Bryan Lee <blee@etud.u-pem.fr>
Raphaël Cheneau <rcheneau@etud.u-pem.fr>
L3 Informatique (Traduction)

31/05/2015

Rapport de projet
Compilateur TPC

Chargé de TP : **Eric LAPORTE**

SOMMAIRE

1) Présentation générale du projet

Le but du projet est de concevoir un compilateur pour le langage de programmation « TPC », au moyen des outils flex et bison. Il devra traduire un code écrit en TPC en un code en langage d'une machine virtuelle, qui saura l'exécuter.

Le présent rapport décrit les choix que nous avons effectués dans la conception du compilateur.

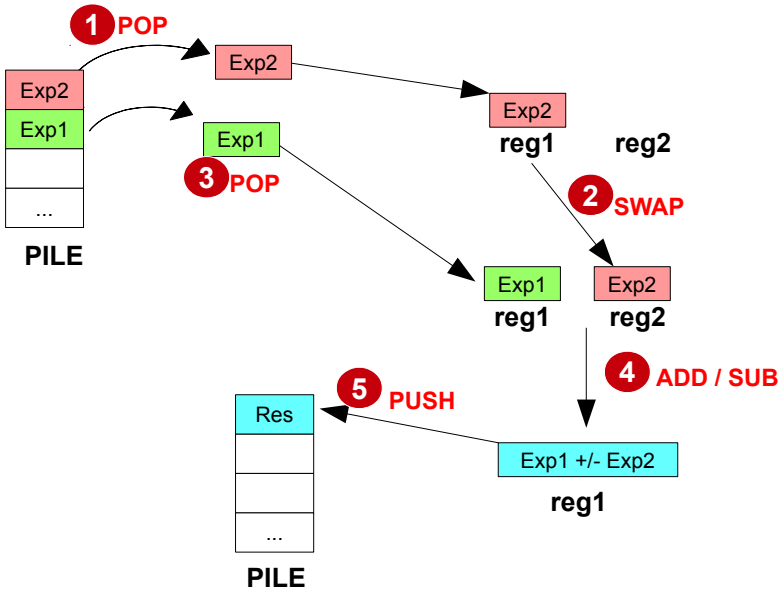
2) Opérations et comparaisons

Les opérations et comparaisons du langage étant traduits en utilisant des raisonnements très similaires, nous allons en décrire une.

2.1 – Exp ADDSUB Exp

L'entité lexicale `ADDSUB` est définie par une expression régulière dans l'analyseur lexical. Cette dernière indique en `ADDSUB` si l'opérateur `+` ou `-` a été utilisé.

Ayant des expressions `Exp1` et `Exp2` évalués et pushés sur la pile :



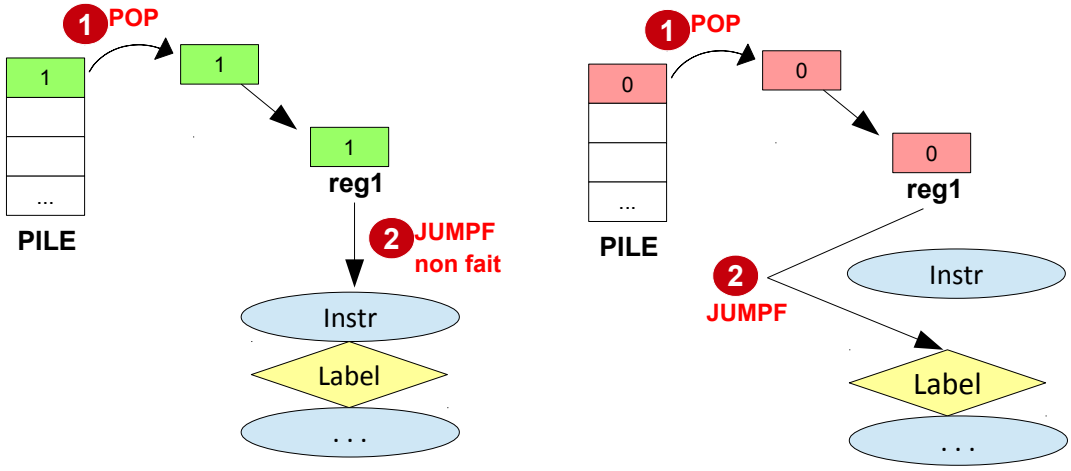
3) Structures conditionnelles

3.1 – IF simple

Nous introduisons une règle `Jumpif` qui évitera le bloc `Instr` et passera directement au `Label` si `Exp` est fausse.

```
if ( Exp ) {
  Jumpif 1
  Instr
}
Label 1
...
```

