RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

LEHRSTUHL INFORMATIK 11 SOFTWARE FÜR EINGEBETTETE SYSTEME

Professor Dr.-Ing. Stefan Kowalewski
Dipl.-Inform. Dominik Franke
Dipl.-Inform. Sebastian Biallas

Aachen, 18. Oktober 2010
SWS: V4/Ü2, ECTS: 7

Einführung in die Technische Informatik

WS 2010/2011

Blatt 1: Musterlösung

ACHTUNG: Die Musterlösung ist ein zusätzliches Serviceangebot. Sie erhebt weder Anspruch auf Vollständigkeit noch auf Korrektheit.

Aufgabe 1: ($\star$) Darstellungsbereich

Aufgabe 2: ($\star$) Interpretation von Bits

Aufgabe 3: ($\star$) Addition und Subtraktion

Aufgabe 4: Bit-Shifts

Unter einem Shift versteht man das Verschieben von Bits. Die übliche Notation ist hierbei $x << k$ für ein k-faches Verschieben nach links und $x >> k$ für ein k-faches Verschieben nach rechts. Es gibt verschiedene Formen mit welchem Wert *aufgefüllt* wird, hier nehmen wir an, dass es sich einfach um Nullen handelt.

Sei der Wert von x gegeben durch die Hexadezimalzahl $(CC)_{16}$. Berechnen Sie $x << 3$ auf einer 8-Bit Architektur, indem Sie zunächst x als Binärzahl darstellen und dann die Shift-Operation durchführen. Welchen ganzzahligen Wert stellt $x << 3$ dar?

Lösungsvorschlag

$0xCC$ entspricht der Binärkodierung 11001100. Damit lässt sich die Shift-Operation einfach berechnen:

$$11001100 << 3 = 01100000$$

Daraus folgt dann, dass $x << 3 = 64 + 32 = 96$.

Aufgabe 5: ($\star$) Zahlensysteme

a) Wandeln Sie folgende Zahlen in die gegebenen Zahlensysteme um:

- $(2012)_3 = ()_2$
- $(4412)_5 = ()_{10}$
- $(192)_{10} = ()_8$

- $(H36G)_{18} = ()_7$
- $(1001010011)_2 = ()_{10}$

b) Wandeln Sie folgende Zahlen in die gegebenen Zahlensysteme um. Nutzen Sie dabei die Beziehungen zwischen den Zahlensystemen:

- $(10110100110100100111101)_2 = ()_{16}$
- $(10011010110010101101 00)_2 = ()_8$
- $(LIMBO)_{25} = ()_5$
- $(A5F2)_{16} = ()_2$

c) Führen Sie folgende Rechenoperationen in den gegebenen Zahlensystemen durch und geben Sie das Ergebnis in dem vorgegebenen Zahlensystem an:

- $(713)_8 + (742)_8 = ()_{16}$
- $(10101101)_2 - (10100110)_2 = ()_{10}$
- $(201)_3 \cdot (22)_3 = ()_6$
- $(A52)_{14} \cdot (A0)_{14} = ()_{18}$

Anmerkung: Zahlensysteme mit einer Basis größer 10 enthalten alphanumerische Zeichen um alle Ziffern darstellen zu können. Beispielsweise enthält das Hexadezimalsystem (Basis 16) folgende Ziffern: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Beim 18er System kommen zu den genannten Ziffern noch die „Ziffern“ G und H hinzu, etc.

Lösungsvorschlag

a) $(2012)_3 = 2 \cdot 3^3 + 1 \cdot 3^1 + 2 \cdot 3^0 = (59)_{10} = (111011)_2$

$(4412)_5 = 4 \cdot 5^3 + 4 \cdot 5^2 + 1 \cdot 5^1 + 2 \cdot 5^0 = (607)_{10}$

$(192)_{10} = 3 \cdot 8^2 + 0 \cdot 8^1 + 0 \cdot 8^0 = (300)_8$

$(H36G)_{18} = H \cdot 18^3 + 3 \cdot 18^2 + 6 \cdot 18^1 + G \cdot 18^0 = 17 \cdot 18^3 + 3 \cdot 18^2 + 6 \cdot 18^1 + 16 \cdot 18^0 =$
$(100240)_{10} = 5 \cdot 7^5 + 6 \cdot 7^4 + 5 \cdot 7^3 + 1 \cdot 7^2 + 5 \cdot 7^1 + 0 \cdot 7^0 = (565150)_7$

