

Programming Exercises Compiler Construction

Code Generation

Auf der nachfolgenden Seite finden Sie den 3-Adress-Code einer Funktion f , die folgende Formel berechnet:

$$s = \sum_{i=0}^n |a[i] - b[i]|$$

Dabei sind die Parameter a , b und n bereits in Registern enthalten, und das Ergebnis s soll auch in einem Register zurückgegeben werden.

Aufgaben

- Umrahmen Sie die Basisblöcke, aus denen der Code besteht, und ergänzen die Kanten des Kontrollfluß-Graphen durch Pfeile.
- Zeichnen Sie für den größten Basisblock (der nach dem Befehl "if $i > n$ goto done" beginnt) die DAG-Darstellung unter Berücksichtigung gemeinsamer Teilausdrücke.
- Eliminieren Sie gemeinsame Teilausdrücke (falls vorhanden) im 3-Address-Code des Basisblocks, indem Sie eliminierte Instruktionen streichen und die TEMP's entsprechend anpassen.
- Tragen Sie für alle (verbliebenen) Instruktionen der Funktion f die def- und use-Mengen in die Tabelle neben dem Code ein. Bestimmen Sie dann die live-in-Mengen aller Instruktionen durch Backward-Propagation von der return-Instruktion und tragen sie ebenfalls in die Tabelle ein.
- Bestimmen Sie aus den live-in-Mengen den Register-Konflikt-Graphen der Funktion f . Berücksichtigen Sie zur Vereinfachung dabei nur die TEMP's $t1 - t7$. (Gehen Sie davon aus, dass für die Variablen a , b , n , i und s bereits feste Register gleichen Namens vorgesehen sind.)

Wieviele zusätzliche Register sind für die TEMP's mindestens nötig, damit die Funktion ohne Register-Spills implementiert werden kann? "Färben" Sie den Graphen, indem Sie jedem TEMP ein Register zuordnen.

- Schreiben Sie mit der gefundenen Registerallokation einfachen Assemblercode für die Funktion.
 - Verwenden Sie a , b , n , i und s direkt als Register.
 - Verwenden Sie folgende 3-Address-Assembler-Befehle: MOV, CMP (compare), BRGT (branch if greater than), BRGE (branch if greater or equal), MUL, ADD, SUB, LD (load indirect), JUMP und RET (return).

Beispiel: Der erste Befehl von f lautet: MOV i , $\$0$

3-Adress-Code der Funktion f:

	def	use	live-in
f: i := 0	i		
s := 0	s		
loop: if i>n goto done		i, n	
t1 := i * 4	t1	i	
t2 := a + t1	t2		
t3 := [t2]			
t4 := i * 4			
t5 := b + t4			
t6 := [t5]			
t7 := t3 - t6			
if t7>=0 goto else			
t7 := - t7			
else: s := s + t7			
i := i + 1			
goto loop			
done: return s			

b)

