Universität Karlsruhe (TH)

Institut für Innovatives Rechnen und Programmstrukturen (IPD)

Übersetzerbau WS 2003/04 — http://www.info.uni-karlsruhe.de/

Dozent: Prof. Dr.rer.nat. G. Goos — goos@ipd.info.uni-karlsruhe.de

Übungsleiter: Rubino Geiß — rubino@ipd.info.uni-karlsruhe.de

Übungsblatt 6 | Ausgabe: 08.01.2004 | Besprechung: 20.01.2004

Aufgabe 1: *Prozeduren mit variabler Parameterzahl*

Wie kann man die Übergabe von Parameter an Prozeduren organisieren, wenn die Anzahl der Parameter nicht fest ist? Welche Voraussetzungen kann man annehmen, unter denen Prozeduren mit variabler Parameterzahl überhaupt sinnvoll sind?

Aufgabe 2: *Auswertung von Ausdrücken / lokale Registerzuteilung*

Gegeben seien die folgenden Maschinenbefehle:

Befehl	Bedeutung
`ld` *offset*, r_i	r_i := Memory [*offset*]
`st` *offset*, r_i	Memory [*offset*] := r_i
`add` r_i, r_j	r_i := r_i + r_j
r_i seien Register	

2.1 *Codeerzeugung*

Erzeugen Sie für folgende Ausdrücke Code:

- $(a1 + (a2 + (a3 + (a4 + a5))))$ und
- $((((a1 + a2) + a3) + a4) + a5)$ sowie
- $((((a1 + a2) + a3) + a4) + (a5 + (a6 + (a7 + a8))))$.

Verwenden Sie zuerst das Verfahren: *zuerst linken, dann rechten Ausdruck auswerten*, dann das Verfahren *zuerst rechten, dann linken Ausdruck auswerten*.

Wieviele Register benötigen Sie?

2.2 *Optimale Registerzuteilung*

Gibt es ein besseres Verfahren? (Hinweis: Die Auswertungsreihenfolge wird abhängig vom Ausdruck bestimmt.) Wann kann dieses Verfahren angewandt werden, welche Probleme können entstehen?

(Siehe [Waite/Goos, 10.2.1])

2.3 *Lösung als AG*

Geben Sie für Ihr Verfahren eine Attributgrammatik an, die die Anzahl der benötigten Register und die Auswertungsreihenfolge der Operanden bestimmt. Betrachten Sie nullstellige, unäre und binäre Operatoren:

```
Expr ::= id | const.
Expr ::= op Expr.
Expr ::= Expr op Expr.
```

2.4 *INMOS*

Der *Transputer* [INMOS, *The transputer instruction set - a compiler writer's guide*, Prentice Hall 1988] hat anstatt eines Registersatzes, bei dem auf jedes Register direkt zugegriffen werden kann, einen *Auswertungskeller* der Tiefe drei. Jedes Kellerelement hat einen Namen:

Name	Position im Keller
`A`	top
`B`	
`C`	bottom

Jeder Prozedur ist eine Prozedurschachtel zugeordnet, in der Variablen gespeichert werden. Diese werden relativ zum `WSP` (Workspace Pointer) adressiert. Der Transputer kennt u.a. die folgenden Befehle:

Befehl	Bedeutung
`LDL` *offset*	`C := B; B := A; A := Memory [WSP +` *offset*`];`
`STL` *offset*	`Memory [WSP +` *offset*`] := A; A := B; B := C; C :=` *undefiniert*`;`
`LDC` *int_const*	`C := B; B := A; A :=` *int_const*`;`
`ADD`	`A := B + A; B := C; C :=` *undefiniert*`;`
`SUB`	`A := B - A; B := C; C :=` *undefiniert*`;`
`REV`	`A,B := B,A; C := C;` *(vertauscht die Werte von* `A` *und* `B`*)*

Entwerfen Sie ein Schema zur Codeerzeugung, das *Spillcode* einfügt. Spillcode sind Instruktionen, die eingefügt werden, um Ausdrücke, für deren Berechnung eigentlich mehr als die verfügbaren Register benötigt werden, dennoch auswerten zu können. Zum Beispiel: `STL/LDL` um Zwischenergebnisse abzuspeichern und wieder in Register zu laden. Die Kostenfunktion, die der Anzahl der Instruktionen entspricht, gilt es zu minimieren.

Betrachten Sie die Grammatik aus Aufgabe *(b)*. Die Konvention sei, daß das Ergebnis (d.h. der Wert von $\texttt{Expr}_0$) in `A` erwartet wird. Der Operator `op` erwartet seine Operanden in den Registern wie folgt:

Regel	Konvention
$\texttt{Expr}_0$ `::= op` $\texttt{Expr}_1$`.`	Der Wert von $\texttt{Expr}_1$ in `A`.
$\texttt{Expr}_0$ `::=` $\texttt{Expr}_1$ `Op` $\texttt{Expr}_2$`.`	Der Wert von $\texttt{Expr}_1$ in `B`;$\texttt{Expr}_2$ in `A`.