Ayant en tête de pile le résultat de l'évaluation de `Exp` , c'est-à-dire `1` si vrai, `0` si faux :



3.2 – IF ELSE

Même raisonnement, sauf que nous introduisons une règle **`Jumpelse`** qui évitera d'entrer dans le bloc `else` en allant directement sur **`Label`** , dans le cas où `Exp` est vraie.

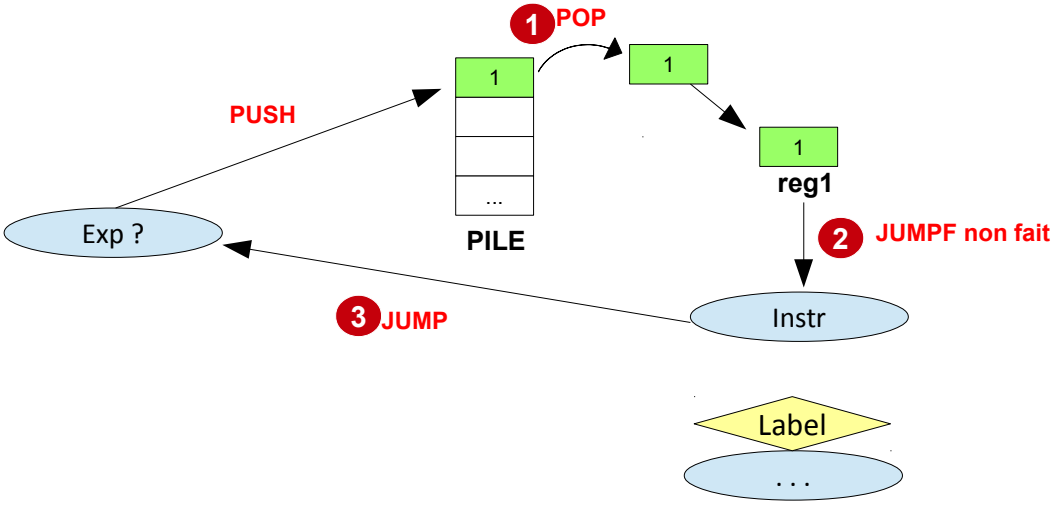
```
if ( Exp ) {
  Jumpif 1
  Instr
  Jumpelse 2
}
else {
  Label 1
  Instr
}
Label 2
...
```

3.3 – Boucles WHILE

Nous posons un `Label` avant l'évaluation de l'expression `Exp`.
Si `Exp` est vraie, on exécute le bloc `instr`. Puis nous revenons au `Label` pour réévaluer l'expression `Exp`.
Sinon, `Jumpwhile` permet d'éviter le bloc `while` et passe directement au `Label` pour exécuter la suite.

```
while Label 1 ( Exp ) {
    Jumpwhile 2
    instr
    Jump 1
}
Label 2
...
```

Exemple avec une expression `Exp` évaluée vraie :



4) Table des symboles

4.1 – Première tentative : problème

Un premier essai de table des symboles consistait à stocker les identifiants en les associant avec leur valeur.
Cela entraînait un problème lors de l'exécution d'une boucle `while` par exemple : si l'expression conditionnelle du `while` utilisait une variable comme compteur, la traduction écrirait la valeur « brute » de la variable dans le code machine. Cela rendrait la réévaluation de la condition totalement impossible car la valeur de la variable ne serait jamais mis à jour dans le code machine.

4.2 – Solution

Au lieu d'associer un identifiant avec sa valeur, nous allons l'associer avec son adresse dans la pile.

