Machine virtuelle de l'université Paris-Est Marne-la-Vallée

Cours de compilation L3

12 mai 2016

1 Introduction

Dans le cadre du cours de compilation du L3 informatique de l'université Paris-Est Marne-la-Vallée, nous utiliserons une *machine virtuelle*. Il s'agit d'une machine abstraite à registres. Elle est composée de :

- un segment de code ;
- une pile de données ;
- deux registres de travail `reg1` et `reg2` ;
- un registre `base` qui pointe sur un emplacement particulier de la pile et qui est manipulé uniquement par la machine (lors des appels de fonction et des retours) ;
- un registre `counter` qui pointe sur un emplacement particulier du segment de code, c'est le compteur ordinal de la machine.

Le segment de code contient la représentation du *v-code* exécuté par la machine. Cette partie de la mémoire est statique et reste inchangée durant l'exécution de la machine virtuelle. L'adresse du début de ce segment est `0`. Ce segment de code peut être vu comme un tableau, chaque case contenant un entier représentant soit une instruction, soit l'argument d'une instruction.

La pile de données permet de stocker toutes les valeurs nécessaires à l'exécution du programme : variables globales, locales et temporaires, mais elle sert aussi de pile d'exécution : lors d'un appel de fonction, on y stocke l'état de la machine virtuelle.

Les deux registres de travail sont ceux sur lesquels s'effectuent toutes les opérations. Le registre `base` permet d'indiquer la portion de la pile qui correspond à l'exécution de la fonction courante.

2 Langage de la machine virtuelle

Le langage de la machine virtuelle est très simple. Chaque ligne contient une commande, et éventuellement un argument (entier) pour cette commande. Les commentaires sont introduits par `#` et durent jusqu'à la fin de la ligne courante.

Les commandes sans argument acceptées par la machine virtuelle sont représentés sur la Figure 1, et celles possédant un argument sur la Figure 2.

NOP	:	Ne fait rien du tout
NEG	:	`reg1`$\leftarrow \neg$`reg1`
ADD	:	`reg1`$\leftarrow$`reg1`+`reg2`
SUB	:	`reg1`$\leftarrow$`reg1`$-$`reg2`
MUL	:	`reg1`$\leftarrow$`reg1`$*$`reg2`
DIV	:	`reg1`$\leftarrow$`reg1`/`reg2`
MOD	:	`reg1`$\leftarrow$`reg1`%`reg2`
EQUAL	:	`reg1`$\leftarrow$`reg1`$=$`reg2`
NOTEQ	:	`reg1`$\leftarrow$`reg1`$\neq$`reg2`
LESS	:	`reg1`$\leftarrow$`reg1`$<$`reg2`
LEQ	:	`reg1`$\leftarrow$`reg1`$\leqslant$`reg2`
GREATER	:	`reg1`$\leftarrow$`reg1`$>$`reg2`
GEQ	:	`reg1`$\leftarrow$`reg1`$\geqslant$`reg2`
PUSH	:	Place la valeur de `reg1` sur la pile
POP	:	Place la valeur en tête de pile dans `reg1` et dépile
SWAP	:	Échange les valeurs de `reg1` et `reg2`
READ	:	Lit un entier et le stocke en `reg1`
READCH	:	Lit un caractère et le stocke en `reg1` (comme entier)
WRITE	:	Affiche la valeur stockée en `reg1`
WRITECH	:	Affiche le contenu de `reg1` vu comme un caractère
HALT	:	Arrête l'exécution de la machine
RETURN	:	Termine l'activation de la fonction en cours et retourne à la fonction appelante (voir plus loin)
LOAD	:	Place dans `reg1` la valeur située à l'adresse `reg1` de la pile
LOADR	:	Place dans `reg1` la valeur située à l'adresse `reg1`+`base` de la pile
SAVE	:	Stocke la valeur de `reg1` à l'adresse `reg2` de la pile
SAVER	:	Stocke la valeur de `reg1` à l'adresse `base`+`reg2` de la pile
TOPST	:	Place dans `reg1` la position courante du sommet de la pile
BASER	:	Place dans `reg1` la valeur contenue dans le registre `base`