$(1001010011)_2 = 1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0 = (595)_{10}$

b) Weil $2^4 = 16$ folgt:
$(10110100110100100111101)_2 = (10\ 1101\ 0011\ 0100\ 1001\ 1101)_2 = (2D349D)_{16}$

Weil $2^3 = 8$ folgt:
$(10011010110010101101 00)_2 = (1\ 001\ 101\ 011\ 001\ 010\ 110\ 100)_2 = (11531264)_8$

Weil $5^2 = 25$ folgt:

$(LIMBO)_{25} = (L\ I\ M\ B\ O)_{25} = (41\ 33\ 42\ 21\ 44)_5$

25er System	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G
5er System	00	01	02	03	04	10	11	12	13	14	20	21	22	23	24	30	31
25er System	H	I	J	K	L	M	N	O									
5er System	32	33	34	40	41	42	43	44									

$(A5F2)_{16} = (1010\ 0101\ 1111\ 0010)_2$

c) $(713)_8 + (742)_8 = (1655)_8 = 1 \cdot 8^3 + 6 \cdot 8^2 + 5 \cdot 8^1 + 5 \cdot 8^0 = (941)_{10} = (3AD)_{16}$

$(10101101)_2 - (10100110)_2 = (111)_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (7)_{10}$

$(201)_3 \cdot (22)_3 = (12122)_3 = 1 \cdot 3^4 + 2 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = (152)_{10} = (412)_6$

$(A52)_{14} \cdot (A0)_{14} = (75960)_{14} = (284480)_{10} = 2 \cdot 18^4 + C \cdot 18^3 + E \cdot 18^2 + 8 \cdot 18^0 = (2CE08)_{18}$

Aufgabe 6: Die 0 im 1er-Komplement

Der Wert einer Kette von Bits der Länge w im 1er-Komplement lässt sich über folgende Formel errechnen:

$$\mathsf{one}(x) \quad = \quad -x_{w-1} \cdot (2^{w-1} - 1) + \textstyle\sum_{i=0}^{w-2}(x_i \cdot 2^i)$$

Wie, und auf wie viele Arten und Weisen, können Sie die Zahl 0 im 1er-Komplement darstellen?

Lösungsvorschlag

Es gibt zwei Darstellungsmöglichkeiten für die 0, und zwar $0 \ldots 0$ und $1 \ldots 1$:

$$\begin{aligned}
\mathsf{one}(0 \ldots 0) &= 0 \cdot (2^{w-1} - 1) + \textstyle\sum_{i=0}^{w-2}(0 \cdot 2^i) \\
&= 0 \\
\mathsf{one}(1 \ldots 1) &= -2^{w-1} + 1 + \underbrace{\sum_{i=0}^{w-2} 2^i}_{2^{w-1}-1} \\
&= 0
\end{aligned}$$

Dabei wird $0 \ldots 0$ üblicherweise als $+0$ interpretiert und $1 \ldots 1$ als -0. Obwohl Rechner mit 1er-Komplement für ganzzahlige Arithmetik früher eingesetzt wurden, basieren alle aktuellen Systeme auf dem 2er-Komplement. Beachte allerdings, dass die IEEE-754 Floating-Point Kodierung ähnlich zum 1er-Komplement ist.

Beweis (der Einzigartigkeit dieser Lösungen):

Für das Argument benötigen wir die geometrische Reihe.

Angenommen $-x_{w-1} \cdot (2^{w-1} - 1) + \sum_{i=0}^{w-2}(x_i \cdot 2^i) = 0$, dann gilt

$$x_{w-1} \cdot (2^{w-1} - 1) = \sum_{i=0}^{w-2}(x_i \cdot 2^i)$$

Wegen $x_{w-1} \in \{0,1\}$ reicht es, zwei Fälle zu betrachten, nämlich $x_{w-1} = 0$ and $x_{w-1} = 1$.

1. Falls $x_{w-1} = 0$, so gilt auch $x_{w-1} \cdot (2^{w-1} - 1) = 0$. Da $x_i \geq 0$ für alle $0 \leq i \leq w-2$, folgt daraus $x_i = 0$ für alle $0 \leq i \leq w-2$.

