Polytech Paris-Sud – 4ème année Informatique –
Projet Compilation – Description du Code intermédiaire

L'interprète du code intermédiaire est une machine à pile. Trois registres *sp*, *fp* et *gp* repèrent respectivement le sommet courant de la pile (stack pointer), l'adresse de base des variables locales (frame pointer), et l'adresse de base des variables globales (global pointer). La pile d'exécution contient des entiers et des adresses. Les chaînes de caractères et les objets sont stockées dans un « tas » et repérés par leurs adresses. Un registre *pc* contient l'adresse de l'instruction courante, et une pile annexe permet de gérer les appels de fonctions. La figure 1 décrit cette machine.

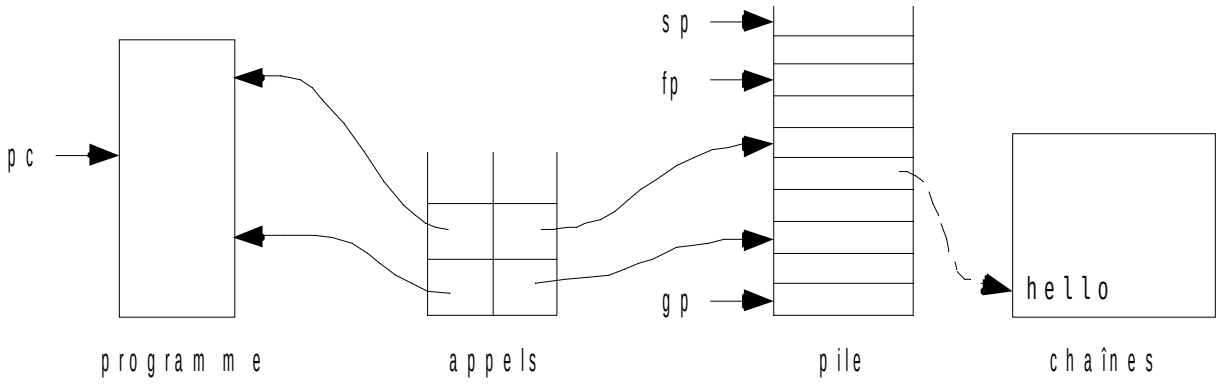

Figure 1 - La machine qui exécute le code intermédiaire

L'état initial de la machine, une fois un programme chargé, est le suivant :
- *gp* et *sp* pointent sur la base de la pile d'exécution ;
- la valeur de *fp* est indéfinie ;
- *pc* désigne la première instruction du programme chargé ;
- la pile des appels est vide.

Le jeu d'instructions de cette machine est décrit ci-dessous. Il utilise les conventions suivantes :
- *n* et *p* désignent des constantes entières ;
- *str* désigne une constante chaîne de caractères entre guillemets, avec les mêmes conventions que le langage C pour ce qui concerne les caractères spéciaux \", \n, \\ ;
- *addr* désigne une adresse du tas, une adresse dans le code ou une adresse dans la pile.
- *label* désigne une adresse dans le programme, relative à son début, ou bien une étiquette symbolique (voir la description de l'interpréteur) ;
- "empiler une ou plusieurs valeurs" signifie les mettre sur la pile et incrémenter *sp* du nombre de valeurs empilées ;
- "dépiler une ou plusieurs valeurs" signifie décrémenter *sp* du nombre de valeurs à dépiler.
- "dépiler un entier (resp. une chaîne)" signifie de plus que le type de la valeur dépilée doit être un entier (resp. une chaîne), sinon une erreur d'exécution est déclenchée.

Instruction	Description
NOP	ne rien faire.
ERR str	déclencher une erreur d'exécution, avec le message *str*.
START	affecter la valeur de *sp* à *fp*. Cette instruction ne peut être exécutée qu'une fois. Avant son exécution, l'exécution d'instructions qui utilisent *fp* provoque une erreur d'exécution, la valeur de *fp* n'étant pas définie.
STOP	arrêter l'exécution du programme.
PUSHI n	empiler la constante entière *n*.
PUSHS str	copier la chaîne *str* dans la zone des chaînes, et empiler son adresse.

PUSHG n	empiler la valeur située dans la pile en *gp[n]*.
PUSHL n	empiler la valeur située dans la pile en *fp[n]*.
PUSHSP	empiler l'adresse de pile contenue dans *sp*.
PUSHFP	empiler l'adresse de pile contenue dans *fp*.
STOREL n	dépiler une valeur et la stocker dans la pile en *fp[n]*.
STOREG n	dépiler une valeur et la stocker dans la pile en *gp[n]*.
PUSHN n	empiler *n* valeurs entières nulles.
POPN n	dépiler *n* valeurs.
DUPN n	empiler les *n* valeurs qui sont en sommet de pile (les dupliquer).
SWAP	échanger les valeurs en sommet et en sous-sommet de pile ; *sp* ne change pas.
EQUAL	dépiler 2 valeurs ; déclencher une erreur d'exécution si elles ne sont pas de même type (entier, adresse) ; empiler 1 si elles sont égales, 0 sinon.
NOT	dépiler un entier et empiler 1 s'il est nul, 0 sinon.
JUMP label	sauter à l'adresse *label* du programme.
JZ label	dépiler une valeur ; si elle est nulle, sauter à l'adresse *label* du programme, sinon passer à l'instruction suivante.
PUSHA label	empiler l'adresse dans le code correspondant à l'étiquette *label*.
CALL	dépiler une adresse *addr* ; empiler les valeurs courantes de *pc* et *fp* dans la pile des appels, affecter *sp* à *fp*, et sauter à l'adresse *addr*. (Voir figure 2)
RETURN	affecter *fp* à *sp*, dépiler de la pile des appels les valeurs de *fp* et *pc*, et incrémenter *pc* pour se retrouver à l'adresse suivant le CALL. (Voir figure 2)
ADD	dépiler deux valeurs entières et empiler leur somme.
SUB	dépiler deux valeurs entières et empiler leur différence (sous-sommet – sommet).
MUL	dépiler deux valeurs entières et empiler leur produit.
DIV	dépiler deux valeurs entières et empiler leur quotient (sous-sommet / sommet). Si le diviseur est nul, déclencher une erreur d'exécution.
INF	dépiler deux valeurs entières et empiler 1 si sous-sommet < sommet, 0 sinon.
INFEQ	dépiler deux valeurs entières et empiler 1 si sous-sommet ≤ sommet, 0 sinon.
SUP	dépiler deux valeurs entières et empiler 1 si sous-sommet > sommet, 0 sinon.
SUPEQ	dépiler deux valeurs entières et empiler 1 si sous-sommet ≥ sommet, 0 sinon.
WRITEI	dépiler un entier et l'imprimer sur la sortie standard.
STR	dépiler un entier et empiler l'adresse d'une chaine contenant la représentation de cet entier.
WRITES	dépiler l'adresse d'une chaîne et imprimer la chaîne sur la sortie standard
CONCAT	dépiler l'adresse de deux chaines et empiler l'adresse d'une chaine représentant leur concaténation (sous-sommet en tête)
STORE n	dépiler une valeur *v* et une adresse *addr* et stocker *v* en *addr[n]* ;
LOAD n	dépiler une adresse *addr* et empiler la valeur située en *addr[n]* ;
ALLOC n	alloue un objet contenant n champs et empile l'adresse de cet objet

L'interprète :

La machine décrite ci-dessus est émulée par un interprète qui prend en entrée un fichier texte contenant le programme. La syntaxe de ce fichier est la suivante :

- Une ligne comprend une instruction ou un commentaire ;
- Les instructions ont la forme suivante (les crochets dénotent les parties optionnelles) :

  [*][label:] code [arg1[, arg2]] [-- commentaire]

  L'étoile dénote un point d'arrêt lors de l'exécution de cette instruction (pour l'aide à la mise au point) ; *label* dénote une étiquette symbolique de branchement ; *code* est l'un des codes d'instructions décrits ci-dessus ; *arg1* et *arg2* sont les arguments du code opération. Un argument peut être un entier, un réel, une chaîne de caractère, ou un label.
- Les commentaires sont introduits par deux tirets (--). Ils doivent être sur des lignes séparées ou à la fin d'une instruction.

L'appel de l'interprète se fait par :

```
interp [-d] [file]
```

L'option *-d* active le mode mise au point décrit ci-dessous. L'argument *file* est le fichier contenant le programme à interpréter. Si cet argument est absent, le programme est lu sur l'entrée standard.

