Projet du cours « Machines Virtuelles »

Licence 3 – Université Paris Diderot

1er avril 2010

On vous demande d'implémenter une machine virtuelle à pile assez simple, très inspirée de la machine virtuelle OCaml. Il est fortement conseillé de lire l'intégralité de la spécification détaillée dans la section 1 avant de se lancer à corps perdu dans la programmation des réponses aux questions de la section 3. Par ailleurs, les instructions de rendu de votre travail sont données dans la dernière partie de ce document (section 4).

1 Spécification de la machine virtuelle

1.1 Conventions

Quand on parlera d'*octet* ou de caractère, on fera référence à un *mot de 8 bits*, vu comme un entier *non signé*.

Quand on parlera d'*entier 16 bits* ou plus simplement d'*entier*, on fera référence à un *mot de 16 bits*, vu comme un entier *non signé*. Toutes les opérations sur les entiers se feront donc modulo 2^{16}.

Les variables en minuscule comme , ou encore feront systématiquement référence à des octets. Celles en majuscules comme ou feront systématiquement référence à des entiers de 16 bits.

La concaténation de deux octets et se note et représente l'entier 16 bits dont l'octet de poids *fort* est et celui de poids *faible* est y.

1.2 Description de la machine virtuelle

La machine est composée de quatre éléments.

1. Le code qui est une suite d'octets et qui doit être chargé depuis un fichier. Les octets sont numérotés à partir de 0.
2. Le pointeur de code[1] qui pointe vers la prochaine instruction à exécuter. Après initialisation de la machine, il doit pointer vers la première instruction[2].
3. Un accumulateur qui peut contenir un entier de 16 bits (soit deux octets).
4. Une pile d'octets. Elle est initialement vide. On pourra se limiter à une pile de 2^{20} octets.

1.3 Fonctionnement de la machine

Chaque étape de fonctionnement de la machine se décompose de la façon suivante :

1. Elle décode l'instruction pointée par le pointeur de code.
2. Le pointeur de code est incrémenté de façon à pointer vers l'instruction suivante.
3. L'instruction décodée en 1 est exécutée.
4. Le cycle recommence.

La machine ne s'arrête qu'en rencontrant l'instruction EXIT. Si le pointeur de code pointe en dehors du code, le comportement est indéfini.

[1] *Program counter* dans la machine OCaml.
[2] C'est-à-dire vers l'octet de numéro 0.

1.4 Format des instructions

Les instructions sont de taille variable, sur un ou deux octets. L'*opcode*[3] se trouve toujours sur les 5 bits de poids fort du premier octet[4].

Les instructions sans argument—telles que PUSH ou ACCINT—sont représentées sur un seul octet (fig. 1).

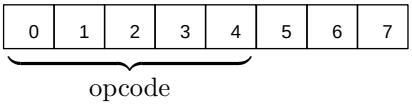


Fig. 1 – Instruction simple représentée sur un seul octet (ex. ACCINT)

Les instructions avec un argument de type octet—telles que CONST [5]—sont représentées sur un entier 16 bits, avec les 8 bits de poids faible représentant l'argument (fig. 2).

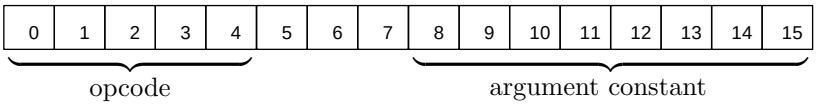


Fig. 2 – Instruction sur 16 bits avec un argument de type *octet* (ex. CONST)

Les instructions avec un argument de type entier—telles que CONSTINT [6]—sont représentées sur un entier 16 bits, avec les 11 bits de poids faible représentant l'argument (fig. 3).

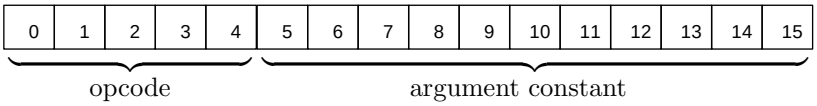
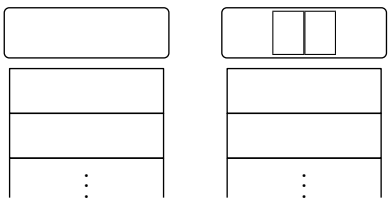


Fig. 3 – Instruction sur 16 bits avec un argument de type *entier* (ex. CONSTINT)

1.5 Description des schémas

Dans la suite, la description de chaque instruction de la machine virtuelle sera donnée de façon textuelle, mais sera aussi accompagnée d'un schéma montrant l'évolution de la pile et de l'accumulateur. La partie gauche du schéma montre la pile[7] et l'accumulateur[8] avant exécution de l'instruction, et la partie droite, les mêmes éléments après. Prenons l'exemple de l'instruction ACCINT dont l'évolution de la pile est montrée sur la figure 4.