4.3 – Types structurés utilisés

Symbole	**TS** (Table des symboles)
- Un identifiant : une chaîne de caractères - Un type : représenté par un entier - Une adresse : représenté par un entier Représentation de tableaux : - Un entier : si l'id définit un tableau, vaut le nombre de dimensions, sinon 0 - Un tableau `dimension` indiquant les tailles des dimensions d'un tableau ex : `tab[5][10]` alors `dimension[0] = 5` `dimension[1] = 10`	- Un tableau de Symboles - L'index de la 1ère case vide

Fonction	**TSfonc** (Table des symb pour fonctions)
- Une table des symboles - Un symbole - Le nombre d'arguments - Un tableau contenant les identifiants des arguments - Un tableau contenant les types des arguments - Un tableau contenant les adresses des arguments	- Un tableau de Fonctions - L'index de la 1ère case vide

4.4 – Affecter une valeur à une variable

Pour affecter une valeur à une variable (identifiant), nous avons une fonction `setID` qui :

Ayant la valeur à affecter en tête de pile (préalable PUSHée)
(1) Vérifie si la table des symboles en cours contient l'identifiant
(2) Si elle ne le contient pas, vérifie la table des symboles des variables globales
(3) Récupère l'adresse de l'identifiant
(4) Met reg1 à l'adresse de l'identifiant (SET)
(5) Met reg2 à l'adresse de l'identifiant (SWAP)
(6) Met reg1 à la valeur à affecter (POP)
(7) Stocke la valeur de reg1 à l'adresse indiquée par la valeur de reg2 (SAVE ou SAVER selon si l'on est dans une fonction ou dans le main).

Le principe pour récupérer la valeur d'un identifiant est à peu près le même sauf que l'on se sert de LOAD/LOADR.

Les valeurs des adresses sont gérées grâce à la variable globale `last_free_adresse` qui est incrémentée chaque fois que l'on insère une variable dans la table des symboles.

Cette variable est remise à 0 pour chaque fonction puis est restaurée à sa valeur antérieure (avant les déclarations de fonctions) pour le main.

Pour savoir dans quelle table des symboles insérer/récupérer une valeur on se sert d'un pointeur `cur_ts` qui pointera sur la table en cours.

5) Les fonctions

Les fonctions ont une table des symboles différentes des variables. Le type d'une fonction ici sera sa valeur de retour, son adresse la valeur de son label.

On doit en plus savoir son nombre d'arguments, leur type et leur identifiant. Tout ceci est fait grâce à la fonction `insertFonction`.

Au moment de l'appel de fonction, on va se charger de remplir la table des symboles de cette fonction avec les valeurs transmises, grâce à la fonction `setFonction` .
On vérifie ensuite qu'il ne manque aucun argument et qu'ils respectent tous le bon type, puis on s'occupe d'y ajouter les variables locales.

Au moment du `RETURN` , si la fonction n'est pas de type `void` , on se charge de mettre dans le registre 1 la valeur de l'expression de retour.

6) Les tableaux

Lors de l'insertion d'un tableau dans la table des symboles, certaines informations sont ajoutées :
- le nombre de dimensions qu'il possède
- la taille de chacune de ces dimensions

Lorsque l'on rencontre une variable de type tableau, la variable `is_tab` est mise à `1`.
Ainsi, lors de son affectation, ou quand on veut récupérer sa valeur, on la traitera différemment d'une variable classique.

Pour obtenir l'adresse à l'indice `i` d'un tableau, on utilise la fonction `setArrIndex` qui va placer en tête de pile son adresse. Récupérer ou affecter entraînera un agencement de la pile légèrement différent (affecter un élément ajoute un élément en plus sur la pile : la valeur de l'expression).

Description de l'algorithme :

Soit `n` le nombre de dimensions.

Pour `i` allant de `0` à `n-1`
Récupérer l'indice de la i-ème dimension
Multiplier l'indice par la taille des dimensions `i+1` à `n`
PUSHer le résultat sur la pile. Pour la dernière dimension `n`, on va juste POPer les résultats des précédentes multiplications et l'ajouter au fur et à mesure à ce `n`. Enfin on ajoute au résultat la valeur de l'adresse du tableau (soit la case 0).

Exemple :

Soit le tableau `tab[2][3][4];`

Calcul de l'adresse de l'élément désigné par `tab[1][0][3]` :

```
  1*(3*4)
+ 0*4
+ 3   auquel on ajoute la valeur de l'adresse du tableau.
```