Lorsqu'un point d'arrêt est rencontré lors de l'exécution, ou lorsque l'on est en mode mise au point, l'interprète affiche la pile, l'instruction courante, et attend une commande parmi les suivantes :

c	continuer	a poser un point d'arrêt sur l'instruction courante
i	exécuter une instruction	d enlever le point d'arrêt de l'instruction courante
p	afficher la pile	+ afficher l'instruction suivante
t	activer / désactiver la trace	- afficher l'instruction précédente
q	quitter	? afficher les commandes disponibles
		<RC> répéter la commande précédente

En mode trace, la pile est affichée après l'exécution de chaque instruction. Le mode trace est désactivé par défaut. La pile est aussi affichée à chaque point d'arrêt, lors d'une interruption utilisateur (^C), et lors d'une erreur d'exécution. Lors de l'affichage de la pile, le commentaire associé à l'instruction qui a provoqué l'empilement de chaque élément de pile est affiché en face de chaque élément. Ainsi, après exécution de l'instruction

```
PUSHI 1         -- valeur de x
```

la pile aura l'aspect suivant :

```
|               1         -- valeur de x
------------
```

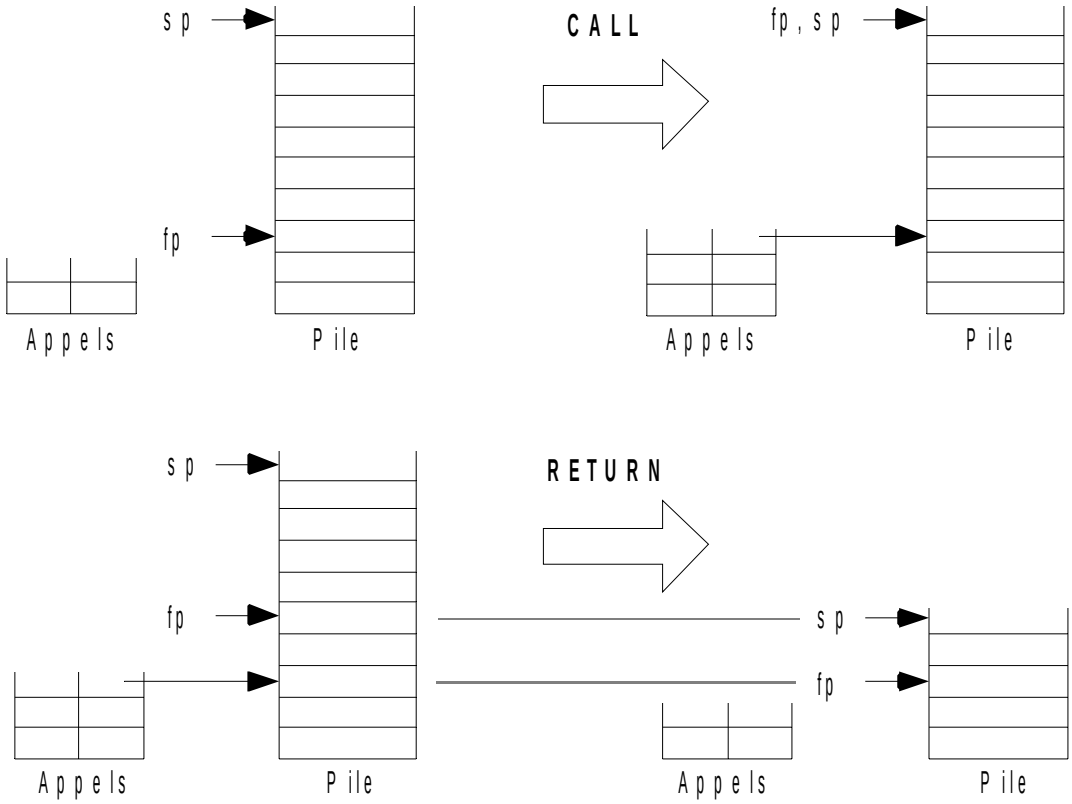

Figure 2 - fonctionnement de CALL et RETURN.
La sauvegarde/restitution de *pc* n'est pas représentée