Übung zur Vorlesung Compiler 1: Grundlagen

Prof. Dr. Andreas Koch
Jens Huthmann, Florian Stock

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 11/12
Aufgabenblatt 4

Abgabemodalitäten

Die Aufgabe sind in 2er Gruppen zu bearbeiten. Ihre Lösungen reichen Sie als PDF-Datei per E-Mail bis zum 16.02.2012 um 23:59 MEZ an der Adresse oc@esa.informatik.tu-darmstadt.de ein. Die E-Mail muss als Betreff "Compiler 1 Aufgabe 4" haben. Geben Sie in Ihrem Lösungsdokument den Namen und die Matrikelnummer aller Gruppenmitglieder an.

Aufgabe 4.1 Code-Schablonen — 40 Punkte

Erzeugen Sie mit Hilfe der Code-Schablonen aus den Folien von Block 05, Seite 8 ff. die TAM-Befehle für das gegebene Programm. Gehen Sie dabei schrittweise, vergleichbar eines Visitors, vor. Jeder Aufruf einer Code-Schablone sollte ein Schritt Ihrer detaillierten Lösung darstellen. Markieren Sie hierbei, wann eine Sprungadresse nicht direkt eingetragen werden kann. In einem solchen Fall markieren Sie auch den Schritt, in dem die Adresse durch Backpatching eintragen wird. Die Adressen der Variablen sind nicht notwendig, es reicht das Sie die Variablennamen angeben.

```
let
   const MAX ~ 10;
   var n: Integer
in begin
   n := 7;
   if (n>0) /\ (n<=MAX) then
      while n > 0 do begin
         if (n = 7) then
            put('!')
         else
            putint(n);
        n := n - 1;
       end
   else
end
```

Listing 1: Programm zur Übersetzung mittels Code-Schablonen

Aufgabe 4.2 Adressen von Konstanten und Variablen — 20 Punkte

Erweitern Sie den gegebenen TAM-Code um Befehle für die Speicherverwaltung und ersetzen Sie die Variablen-, Prozedur- und Funktionsnamen durch ihre konkreten Adressen gemäß den Beispiel in Abbildung 1. Die Adressen der primitiven Routinen können sie der folgenden Tabelle entnehmen.

Adresse	Routine
2	not
8	add
9	sub
10	mult
16	gt
24	puteol
26	putint

```
let
  var x : Integer;
  const n ~ 0-1
in
  x := n + 1;
```

(a) Triangle-Code

```
0  ...
1  LOADL      0
2  LOADL      1
3  CALL       sub
4  LOAD   (1) n
5  LOADL      1
6  CALL       add
7  STORE  (1) x
8  ...
9  HALT
```

(b) TAM-Code

```
0  PUSH       1
1  LOADL      0
2  LOADL      1
3  CALL   (SB) 9[PB]
4  LOAD   (1) 1[SB]
5  LOADL      1
6  CALL   (SB) 8[PB]
7  STORE  (1) 0[SB]
8  POP    (0) 2
9  HALT
```

(c) Lösung TAM-Code

Abbildung 1: Beispiel

```
let
   const MAX ~ 10;

   proc test(a : Integer) ~
   let
      var b : Integer
   in
      begin
         b := 2 * a;
         putint(b)
      end
in
   test(MAX)
```

(a) Triangle-Code

```
0  JUMP       10[CB]
1  ...
2  LOADL      2
3  LOAD   (1) a
4  CALL       mult
5  STORE  (1) b
6  LOAD   (1) b
7  CALL       putint
8  ...
9  RETURN(0)  1
10 LOADL      10
11 CALL   (SB) test
12 HALT
```

(b) TAM-Code

Abbildung 2: Aufgabe a

```
let
   var a : Integer;

   proc test(var x : Integer) ~
   let
      var b : Integer;
      const A ~ 10
   in
      begin
         b := 10;
         x := x + A + b;
         putint(A + x)
      end
in
   begin
      a := 4;
      test(var a)
   end
```

(a) Triangle-Code

```
 0  ...
 1  JUMP        20[CB]
 2  ...
 3  LOADL       10
 4  STORE (1)   b
 5  LOAD  (1)   x
 6  LOADI (1)
 7  LOADL       10
 8  CALL        add
 9  LOAD  (1)   b
10  CALL        add
11  LOAD  (1)   x
12  STOREI(1)
13  LOADL       10
14  LOAD  (1)   x
15  LOADI (1)
16  CALL        add
17  CALL        putint
18  ...
19  RETURN(0)   1
20  LOADL       4
21  STORE (1)   a
22  LOADA       a
23  CALL  (SB)  test
24  ...
25  HALT
```

(b) TAM-Code

Abbildung 3: Aufgabe b

```
let
   var a : Integer;
   var b : Integer;

   func f(i : Integer) : Integer ~
      i * i;

   proc p(l : Integer, var u : Integer) ~
   begin
      while u > l do begin
         putint(f(u));
         puteol();
         u := u - 1;
      end
   end
in
   begin
      a := 10;
      b := 0;
      p(b, var a);
   end
```

(a) Triangle-Code

```
 0  ...
 1  ...
 2  JUMP        7[CB]
 3  LOAD  (1)   i
 4  LOAD  (1)   i
 5  CALL        mult
 6  RETURN(1)   1
 7  JUMP        26[CB]
 8  JUMP        20[CB]
 9  LOAD  (1)   u
10  LOADI (1)
11  CALL  (SB)  f
12  CALL        putint
13  CALL        puteol
14  LOAD  (1)   u
15  LOADI (1)
16  LOADL       1
17  CALL        sub
18  LOAD  (1)   u
19  STOREI(1)
20  LOAD  (1)   u
21  LOADI (1)
22  LOAD  (1)   l
23  CALL        gt
24  JUMPIF(1)   9[CB]
25  RETURN(0)   2
26  LOADL       10
27  STORE (1)   a
28  LOADL       0
29  STORE (1)   b
30  LOAD  (1)   b
31  LOADA       a
32  CALL  (SB)  p
33  ...
34  HALT
```

(b) TAM-Code

Abbildung 4: Aufgabe c

Aufgabe 4.3 Wissensfragen — 15 Punkte

a) Benennen Sie das Eingabeformat und die Ausgabesprache der Kompilierungsphase Code Generation und erläutern Sie in Stichpunkten deren Vorgehen und Aufgaben.

b) Angenommen die Syntax eines vorhandenen Triangle-Befehls ändert sich marginal (etwa: Ersetzung des Zeichens zur Durchführung einer Variablenzuweisung). Welche Auswirkungen hat dies auf den Vorgang der Code Generation?

c) Beschreiben Sie detailliert, in welcher Situation das sogenannte Backpatching notwendig wird und was dessen Aufgabe ist.

d) Erläutern Sie, ob die Qualität einer Code-Schablone in Bezug auf deren Befehlsfolge Auswirkungen auf die Ausführungsgeschwindigkeit des erzeugten Programms hat.

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism