5. Übungsblatt

Abgabe bis 17.01.2010 (08:00 Uhr) über GOYA

Aufgabe 1 (5 Punkte)

Gegeben sei ein beliebige Zahl $(z)_b = z_n z_{n-1} \dots z_1 z_0$ zur Basis b mit den Ziffern z_n, z_{n-1}, $\dots$, z_1 und z_0. Dann wissen wir bereits, dass wir den Wert dieser Zahl im Dezimalsystem wie folgt berechnen können:

$$(z)_{10} = z_n \times b^n + z_{n-1} \times b^{n-1} + \dots + z_1 \times b + z_0$$

Das Horner-Schema[1] besagt nun, dass wir durch kleveres Ausklammern von b den Wert auch wie folgt berechnen können:

$$(z)_{10} = ((\dots((((z_n) \times b + z_{n-1}) \times b + z_{n-2}) \times b + \dots) \times b + z_1) \times b + z_0$$

Leiten Sie aus dieser Darstellung eine rekursive Formel mit geeigneter Abbruchbedingung zum Berechnen des Dezimalwertes einer beliebigen Zahl z zur Basis b ab. Erläutern Sie kurz die gewählte Rekursion als Java-Kommentar im Quelltext. Implementieren Sie in der Datei **`Horner.java`** eine Methode `hornerRek()`, die als Argumente eine Zeichenkette, welche die Zahl z beschreibt, und einen Zahl b, welche die Zahlenbasis angibt, erhält. Der Rückgabewert der Methode soll dem Dezimalwert von $(z)_b$ entsprechen und nach Ihrer rekursiven Definition berechnet werden.
In der `main`-Methode sollen Sie eine Zahl und deren Basis von der *Standardeingabe* lesen und das Ergebnis Ihrer Methode auf der *Standardausgabe* ausgeben. Dies soll solange geschehen, bis die Eingabe entsprechend abgeschlossen wird.
Beispiel:

```
# java Horner.java
ABCD
16
43981
```

[1] `https://secure.wikimedia.org/wikipedia/de/wiki/Horner_Schema`

```
3456
9
2562
```

(Hinweis: Sie können davon ausgehen, dass „Ziffern“ mit Werten größer 9 immer als Lateinische Großbuchstaben entsprechend der Vorlesung angegeben sind.)

Toy Machine

In der Vorlesung haben Sie die Toy-Maschine kennen gelernt. Um mit dieser Programme ausführen zu können, müssen Sie diese bei sich einrichten. Dazu gibt es folgende Möglichkeiten:

1. Sie importieren die *graphische* Variante XToy in Ihre Eclipse-Umgebung. Dazu wählen Sie im Menü „File“ die Option „Import...“. Bei der Art des Projektes wählen Sie nun „SVN“ bzw. „Project from SVN“. In den anschließenden Dialogen wählen sie als Repository folgende URL: `https://svn.informatik.hu-berlin.de/svn/gdp-10/XToy/`. Als Login nutzen Sie bitte Ihren Informatik-Account.
   Um das Projekt auszuführen, klicken Sie im „Package Explorer“ auf den Projektnamen („XToy“) und führen das Projekt einfach aus („Run“). Sollten Sie gefragt werden, welches die Hauptklasse ist, so wählen Sie bitte „XToy“.
   Die graphische Variante bietet den Vorteil, dass Sie die Programme hier direkt schreiben, testen und debuggen können (Schritt-für-Schritt-Ausführung, Übersicht über Register und Speicher).

2. Sie können sich alternativ die Datei `http://sar.informatik.hu-berlin.de/teaching/2010-w/2010-w%20GdP/etc/toy/toy_1.4.zip` herunterladen, und anschließend in dem Verzeichnis, in dem die Datei bei Ihnen liegt, folgenden Befehl ausführen: `java -jar toy_1.4.zip`
   Daraufhin startet die gleiche graphische Oberfläche wie unter Punkt 1.

