Metody kompilacji

Laboratorium 7

Piotr Błaszyński

21 kwietnia 2017

Zadania (wyjaśnienie w dalszej części dokumentu):

- dodać kompilację instrukcji pętli,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację tworzenia i używania statycznych tablic jednowymiarowych,
  - reguły gramatyki,
  - generowanie kodu wynikowego,

Pętle można realizować przy pomocy skoku warunkowego i licznika. Licznik jest zmienną, wartość tej zmiennej należy modyfikować w odpowiednim momencie (koniec pętli, początek pętli (za wyjątkiem pierwszej iteracji)).

Wyjściowe reguły gramatyki dla instrukcji pętli (podobna do pętli w C, wszystkie elementy wymagane):

```
for_expr
        :for_begin code_block
        {gen_etykiety_koncowej_i_skoku();}
for_begin
        :FOR '(' init_expr ';' cond_expr ';' inc_expr ')'
           ' {gen_warunku_i_skoku();}
```

Dla nastepującego kodu pętli:

```
for (i = 0 ; i < 10 ; ++i )
{
        z = z + i ;
}
z = z*3;
```

Należy wygenerowac następujący kod (symbolicznie):

```
 i=0;
 goto LBL5
LBL6:
 ++i;
LBL5:
 if(i>=10)
   goto LBL7:
 {
     z = z + i ;
 }
 goto LBL6:
LBL7:
 z = z*3;
```

Dzięki powyższej konstrukcji, nie ma potrzeby zapamiętywania kodu wyrażenia warunkowego i inkrementującego (ceną za takie uproszczenie jest dołożenie dodatkowej etykiety). Etykiety sa ponumerowane w kolejności ich pojawiania się (pierwszy pojawia się skok do etykiety LBL5). Etykiety (w przykładzie LBL6 i LBL7) nazleży zapamiętać na stosie etykiet. W akcji semantycznej wywoływanej po dopasowaniu całej konstrukcji (wraz z blokiem kodu) pętli nalżey zdjąć etykiety i wygenerować instrukcję skoku do drugiej oraz umieścić pierwszą etykietę w kodzie (z dwukropkiem).

Wygenerowany kod assemblera (mnemoniki):

```
 li $t0,0
 sw $t0,x
 b LBL5
LBL6:
 #te 4 linie mozna prosciej zapisac (ale nie trzeba)
 lw $t0, i
 li $t1, 1
 add $t0, $t0, $t1
 sw $t0, i
LBL5:
 lw $t2, i
 li $t3, 10
 bge $t2, $t3, LBL7
 lw $t0, z
 lw $t1, i
 add $t0, $t0, $t1
 sw $t0, result1
 lw $t0, result1
 sw $t0, z
 b LBL6
LBL7:
 lw $t0, z
 li $t1, 3
 mul $t0, $t0, $t1
 sw $t0, result2
 lw $t0, result2
 sw $t0, z
```

W ty **Uwaga**, to jest jedna możliwa metoda generowania kodu dla instrukcji pętli, możliwych modyfikacji jest jeszcze wiecej niż przy instrukcjach warunkowych. Można np. zmienić położenie skoków, zapamietywać inne wartości.

Obsługa **tablic jednowymiarowych** składa się z dwóch elementów: zapamiętania rozmiaru tablicy i odwoływania się do elementu tablicy (uwaga: przypominam, że odwołania mogą występować po lewej i prawej stronie wyrażeń).

Wyjściowe reguły gramatyki dla deklaracji (uproszczone w stosunku do C - w C rozmiar można określić wyrażeniem stałym i deklarować wiele zmiennych):

```
arr_decl
 :INT ID '[' LC ']' ';'
       {tablica_symboli[$2], typ=ARRI, rozmiar=$4; }
 :FLOAT ID '[' LC ']' ';'
       {tablica_symboli[$2], typ=ARRF, rozmiar=$4; }
```

Wyjściowe reguły gramatyki dla indeksowania:

```
arr_expr
        :ID '[' wyr ']'
```

Rozmiar tablicy w MIPS podajemy po dwukropku (lepiej użyć *.word*). Przy odwoływaniu się do tablicy należy pomnożyć przez 4 wartość przesunięcia względem adresu początkowego, adres początku tablicy ładujemy przy pomocy *la*. Wartość pod adres wpisujemy lub pobieramy przy pomocy nawiasów okrągłych (np. *sw* $\$t0, (\$t4)$)
Dla kodu:

```
int a[10];
a[3] = 2;
x = a[2+1];
```

Należy wygenerowac następujący kod (mnemoniki):

```
.data
        a: .word 0:10
        x: .word 0
        result1: .word 0
.text
li $t0, 2
la $t4, a
li $t5, 3
mul $t5, $t5, 4
add $t4, $t4, $t5
sw $t0, ($t4)

li $t0, 2
li $t1, 1
add $t0, $t0, $t1
sw $t0, result1
la $t4, a
lw $t5, result1
mul $t5, $t5, 4
add $t4, $t4, $t5
lw $t0, ($t4)
sw $t0, x
```