Evolution de la pile (ACCINT)

Fig. 4 – Exemple de schéma montrant l'évolution de la pile avant et après l'exécution d'une instruction.

Avant exécution, l'accumulateur contient un entier 16 bits quelconque [9] et les deux premiers emplacements de la pile contiennent les octets et [10]. On ne sait rien sur l'état de la pile après . Elle peut contenir d'autres octets ou simplement être vide. Après exécution de l'instruction, la pile n'a pas évolué,

[3] C'est-à-dire l'entier qui détermine l'instruction.

[4] Donc du seul octet quand l'instruction est stockée sur un octet, et de l'octet de poids fort quand l'instruction est stockée sur deux octets.

[5] On notera le *n minuscule*.

[6] On notera le *N majuscule*.

[7] Partie basse du schéma. Les cases sont rectangulaires. Le haut de la pile est en haut !

[8] Partie haute du schéma. La case est arrondie.

[9] Notez une fois de plus l'usage de la majuscule X pour dénoter un *entier 16 bits*.

[10] Lettres minuscule y et z pour dénoter des *octets* donc sur 8 bits.

mais l'accumulateur contient maintenant , l'entier 16 bits obtenu par concaténation des deux octets et .

2 Instructions de la machine virtuelle

Cette section contient la description de la sémantique des instructions. Tout au long de cette spécification, on indique les tests qui permettent de valider incrémentalement l'implémentation des instructions. Ces tests sont fournis dans le répertoire `tests` de l'archive. La commande `make check` effectuée à la racine du projet permet de confronter la machine virtuelle à cet ensemble de tests.

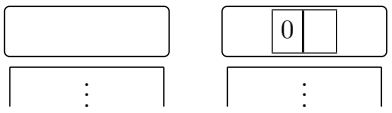
EXIT (opcode 0)

Arrête la machine avec comme code de retour l'octet de poid faible de l'accumulateur.

Pas de changement (EXIT)

CONST n (opcode 1)

Met à l'octet de poids faible de l'accumulateur et à 0 celui de poids fort.



Evolution de la pile (CONST)

PRINTCHAR (opcode 2)

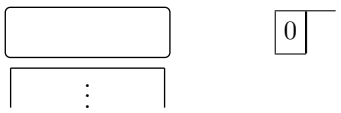
Affiche sur la sortie standard le caractère dont le code ASCII correspond à l'octet de poid faible de l'accumulateur.

Pas de changement (PRINTCHAR)

Le test 1 doit passer.

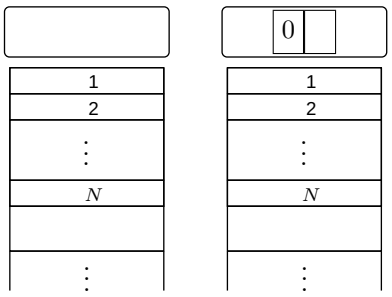
GETCHAR (opcode 3)

Lit un caractère sur l'entrée standard et met son code ASCII dans l'accumulateur. L'octet de poids fort est donc à 0. Si la fin de fichier est atteinte, met le mot de 16 bits avec uniquement des 1 dans l'accumulateur.



ACC N (opcode 6)

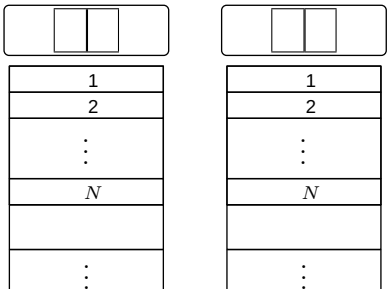
Récupère dans l'accumulateur le ($+1$)-ième octet de la pile. L'octet de poids fort de l'accumulateur est mis à 0.



Evolution de la pile (ACC)

ASSIGN N (opcode 7)

Remplace le ($+1$)-ième élément de la pile par l'octet de poids faible de l'accumulateur.

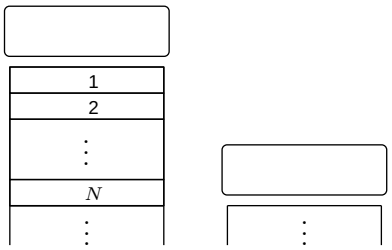


Evolution de la pile (ASSIGN)

le test 4 doit passer.

