die polyadische Struktur des österreichischen Münz- und Banknotensystems

Im folgenden gelte eine Beschränkung auf Schillingbeträge. Dann gibt es offenbar Münzen bzw. Banknoten mit folgenden Werten (in ÖS):

$$\{1.-, 5.-, 10.-, 20.-, 100.-, 500.-, 1,000.-, 5,000.-\}$$

Fragen:

1. Wie kann man unter der Annahme, daß man hinreichend viele Münzen / Banknoten zur Verfügung hat, den Betrag von

   $$S8,783.-$$

   mit möglichst wenigen Einheiten darstellen?

2. Wenn man jeden Betrag bis S 9,999.– stets mit einem Minimum an Einheiten darstellen will, wieviele Münzen / Banknoten braucht man?

Natürlich ist es ziemlich klar, wie man zur Lösung kommt!

Lassen Sie mich Ihnen aufzeigen, daß man dabei intuitiv eine *polyadische Struktur* benützt. Hier die nötigen Definitionen.

1. Seien $b_0 := 5, b_1 := 2, b_2 := 2, b_3 := 2,5, b_4 := 2, b_5 := 5, b_6 := 2, b_7 := 5$. b_0 ergibt sich zu 5, weil eben $5 \times S1.- = S5.-$, $b_1 = 2$, weil $2 \times S5.- = S10.-$, etc.

   Auch wenn in der oben gebrachten Definition die b_j ganze Zahlen sind, wurde hier etwa $b_3 = 2,5$ gesetzt, weil eben $2,5 \times S20.- = S50.-$ ergibt. Das soll im weiteren nicht stören, weil die Produkte

$$\begin{aligned}
b_0 &= 5\\
b_0 \times b_1 &= 10\\
b_0 \times b_1 \times b_2 &= 20\\
b_0 \times b_1 \times b_2 \times b_3 &= 50\\
b_0 \times b_1 \times b_2 \times b_3 \times b_4 &= 100\\
b_0 \times b_1 \times b_2 \times b_3 \times b_4 \times b_5 &= 500\\
b_0 \times b_1 \times b_2 \times b_3 \times b_4 \times b_5 \times b_6 &= 1000\\
b_0 \times b_1 \times b_2 \times b_3 \times b_4 \times b_5 \times b_6 \times b_7 &= 5000
\end{aligned}$$

   genau die ganzzahligen Werte der Münzen / Banknoten sind.

2. Nun werden die Ziffernsysteme festgelegt. Der Übersicht halber sind in runder Klammer die entsprechenden Geldeinheiten beigefügt.

$$\begin{aligned}
D_0(1) &:= D_5(100) := D_7(1000) := \{0,1,2,3,4\}\\
D_1(5) &:= D_2(10) := D_4(50) := D_6(500) := \{0,1\}\\
D_3(20) &:= \{0,1,2\}\\
D_8(5000) &:= I\!N
\end{aligned}$$

   Daß $D_8(5000)$ grundsätzlich unendlich zu wählen ist, wird im Weiteren nicht stören.

3. Die Lösung der ersten Frage läuft darauf hinaus, die polyadische Entwicklung

$$8,783 = a_0 + \sum_{j:=0}^{7} a_{j+1} b_0 \times \cdots \times b_j$$

$$= a_0 + 5a_1 + 10a_2 + 20a_3$$
$$+50a_4 + 100a_5 + 500a_6 + 1000a_7 + 5000a_8$$

mit $a_j \in D_j(\cdots), j := 0,1,2,....8$ anzugeben.

Das Verfahren, die a_j zu bestimmen, ist klar: Man dividiert 8,783 mit Rest durch 5000, sodaß $a_8 = 1$ und Rest 3,783 verbleibt. Nun dividiert man diesen Rest, nämlich 3,783, durch 1000, um $a_7 = 3$ und Rest 783 zu bekommen, etc.

Man bekommt als Lösung, wie zu erwarten:

$$8,783 = 3+0\times5+1\times10+1\times20+1\times50+2\times100+1\times500+3\times1000+1\times5000.$$

4. Die Lösung der zweiten Frage kann direkt von den Anzahlen der Elemente in den Mengen D_j, $j = 0,\ldots,8$ abgelesen werden. Man findet sofort, daß man

Einheit/S :	1.–	5.–	10.–	20.–	100.–	500.–	1000.–	5000.–
Anzahl :	4	1	4	2	4	1	4	1

benötigt.

Literatur

[G77] H.H. Goldstine,
A History of Numerical Analysis,
Springer, 1977

[GR87] S.K.Grosser, H.Rupprecht,
BASIC Mathematikprogramme,
Hölder-Pichler-Tempsky Wien, 1987

[IEEE-754] An American National Standard,
IEEE Standard for Binary Floating-Point Arithmetic,
754-1985

[Kn81] D.E.Knuth,
The Art of Computer Programming, Volume 2: Seminumerical Algorithms,
Addison-Wesley, 1981

[KN84] H.Kaiser, W.Nöbauer,
Geschichte der Mathematik für den Schulunterricht,
Hölder-Pichler-Tempsky Wien, 1984
(Eine Neuauflage ist in Arbeit)

[SB92] J.Stoer, R.Bulirsch,
Introduction to Numerical Analysis,
Springer, 1992

[Tr94] J.Trabesinger,
Zifferndarstellungen bezüglich komplexer Basen,
Diplomarbeit, TU-Wien, 1994

[Sch94] G.Schmidt,
Die Gleitkommaarithmetik des Datentypes DOUBLE (64 Bit) nach IEEE 754-1985,
Manuskript Wien, 1994

[W89] H.Wußing,
Vorlesungen zur Geschichte der Mathematik,
VEB, 1989