3. In den Vorlesungsbeispielen befindet sich ein Interpreter für Toy-Programme im Paket `gdp.c1_01.Toy`. Das eigentliche Toy-Programm muss als Argument in den Ausführungseigenschaften der `Toy.java` angegeben werden. Danach kann das Konsolenfenster in Eclipse wie gewohnt als Standardeingabe für die Interaktion genutzt werden.

4. Sie können die Datei `Toy.java` auch direkt auf der Kommandozeile ausführen (`java Toy programm.toy`). Dies empfehlen wir Ihnen aber nur, wenn Sie wissen, wie Sie die benötigten Klassen wie in `gdp.stdlib` auf der Kommandozeile bereit stellen können.

Aufgabe 2 (12 Punkte)

Die Toy Machine bietet wie viele frühere Rechner nur die Möglichkeit, Zahlen im Hexadezimalsystem ein- bzw. auszugeben. Für uns Menschen erschwert dies die Eingabe,

da wir Zahlen im Dezimalsystem gewohnt sind. Wollen wir also die Zahl $(1234)_{10}$ eingeben, müssen wir diese erst ins Hexadezimalsystem umrechnen und stattdessen $(04D2)_{16}$ eingeben.
Als Lösung für dieses Problem wurden die *Binary coded decimals* (BCD) eingeführt, bei denen jede Ziffer einer Dezimalzahl mit vier Bits kodiert wird. Die Zahl $(1234)_{10}$ wird somit einfach als $(1234)_{16}$ (bzw. $(0001\,0010\,0011\,0100)_2$) geschrieben. Damit solch eine Eingabe in Operationen genutzt werden kann, muss die BCD-Zahl in eine normale Zahl umgewandelt werden.
Schreiben Sie also ein Toy-Programm `BCD.toy`, dass eine BCD-Zahl von der Standardeingabe liest, und den tatsächlichen Wert der BCD-Zahl auf der Standardausgabe ausgibt.

a) Für das Ausrechnen werden Sie mehrfach die Multiplikation zweier Zahlen benötigen. Übernehmen Sie dazu aus der Vorlesung die *Funktion* zum Multiplizieren (`function.toy`). Überlegen Sie sich beim Bearbeiten der nächsten Teilaufgabe eine sinnvolle Wahl der Zeilen, an der Sie Ihre Funktion im Quelltext unterbringen.
   Als Konvention sollten wie in der Vorlesung die beiden Zahlen a und b in $R[A]$ und $R[B]$ liegen und das Ergebnis in $R[C]$ landen. Die Rücksprungadresse soll sich im Register F befinden. Achten Sie darauf, welche Register durch Aufruf der Funktion möglicher Weise verändert werden!

b) Ihr Programm soll nun eine BCD-kodierte Zahl von der Standardeingabe lesen und daraus den tatsächlichen Wert berechnen. Die Eingabe ist ein Wort, also zwei Bytes und somit vier Ziffern für eine BCD-Zahl. Sie müssen jeweils die vier Bits, die eine der vier Ziffern kodieren, entsprechend ihrer Stelle mit einer Zehnerpotenz multiplizieren. Überlegen Sie, wie Sie auf die vier Bits, die einer Ziffer entsprechen, zugreifen können und welchem Wert diese Bits entsprechen. Das Ergebnis soll auf der Standardausgabe ausgegeben werden.

Wenn Sie also als Eingabe an Ihr Programm `1234` wählen, soll die Ausgabe `04D2` sein:

```
# java Toy BCD.toy
1234
04D2

# java Toy BCD.toy
0064
0040
```

Hinweis: Bitte kommentieren Sie Ihren Toy-Quelltext ausführlich, also die einzelnen Gedankenschritte, die Sie in den nächsten Zeilen Quelltext umsetzen wollen. Bei Operationen geben Sie an, wo die Operanden herkommen, und wo das Ergebnis zu finden sein soll (z.B. „`// an Speicherstelle 01/in Register A/... soll das Zwischenergebnis nach jedem Schleifendurchlauf stehen.`“)