POP N (opcode 8)

Supprime les premiers éléments de la pile.

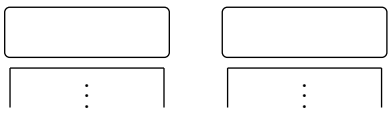


Evolution de la pile (POP)

le test 5 doit passer.

CONSTINT N (opcode 9)

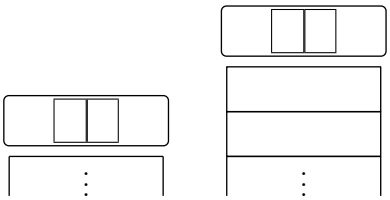
Met à l'accumulateur.



Evolution de la pile (CONSTINT)

PUSHINT (opcode 10)

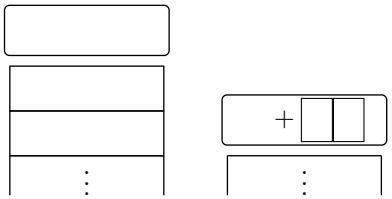
Pousse sur la pile l'octet de poids faible de l'accumulateur puis celui de poids fort.



Evolution de la pile (PUSHINT)

ADDINT (opcode 11)

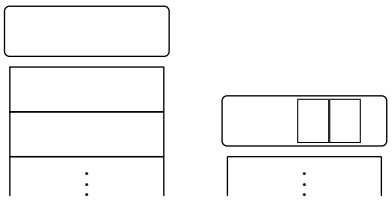
Ajoute à l'accumulateur l'entier sur le haut de la pile et supprime ce dernier.



Evolution de la pile (ADDINT)

MULINT (opcode 12)

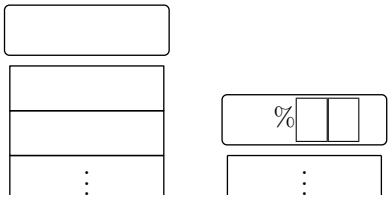
Multiplie l'accumulateur par l'entier sur le haut de la pile et supprime ce dernier.



Evolution de la pile (MULINT)

MODINT (opcode 13)

Divise l'accumulateur par l'entier sur le haut de la pile, supprime ce dernier et stocke le reste de la division dans l'accumulateur.

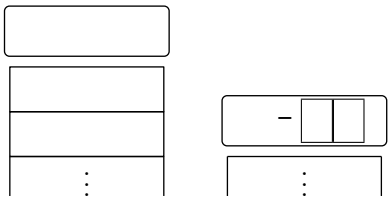


Evolution de la pile (MODINT)

Le test 6 doit passer.

SUBINT (opcode 14)

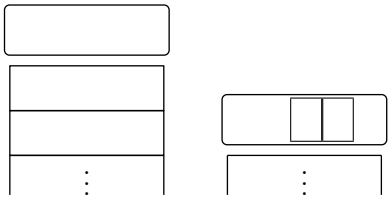
Retire à l'accumulateur l'entier sur le haut de la pile et supprime ce dernier. Comme toutes les oppérations, la soustraction se fait modulo 2^{16}. Par exemple, $0 - 1$ vaut 0xFFFF.



Evolution de la pile (SUBINT)

DIVINT (opcode 15)

Divise l'accumulateur par l'entier sur le haut de la pile et supprime ce dernier. Si l'entier sur le haut de la pile vaut 0, le comportement est indéfini.



Evolution de la pile (DIVINT)

Le test 7 doit passer.

BRANCH N (opcode 16)

Saute à l'instruction , c'est-à-dire que le pointeur de code pointe vers l'octet numéro . On rappelle que les octets du code sont numérotés à partir de 0.

Pas de changement (BRANCH)

Le test 8 doit passer.

BRANCHIF N (opcode 17)

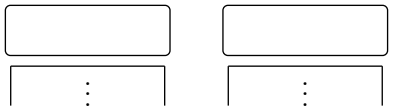
Saute à l'instruction , uniquement si le contenu de l'accumulateur est différent de 0.

Pas de changement (BRANCHIF)

Le test 9 doit passer.

BRANCHACC (opcode 18)

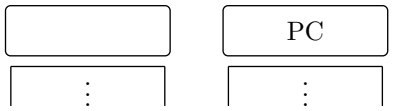
Saute à l'instruction où est le contenu de l'accumulateur. Le comportement n'est pas spécifié si cette instruction n'existe pas.



Evolution de la pile (BRANCHACC)

ACCPC (opcode 19)

Met dans l'accumulateur le numéro de l'instruction *suivante*.



Evolution de la pile (ACCPC)

Le test 10 doit passer.