2. Falls $x_{w-1} = 1$, so gilt $x_{w-1} \cdot (2^{w-1} - 1) = 2^{w-1} - 1 = \sum_{i=0}^{w-2}(x_i \cdot 2^i)$. Wegen $0 \leq x_i \leq 1$ und $2^i > 0$ für alle $0 \leq i \leq w-2$ ist $\sum_{i=0}^{w-2} 2^i$ obere Schranke. Mittels der geometrischen Reihe folgt:

$$2^{w-1} - 1 = \sum_{i=0}^{w-2}(x_i \cdot 2^i) \leq \sum_{i=0}^{w-2} 2^i = \frac{1 - 2^{w-1}}{1-2} = \frac{2^{w-1} - 1}{2-1} = 2^{w-1} - 1$$

Alternativer Lösungsvorschlag: Gesucht sind die Zahlen x, für die $\mathsf{one}(x) = 0$ ist:

$$-x_{w-1} \cdot (2^{w-1} - 1) + \sum_{i=0}^{w-2}(x_i \cdot 2^i) = 0$$

Aus dieser Darstellung folgt, dass eine Lösung $(x_{w-1}, \ldots, x_0) = (0, \ldots, 0)$ ist. Eine weitere Lösung für $x_{w-1} = 0$ gibt es nicht, da $x_i \in \{0,1\}$ für alle $i \in \{0, \ldots, w-1\}$ und $\sum_{i=0}^{w-2}(x_i \cdot 2^i) > 0$, falls mindestens ein $x_i > 0$ ist.

Daher betrachte nun den Fall $x_{w-1} = 1$. Dann gilt:

$$\begin{aligned} & -\left(2^{w-1} - 1\right) + \sum_{i=0}^{w-2}(x_i \cdot 2^i) = 0 \\ \Leftrightarrow \quad & 2^{w-1} - 1 = \sum_{i=0}^{w-2}(x_i \cdot 2^i) \end{aligned}$$

Für die geometrische Reihe $\sum_{i=0}^{n-1} q^i$ gilt für $q \neq 1$: $\sum_{i=0}^{n-1} q^i = \frac{1-q^n}{1-q}$. Damit ergibt sich:

$$\begin{aligned} \sum_{i=0}^{w-2}(x_i \cdot 2^i) & \overset{x_i \in \{0,1\}}{\leq} \sum_{i=0}^{w-2}(1 \cdot 2^i) \\ & \overset{\text{geo. Reihe}}{=} \frac{1 - 2^{w-1}}{1-2} \\ & = 2^{w-1} - 1 \end{aligned}$$

Weitere Lösungen gibt es nicht, da die Gleichheit $\sum_{i=0}^{w-2}(x_i \cdot 2^i) = \sum_{i=0}^{w-2}(1 \cdot 2^i)$ nur für $x = (1, \ldots, 1)$ gilt.
Also gibt es im 1er-Komplement genau zwei Darstellungsmöglichkeiten für die Zahl 0, nämlich $0 = (0, \ldots, 0) = (1, \ldots, 1)$.

Aufgabe 7: ($\star$) Gleitkommazahlen nach IEEE-754

Stellen Sie die Zahlen -10.625 und 0.3828125 im IEEE-754 Format dar.

Lösungsvorschlag

-10.625:
Die Zahl ist negativ, also gilt $sign = 1$
Vorkommastellen: $(10)_{10} = (8+2)_{10} = (1010)_2$
Nachkommastellen: $(0.625)_{10} = (\frac{1}{2} + \frac{1}{8})_{10} = (0.101000\ldots)_2$
Normalisierung: $(10.625)_{10} = (1010.101)_2 = (1.010101)_2 \cdot 2^3$
Exponent:$(3)_{Exponent} = (3+127)_{10} = (10000010)_2$
$-10.625 = 1\ 10000010\ 01010100000000000000000$

0.3828125:
positive Zahl, also sign = 0
Vorkommastelle: $(0)_{10} = (0)_2$
Nachkommastellen:
$0.3828125 \cdot 2 = 0.765625 \Rightarrow 0$
$0.765625 \cdot 2 = 1.53125 \Rightarrow 1$
$0.53125 \cdot 2 = 1.0625 \Rightarrow 1$
$0.0625 \cdot 2 = 0.125 \Rightarrow 0$
$0.125 \cdot 2 = 0.25 \Rightarrow 0$
$0.25 \cdot 2 = 0.5 \Rightarrow 0$
$0.5 \cdot 2 = 1 \Rightarrow 1$
Normalisierung: $(0 + 0.0110001)_2 = (1.10001)_2 \cdot 2^{-2}$
Exponent: $(-2+127)_{10} = (64+32+16+8+4+1)_{10} = (01111101)_2$
$0.3828125 = 0\ 01111101\ 10001000000000000000000$

Aufgabe 8: Fallstricke in der Programmierung

Mitunter verhalten sich Computerprogramme anders als erwartet, was mit Unachtsamkeiten bei der Zahleninterpretation zusammenhängen kann. Betrachten Sie z. B. folgendes Programm in C:

```
float sum(float a[], unsigned int length) {
    int i = 0;
    float result = 0.0f;
    for (i = 0; i <= length-1; i++) {
        result += a[i];
    }
    return result;
}
```

In diesem Programm werden die in einem Array `a` gespeicherten Werte aufsummiert. Die Summe wird dann zurückgeliefert. Was passiert, falls das Programm mit dem Wert 0 für den Parameter `length` aufgerufen wird? Wie kann man dieses Problem einfach beheben?

