P. Thiemann, M. Neubauer Wintersemester 2004/2005

Compilerbau

http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2004/

Übungsblatt 13

Abgabe: 23.02.2005

Aufgabe 1 (Registerallokation):

Das folgende Programm wurde für eine Drei-Registermaschine mit Registern r_1, r_2 und r_3 übersetzt, wobei r_1 und r_2 (*caller-save*) Argumentenregister und r_3 ein *callee-save* Register ist.

Konstruiere den Register-Interferenzgraphen des Programms und führe daran den Registerallokationsalgorithmus aus der Vorlesung inklusive Verschmelzung Schritt für Schritt durch.

f:	$c \leftarrow r_3$	
	$p \leftarrow r_1$	
	`if` $p = 0$ `goto` L_1	
	$r_1 \leftarrow$ `Mem`$[p]$	
	`call` f	(uses r_1, defines r_1, r_2)
	$s \leftarrow r_1$	
	$r_1 \leftarrow$ `Mem`$[p + 4]$	
	`call` f	(uses r_1, defines r_1, r_2)
	$t \leftarrow r_1$	
	$u \leftarrow s + t$	
	`goto` L_2	
L_1:	$u \leftarrow 1$	
L_2:	$r_1 \leftarrow u$	
	$r_3 \leftarrow c$	
	`return`	(uses r_1, defines r_1, r_2)

Falls mehrere Kandidaten für einen *Spill* existieren, so wähle den mit geringster Priorität, wobei diese definiert ist durch

(#uses+defs-outside-loop + 10 × #uses+defs-within-loop)/outdegree.

Aufgabe 2 (*Reaching Definitions*):
Berechne für das folgende Program für jede Anweisung die jeweils *Reaching Definitions*:

```
x := 1
y := 1
if z <> 0
  then x := 2
  else y := 2
w := x + y
```

Gehe dazu wie folgt vor:

(i) Zeichne den Kontrollflussgraphen (CFG) des Programs.

(ii) Zur Berechnung der Lösung von Datenflussgleichungen ist es von Vorteil, die Knoten des Graphen in einer quasi-topologisch sortierten Reihenfolge zu betrachten, so dass die meisten Knoten vor ihren Nachfolgeknoten betrachtet werden.

Berechne eine quasi-topologische Sortierung `sorted` für den CFG mit Hilfe des folgenden Tiefensuchealgorithmus, der für azyklische Graphen eine topologische Sortierung und für zyklische Graphen eine quasi-topologische Sortierung berechnet:

```
Topological-Sort:
  N ← number of nodes
  for all nodes i
    mark[i] ← false
  DFS(start-node)

function DFS(i)
  if mark[i] = false
     mark[i] = true
     for each successor s of node i
        DFS(s)
     sorted[N] ← i
     N ← N - 1
```

(iii) Berechne die `gen`- und `kill`-Mengen der einzelnen Anweisungen.

(iv) Berechne die *Reaching Definitions* für die einzelnen Anweisungen des Programs, indem Du die zugehörigen Datenflussgleichungen mit Hilfe des folgenden iterativen Verfahrens löst:

```
repeat
  for i ← 1 to N
    n ← sorted[i]
    in ← ⋃_{p∈pred[n]} out[p]
    out ← gen[n] ∪ (in - kill[n])
until no out set changed in this iteration
```

Gib die Zwischenergebnisse der einzelnen Iterationsschritte separat mit an!