Metody kompilacji

Laboratorium 8

Piotr Błaszyński

5 maja 2018

Zadania (wyjaśnienie w dalszej części dokumentu):

- dodać obsługę liczb zmiennoprzecinkowych,
  - generowanie kodu obliczeń,
  - konwersja lub zgłoszenie błędu,
- dodać obsługę statycznych tablic wielowymiarowych,
  - reguły gramatyki,
  - tworzenie wpisu w tablicy symboli,
  - obliczanie indeksu,

Obsługę liczb zmiennoprzecinkowych w MIPS realizuje się przy pomocy innych rejestrów oraz zestawu instrukcji niż ich całkowite odpowiedniki. Ponadto, należy uprościć implementację odwołań do stałych zmiennoprzecinkowych poprzez traktowanie ich jak zmiennych. Rejestry zmiennoprzecinkowe będące odpowiednikami rejestrów $\$t0 - \$t7$ to $\$f0 - \$f31$ - do przechowywania liczb typu float używa się pojedynczego rejestru, do przechowywania liczb typu double używa się pary rejestrów. Wymaganie w kompilatorze dotyczy tylko liczb typu float, więc dalej będą opisywane tylko instrukcje ich dotyczące. Do załadowania wartości zmiennej do rejestru służy instrukcja: $l.s\ rejestr, zmienna$. Do przechowania wartości z rejestru do zmiennej służy instrukcja: $s.s\ rejestr, zmienna$. Do wykonywania operacji arytmetycznych operacje:

- $add.s\ rejestr_wynik, rejestr_arg1, rejestr_arg2$ - dodawanie,
- $sub.s\ rejestr_wynik, rejestr_arg1, rejestr_arg2$ - odejmowanie,
- $mul.s\ rejestr_wynik, rejestr_arg1, rejestr_arg2$ - mnożenie,

- $div.s\ rejestr_wynik, rejestr_arg1, rejestr_arg2$ - dzielenie,

Dla przykładowego kodu dodającego dwie liczby zmiennoprzecinkowe:

```
z=3.14+6.28;
y=3.14+5.12;
```

Należy wygenerować (stała może wystąpić w sekcji danych raz lub wielokrotnie, jeżeli występuje tylko raz to jej wszystkie wystąpienia zastępujemy tym samym identyfikatorem):

```
.data
  z: .float 0
  float_var1: .float 3.14
  float_var2: .float 6.28
  y: .float 0
  float_var3: .float 5.12
  tmp1: .float 0
  tmp2: .float 0
.text
  l.s  $f0, float_var1
  l.s  $f1, float_var2
  add.s $f0, $f0, $f1
  s.s $f0, tmp1

  l.s $f0, tmp1
  s.s $f0, z

  l.s  $f0, float_var1
  l.s  $f1, float_var3
  add.s $f0, $f0, $f1
  s.s $f0, tmp2

  l.s $f0, tmp2
  s.s $f0, y
```

W przypadku jeżeli język zakłada możliwość konwersji pomiędzy zmiennymi typów zmiennoprzecinkowych i całkowitych, konieczne jest załadowanie wartości do rejestru, a następnie wywołanie konwersji - konwersje w obie strony wykonuje się na rejestrze zmiennoprzecinkowym.

Konwersja z wartości całkowitej na zmiennoprzecinkową:

```
.text
  li $t0, 10
  mtc1 $t0, $f0
  cvt.s.w $f1, $f0
```

Konwersja z wartości zmiennoprzecinkowej na całowitą:

```
.data
 float_var1: .float 3.14
.text
  li $t0, 10
  mtc1 $t0, $f0
  cvt.s.w $f1, $f0
  l.s $f2, float_var1
  add.s $f1, $f1, $f2
  cvt.w.s $f0, $f1
  mfc1 $t0, $f0
```

Obsługa tablic wielowymiarowych składa się z dwóch elementów: odczytu rozmiarów tablicy (np. z deklaracji) i odwoływania się do elementów tablicy.

Wyjściowe reguły gramatyki dla tablic wielowymiarowych (to jest tylko propozycja):

```
arr_decl
        :arr_start dim_decl {;}
        ;

arr_start
        : arr_type ID
        ;

dim_decl
        : '[' size_const ']' {;}
        ;

size_const
        : size_const ',' size_value {;}
        | size_value    {;}
        ;

size_value
        : LC {;}
        ;
```

Dla tablic wielowymiarowych konieczne jest zapamiętanie w tablicy symboli rozmiarów dla poszczególnych wymiarów. Można (bardzo przydatne) zapamietać rozmiary pojedynczego elementu dla poszczególnych wymiarów.
Dla deklaracji:

```
int a[4,3,5];
```

Informacja o tablicy powinna zawierać takie informacje:

```
dims: [4,3,5]
sizes: [15,5,1]
```

Przy generowaniu odwołań do tablic postępowanie wygląda podobnie jak przy tablic jednowymiarowych, z tą różnicą, że należy odpowiednie odwołania mnożyć przez wartość z informacji o rozmiarach (sizes). Należy również pamiętać o mnożeniu wartości przez 4 (można te mnożenia połączyć w jedno).