Lösungsvorschlag

Falls man die Funktion mit dem Wert 0 für `length` aufruft, so stürzt das Programm ab. Das liegt daran, dass `length` eine vorzeichenlose Zahl ist (`unsigned int`), und C die

Vergleichsoperation von einem `signed int` und einem `unsigned int` als `unsigned` ausführt. Demnach ergibt die Operation `length-1` nicht -1 wie erwartet, sondern `MAXINT` (die grösste darstellbare Zahl).

Das bedeutet, dass die Schleife `MAXINT`-fach durchlaufen wird und demnach auf das Array `a` ausserhalb des gültigen Bereichs zugegriffen wird. Der Bug wird behoben, indem man:

- das `unsigned` im Typen streicht, oder
- `i<=length-1` durch `i<length` ersetzt.

Zusätzliche Informationen: Ein einfacher Weg solche Fehler zu vermeiden, ist einfach keine vorzeichenlosen Zahlentypen zu verwenden. C ist überhaupt eine der wenigen Sprachen, welche vorzeichenlose Ganzzahlen unterstützen. Java, z.B., unterstützt nur vorzeichenbehaftete Typen. Vorzeichenlose Darstellungen sind vor allem dann sinnvoll, wenn man die Variablen als Bit-Ketten betrachtet ohne eine numerische Interpretation.

Aufgabe 9: Assoziativität bei IEEE-754

Betrachten Sie folgendes Programm, unter der Annahme dass der `float` Datentyp Fließkommazahlen nach dem IEEE-754 Standard bereitstellt:

```
float a, b, c, d;
float x, y;
...
x = a + b + c;
y = b + c + d;
```

Offensichtlich enthält dieses Programm Additionen, die mehrfach auf den gleichen Werten ausgeführt werden, nämlich die Summenbildung von `b` und `c` in beiden Zuweisungen. Um Rechenzeit für dieses Programm zu optimieren, könnte man einfach eine zusätzliche Variable `t` einführen, um den Wert der Addition zwischenzuspeichern.

```
t = b + c;
x = a + t;
y = t + d;
```

Verhält sich dieses optimierte Programme genauso wie das ursprüngliche Programm? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Nein, das Programm verhält sich nicht gleich, was mit den Rundungsfehlern bei den Operationen zusammenhängt. Die Addition über den reellen Zahlen ist assoziativ, d. h. für $a, b, c \in \mathbb{N}$ gilt $(a + b) + c = a + (b + c)$.

Als Beispiel betrachte man $(3.14 + 1e10) - 1e10$. Das Resultat ist 0.0, da der Wert 3.14 aufgrund des internen Rundungsfehlers verloren geht. Andererseits ist das Resultat von $3.14 + (1e10 - 1e10)$ gleich 3.14, da die Subtraktion $1e10 - 1e10$ exakt ist.

Weitere Infos: Bei Maschinenzahlen geht die Assoziativität verloren. Berechnet man z. B. mit dem Taschenrechner mit einem 10-stelligen Display $3.14 + 1e10 - 1e10$, so erhält man:

$$\begin{aligned} & (3.14 + 1e10) - 1e10 = 1e10 - 1e10 = 0 \\ \text{aber:} \quad & 3.14 + (1e10 - 1e10) = 3.14 + 0 = 3.14 \end{aligned}$$

Die Menge der Maschinenzahlen ist endlich und deshalb sind die elementaren Rechenoperationen $+, -, *, /$ im Gegensatz zu $\mathbb{R}$ nicht abgeschlossen. Man spricht auch von einer Pseudoarithmetik. Bei der ersten Rechnung geht der Wert 3.14 durch einen Rundungsfehler verloren. Das Kommutativgesetzt bezüglich Addition und Multiplikation gilt bei Maschinenzahlen, das Distributivgesetz hingegen genauso wie das Assoziativgesetz nicht.