PLP - Laboratoire 3

Compilation

22 janvier 2020

Étudiants
Doran Kayoumi
Chau Ying Kot

Professeur
Marc Dikötter
Assistant
Nadir Benallal

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

1 Introduction

Ce laboratoire fait suite au dernier laboratoire rendu le 18 décembre 2019, il nous ait demandé de créer un compilateur et générer un code "assembleur" qui sera exécuté par notre machine abstraite *SAM*.

2 Compilation

Pour générer le code qui sera exécuté par notre machine abstraite, notre compilateur va générer du code pour chaque "type" d'instruction.

Notre compilateur peut génère du code pour les instructions suivante :

- Constant de type int
- Opérateur binaire : addition, soustraction et multiplication
- If/Else
- `Let`
- Variables
- Définition de fonction
- L'appel de fonction

Notre compilateur ne peut pas compiler des fonctions appelant d'autre fonction ou soi-même.
Les opérations binaires utilisent la notation polonaise inversée.

2.1 Résultat de notre compilateur

```
*Lab13> compile [(Let "x" (Cst 2) (Let "y" (Cst 3) (Bin '+' (Var "x") (Var "y"))))]
[JMP 0,LINK 2,INT 2,STORE 1,INT 3,STORE 2,LOAD 1,LOAD 2,ADD,DOT,HALT]
```

FIGURE 1 – Compilation du `Let`

```
*Lab13> compile  [Def "square" ["x"] (Bin '*' (Var "x") (Var "x")), Let "x" (Let "y" (Cst 4) (App "square" [Var "y"])) (App "square" [Var "x"])]
[JMP 12,LINK 0,LOAD (-2),LOAD (-2),MPY,STORE (-2),UNLK,EXIT 0,LINK 2,INT 4,STORE 1,LOAD 1,CALL 2,STORE 1,LOAD 1,CALL 2,DOT,HALT]
```

FIGURE 2 – Compilation de la définition de la fonction square

3 Machine abstraite

Notre machine *SAM* est une machine à pile qui est capable de traiter, en plus des instructions données, les instructions suivantes :

- `SUB` : Effectue une soustraction entre les deux premières valeurs de la stack et met le résultat sur le dessus de celle-ci.
- `LINK n` : Alloue `n` espace mémoire pour les variables en déplaçant le stack pointer de `n` emplacement.
- `STORE n` : Déplace la valeur à l'emplacement `n` de la zone mémoire pour les variable en haut de la pile.
- `LOAD n` : Met une copie (charge) de la valeur qui se trouve à la position `n`, de la zone mémoire pour les variables, sur le haut de la pile.
- `CALL n` : Il correspond à un appel de fonctions, `n` est l'adresse absolue de la fonction. Avant modifier l'instruction pointer pour qu'elle pointe sur la fonction, on va sauvegarder sur la pile l'emplacement de l'instruction suivant le `CALL` à exécuter après l'exécution de la fonction.
- `UNLK` : Désalloue les ressources utiliser pour l'exécution de la fonction et remette le frame pointé à la valeur avant l'appel de la fonction
- `EXIT n` : On désalloue les `n` espaces réservés pour les paramètres de la fonction, on remette l'instruction pointer à la prochaine instruction à exécuter

3.1 Résultat d'exécution

```
[  0] JMP    40   SP  0 :
[ 42] LINK    0   SP  1 :   0
[ 44] INT     5   SP  2 :   0   5
[ 46] CALL   14   SP  3 :   0   5  48
[ 14] LINK    1   SP  5 :   0   5  48   0   0
[ 16] LOAD   -2   SP  6 :   0   5  48   0   0   5
[ 18] JZR    15   SP  5 :   0   5  48   0   0
[ 20] LOAD   -2   SP  6 :   0   5  48   0   0   5
[ 22] CALL    2   SP  7 :   0   5  48   0   0   5  24
[  2] LINK    0   SP  8 :   0   5  48   0   0   5  24   3
[  4] LOAD   -2   SP  9 :   0   5  48   0   0   5  24   3   5
[  6] INT     1   SP 10 :   0   5  48   0   0   5  24   3   5   1
[  8] SUB         SP  9 :   0   5  48   0   0   5  24   3   4
[  9] STORE  -2   SP  8 :   0   5  48   0   0   4  24   3
[ 11] UNLK        SP  7 :   0   5  48   0   0   4  24
[ 12] EXIT    0   SP  6 :   0   5  48   0   0   4
[ 24] STORE   1   SP  5 :   0   5  48   0   4
...
[ 39] UNLK        SP  8 :   0   5  48   0   4   5  24  32
[ 40] EXIT    0   SP  7 :   0   5  48   0   4   5  24
[ 32] MPY         SP  6 :   0   5  48   0   4 120
[ 33] JMP     2   SP  6 :   0   5  48   0   4 120
[ 37] STORE  -2   SP  5 :   0 120  48   0   4
[ 39] UNLK        SP  3 :   0 120  48
[ 40] EXIT    0   SP  2 :   0 120
[ 48] DOT         SP  1 :   0
120
```

Listing 1 – Une partie de la sortie de l'exécution de la fonction factorielle de 5

4 Résultats

```
let x = 2 in
   let y = 3 in x + y
```

Listing 2 – Code à compiler

```
JMP 0
LINK 2
INT 2
STORE 1
INT 3
STORE 2
LOAD 1
LOAD 2
ADD
DOT
HALT
```

Listing 3 – Résultat de la compilation

```
[   0] JMP     0   SP  0 :
[   2] LINK    2   SP  3 :    0   0   0
[   4] INT     2   SP  4 :    0   0   0   2
[   6] STORE   1   SP  3 :    0   2   0
[   8] INT     3   SP  4 :    0   2   0   3
[  10] STORE   2   SP  3 :    0   2   3
[  12] LOAD    1   SP  4 :    0   2   3   2
[  14] LOAD    2   SP  5 :    0   2   3   2   3
[  16] ADD         SP  4 :    0   2   3   5
[  17] DOT         SP  3 :    0   2   3
5
```

Listing 4 – Sortie de la machine SAM