FIGURE 1 – Instructions de la machine virtuelle

SET	n	:	`reg1`$\leftarrow n$
LABEL	n	:	Déclare le label numéro n
JUMP	n	:	Branchement à l'emplacement du label n dans le segment de code
JUMPF	n	:	Branchement à l'emplacement du label n dans le segment de code si `reg1` vaut 0
ALLOC	n	:	Effectue n fois l'opération PUSH 0
FREE	n	:	Effectue n fois l'opération POP, sans modifier la valeur du registre 1
CALL	n	:	Sauvegarde dans la pile une partie de l'état de la machine et effectue un branchement à l'emplacement du label n dans le segment de code

FIGURE 2 – Instructions avec argument

Après chaque instruction, on passe à l'instruction suivante (incrémentation du registre `counter`), excepté pour les instructions `JUMP`, `JUMPF` et `CALL` qui induisent explicitement un branchement dans le segment de code et l'instruction `RETURN` qui provoque un branchement dépendant des instructions stockées lors de l'activation de la fonction.

Notez que lors du chargement du programme par la machine virtuelle, les labels sont effacés et les branchements se font selon l'adresse des instructions dans le segment de code.

L'appel de `CALL` provoque l'empilement successif d'un pointeur indiquant quelle instruction exécuter au retour de la fonction, et de `base`. À la suite de quoi, `base` est mis à jour pour pointer sur le sommet de la pile d'exécution qui correspondra au début de la zone dédiée à la fonction appelée. Attention, les autres registres ne sont pas automatiquement sauvegardés lors de l'appel d'une fonction.

`RETURN` supprime la portion de pile dédiée à la fonction courante, y compris les valeurs empilées lors de l'appel de la fonction, et restaure `base` ainsi que le pointeur sur le segment de code.

3 Appel de fonction

L'appel de fonction ne gère pas les paramètres, et la commande `RETURN` ne gère pas la valeur de retour. Pour appeler une fonction avec n arguments, on pourra empiler ces n arguments sur la pile d'exécution puis procéder à l'appel de la fonction. Leur adresse relative sera alors située entre $-n-2$ et -3, les emplacements d'adresse respective -2 et -1 étant occupés par les informations nécessaires à la restauration de `counter` et `base` lors du retour de la fonction. La valeur de retour pourra, elle, être placée dans `reg1`. Une autre option peut être de lui réserver, par convention, le premier espace de la pile.

4 Exemple

Exemple de code accepté par la machine virtuelle :

```
#       Commande Arg
         SET     4       #reg1 := 4
         PUSH            #push 4
         SET     8       #reg1 := 8
         SWAP            #reg2 := 8
         POP             #reg1 := 4
         LESS            #reg1 := 4<8
         JUMPF   0       #goto 0 si reg1=0
         SET     3       #reg1 := 3
         PUSH            #push 3
         SET     5       #reg1 := 5
         SWAP            #reg2 := 5
         POP             #reg1 := 3
         LESS            #reg1 := 3<5
         JUMPF   1       #goto 1 si reg1=0
```

```
SET     4       #reg1 := 4
PUSH            #push 4
SET     70      #reg1 := 70
SWAP            #reg2 := 70
POP             #reg1 := 4
ADD             #reg1 := 4+70
WRITE           #---affichage 74
LABEL   0
LABEL   1
SET     12      #reg1 := 12
PUSH            #push 12
SET     5       #reg1 := 5
SWAP            #reg2 := 5
POP             #reg1 := 12
LESS            #reg1 := 12<5
JUMPF   2       #goto 2 si reg1=0
SET     97
WRITE           #---affichage 97
WRITECH         #---affichage a
LABEL   2
HALT
```

5 Licence

Cette documentation est fournie avec la machine virtuelle `virtual_mlv`. La machine virtuelle et la documentation sont placées sous licence GPL, v3. Ses auteurs sont Nicolas Bedon, Quentin Campos, Gaël Fuhs, Wuthy Hay, Sylvain Lombardy, Jefferson Mangue, Christophe Morvan et Céline Noël.

Le code est disponible à l'adresse `https://github.com/cmorvan/virtual_mlv`.

Quand on invoque la machine virtuelle avec l'option `-debug`, elle affiche le segment de code puis l'exécute instruction par instruction au fur et à mesure que l'utilisateur appuie sur la touche Entrée. Après chaque instruction, elle affiche le contenu de la pile de données.

Avec l'option `-help`, elle imprime les options disponibles.