

Traduction : La machine virtuelle

—2007-2008—

La machine virtuelle que nous utiliserons est disponible à l'adresse suivante :

<http://igm.univ-mlv.fr/~lombardy/ens/Traduction/VM/vm.tgz>

Il s'agit d'une machine abstraite à pile. 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.
- un registre **counter** qui pointe sur un emplacement particulier du segment de code.

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 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.

1 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 :

NEG	:	$\text{reg1} \leftarrow -\text{reg1}$
ADD	:	$\text{reg1} \leftarrow \text{reg1} + \text{reg2}$
SUB	:	$\text{reg1} \leftarrow \text{reg1} - \text{reg2}$
MULT	:	$\text{reg1} \leftarrow \text{reg1} * \text{reg2}$
DIV	:	$\text{reg1} \leftarrow \text{reg1} / \text{reg2}$
MOD	:	$\text{reg1} \leftarrow \text{reg1} \% \text{reg2}$
EQUAL	:	$\text{reg1} \leftarrow \text{reg1} = \text{reg2}$
NOTEQ	:	$\text{reg1} \leftarrow \text{reg1} \neq \text{reg2}$
LOW	:	$\text{reg1} \leftarrow \text{reg1} < \text{reg2}$
LEQ	:	$\text{reg1} \leftarrow \text{reg1} \leq \text{reg2}$
GREAT	:	$\text{reg1} \leftarrow \text{reg1} > \text{reg2}$
GEQ	:	$\text{reg1} \leftarrow \text{reg1} \geq \text{reg2}$
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 une valeur et la stocke en reg1
WRITE	:	Affiche la valeur stockée en reg1
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 reg2+base de la pile

Les commandes avec un argument sont :

SET	<i>n</i>	:	$\text{reg1} \leftarrow n$
LABEL	<i>n</i>	:	Déclare le label numéro <i>n</i>
JUMP	<i>n</i>	:	Effectue un branchement à l'emplacement du label <i>n</i> du segment de code
JUMPF	<i>n</i>	:	Effectue un branchement à l'emplacement du label <i>n</i> du segment de code si reg1 vaut 0
ALLOC	<i>n</i>	:	Alloue <i>n</i> emplacements supplémentaires en tête de pile
FREE	<i>n</i>	:	Libère <i>n</i> emplacements en tête de pile
CALL	<i>n</i>	:	Sauvegarde l'état de la machine dans la pile et effectue un branchement à l'adresse <i>n</i> du segment de code

Après chaque instruction, on passe à l'instruction suivante (incrémentement du registre **counter**), exceptée 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**. A 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 registres ne sont pas automatiquement sauvegardés lors de l'appel d'une fonction.

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

2 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**.

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
	LOW		#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
	LOW		#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 70
	LABEL	0	
	LABEL	1	
	SET	12	#reg1 := 12
	PUSH		#push 12
	SET	5	#reg1 := 5
	SWAP		#reg2 := 5
	POP		#reg1 := 12
	LOW		#reg1 := 12<5
	JUMPF	42	#goto 2 si reg1=0
	SET	12	
	WRITE		#---affichage 12
	LABEL	2	
	HALT		