3 Questions

3.1 Implantation de la machine virtuelle

Code fourni Dans le répertoire de travail, se trouvent trois sous-répertoires `ocaml`, `C` et `java`. Vous devez éditer le fichier nommé `[vV]m.(c|java|ml)` et utiliser la commande `make` à la racine du projet pour compiler. Notez dès à présent qu'un projet qui ne compile pas aura une note de 0. Il est donc recommandé de compiler au fur et à mesure du développement. Enfin, un jeu de tests est fourni dans le répertoire `tests`. Pour tester votre machine, vous devez lancer la commande `make check` à la racine du projet.

Test de décodage des instructions Pour vérifier votre décodage des instructions, nous vous demandons d'appeler, pour chaque instruction décodée, la fonction `trace_simpl(X)` pour les instructions simples, et `trace_cplx(X,n)` pour les instructions sur 2 octets, où `X` est le nom de l'instruction et `n` l'éventuel argument. Vous pouvez alors lancer votre vm avec l'option `-v` pour voir les instructions en cours d'exécution.

Un test n'est pas une preuve ! Gardez à l'esprit que le jeu de tests fourni est une condition nécessaire mais pas suffisante pour garantir la validité de votre implantation.

3.2 Programmation dans le langage de la machine virtuelle

Pour terminer, vous devez écrire les programmes suivants en code-octet. Ces programmes doivent être sauvegardés dans les fichiers correspondants du répertoire `programs`. Vous pouvez utiliser l'assembleur fourni dans le répertoire `asm`. Le programme `asm.byte` s'utilise en mode flux, c'est-à-dire à l'aide d'une commande de la forme `asm.byte < input_file > output_file`.

La grammaire d'entrée de cet outil est :

::= +

::= L : $\mathbb{N}$
| CONST $\mathbb{N}$
| CONSTINT $\mathbb{N}$
| PUSH
| PUSHINT
| ADDINT
| SUBINT
| DIVINT
| MULINT
| MODINT
| EXIT
| ACC $\mathbb{N}$
| ACCINT
| POP $\mathbb{N}$
| ASSIGN $\mathbb{N}$
| BRANCH L $\mathbb{N}$
| BRANCHIF L $\mathbb{N}$
| GETCHAR
| PRINTCHAR
| ACCPC
| BRANCHACC

Par ailleurs, on ne peut avoir qu'une seule étiquette « i » par ligne.

Enfin, les commentaires sont insérés à l'aide du caractère « ; », placé en début ou en fin de ligne. Dans les exercices qui suivent, il est **très** recommandé de **commenter** votre code au fur et à mesure pour comprendre ce que vous faites. Ces programmes doivent être enregistrés dans le répertoire `programs`.

Affichage du code ASCII d'un caractère Écrire un programme dans le langage qui :
- lit un caractère 'c' sur l'entrée standard ;
- affiche le code ASCII de 'c' en décimal sur la sortie standard.

Simulation d'un appel de fonction Écrire un programme formé de deux parties indépendantes. La première partie est le code d'une **fonction** qui attend un octet en argument et affiche son code ASCII sur la sortie standard. La seconde partie de ce programme appelle cette fonction une première fois sur l'octet 42 et une seconde fois sur l'octet 21.

Somme des carrés Écrire un programme qui :
- lit un caractère 'c' sur l'entrée standard ;
- calcule valant 'c' - '0' ;
- vérifie que est compris entre 0 et 5 ;
- si ce n'est pas le cas, le programme s'arrête ;
- si c'est le cas, il calcule et affiche la somme de '0' et de l'octet de poids faible de $\sum_{k=1}^{y}$ ^{2} (à l'aide de la fonction de l'exercice précédent).

4 Instructions de rendu

Vous **devez** envoyer un mail contenant une archive de votre travail nommée « `vm-nom-prenom.tar.gz` » aux adresses « `yrg@pps.jussieu.fr` » et « `alexandre.pilkiewicz@inria.fr` » avec pour sujet :

`L3 Projet du cours de machines virtuelles`

Cette archive devra se décompresser en un répertoire nommé « `vm-nom-prenom` ». De plus, un fichier AUTEUR contiendra votre nom, votre prénom et votre numéro d'étudiant.

La date limite de rendu est le :

mardi 6 avril 2010 à 16h59

Comme indiqué plus haut, votre projet **doit** compiler *via* la commande `make` effectuée à la racine du projet. Enfin, vos programmes seront relus : veillez donc à ce qu'ils soient **lisibles** (c'est-à-dire indentés, commentés, modularisés...) Tout plagiat aura pour conséquence une note de 0.