Description de la machine à registres

Les registres

La machine contient 8 registres (eax, ebx, ecx, edx, epb, esp, eflags, eip) :

- 6 registres généraux eax, ebx, ecx, edx, epb,esp. Remarque : ebp et esp servent à gérer la pile. esp est le sommet de pile et ebp sert de base.
- 1 registre d'état eflags avec 2 drapeaux ZF et LT (ZF est l'indicateur de valeur nulle et LT de valeur inférieure à 0).
- 1 pointeur d'instruction eip.

Les accès aux registres ou à la mémoires mettent le registre d'état, eflags, à jour.

La boucle d'interprétation est en gros la suivante :

```
top :
      charger l'instruction placée à l'adresses eip
      incrémenter eip
      exécuter l'instruction
      mettre à jour eflags
      goto top
```

Les instructions :

Les instructions ont de zéro à deux paramètres et ont la forme suivante :

- op dest,source
- op dest
- op

Les instructions comprennent :

- quatre opérateurs numériques sur les entiers : add, sub, mul, div.
- deux opérateurs de copie : mov, lea.

quatre opérateurs de manipulation de la pile : push, pop, enter, leave

- deux opérateurs d'entrées sorties : in, out
- six opérateurs de saut conditionnels : jz, jnz, jg, jge, jl, jle
- un opérateur de saut inconditionnel : jmp
- un opérateur pour appeler une routine et un opérateur pour le retour : call, ret

Les opérandes

Les opérandes peuvent suivant les cas correspondre

- à une valeur entiere décimale,
- à un registre,
- à un accès mémoire :
    - un nom de variable (compteur, i, ...) ;
    - une indirection ([reg]) ;
    - une indirection associé à un déplacement (entier[reg], variable[reg], [reg][reg]).

Exemples d'opérandes :

- 7 correspond à la valeur +7
- -12 correspond à la valeur -12
- eax correspond au contenu du registre eax

- compteur correspond à la valeur de la mémoire associé à l'identificateur compteur
- [eax] correspond au contenu de la mémoire dont l'adresse est dans eax
- -8[eax] correspond au contenu de la mémoire dont l'adresse est égale à (eax - 8)
- tab[eax] correspond au contenu de la mémoire dont l'adresse est égale à (adresse de tab + eax)
- [eax][ebx] correspond au contenu de la mémoire dont l'adresse est égale à (eax + ebx)

Tableau de synthèse :

instruction	destination	source	signification	exemple
nop			no operation : instruction vide	nop
add	reg	opérande	reg = reg + opérande	add eax, compteur
sub	reg	opérande	reg = reg - opérande	sub ebx, [esp]
mul	reg	opérande	reg = reg * opérande	mul eax, tab[ebx]
div	reg	opérande	reg = reg / opérande	div eax, +8[esp]
mov	reg	opérande	reg = opérande	mov eax, +45
	mémoire	reg	mémoire = reg	mov [ebx][ecx], eax
lea	reg	mémoire	reg = adresse mémoire	lea eax, compteur
	reg	étiquette	reg = adresse étiquette	lea eax, pgcd
push	opérande		esp = esp - 4 [esp] = opérande	push compteur push eax
pop	mémoire		mémoire = [esp] esp = esp + 4	pop eax

enter	entier		push ebp
			mov ebp, esp
			sub esp, n
				enter 24

leave			mov esp, ebp pop ebp	leave
in	reg		reg = entrée	in eax
out	reg		sortie = reg	out edx
jz	étiquette		si ZF=1 alors saut vers étiquette	jz sortie
jnz	étiquette		si ZF=0 alors saut vers étiquette	jnz L1
jg	étiquette		si LT=0 et ZF=0 alors saut vers étiquette	
jge	étiquette		si LT=0 alors saut vers étiquette	
jl	étiquette		si LT=1 alors saut vers étiquette	
jle	étiquette		si LT=1 ou ZF=1 alors saut vers étiquette	
jmp	étiquette		eip = etiquette (saut vers étiquette)	jmp fin
	reg		eip = reg (saut vers le contenu du registre)	jmp eax
call	étiquette		push(eip) eip = etiquette (saut vers étiquette)	call quicksort
	reg		push(eip) eip = reg (saut vers le contenu du registre)	call eax
ret			eip = pop()	ret

Code assembleur :

Le code machine est séparé en 2 parties :

- une partie pour les données (DATA SEGMENT) avec la déclaration des données
- une partie pour le code (CODE SEGMENT) avec les instructions

Exemple :

Le programme suivant (calcul de pgcd)

```
let a = input;
let b = input;
while (0 < b)
do (let aux=(a mod b); let a=b; let b=aux );
a
.
```

peut correspondre au code suivant :

```
DATA SEGMENT
   b DD
   a DD
   aux DD
DATA ENDS
CODE SEGMENT
        in eax
        mov a, eax
        in eax
        mov b, eax
   debut_while_1:
        mov eax, 0
        push eax
        mov eax, b
        pop ebx
        sub eax,ebx
        jle faux_gt_1
        mov eax,1
        jmp sortie_gt_1
   faux_gt_1:
        mov eax,0
   sortie_gt_1:
        jz sortie_while_1
        mov eax, b
        push eax
        mov eax, a
        pop ebx
        mov ecx,eax
        div ecx,ebx
        mul ecx,ebx
        sub eax,ecx
        mov aux, eax
        mov eax, b
        mov a, eax
        mov eax, aux
        mov b, eax
        jmp debut_while_1
   sortie_while_1:
        mov eax, a
CODE ENDS
```

Exemples de génération de code possible

exp1 + exp2 :
generer(exp1)
generer(exp2)
pop ebx
pop eax
add eax, ebx
push eax

let x = exp
generer(exp)
pop eax
mov x, eax
push eax

exp1 ; exp2
generer(exp1)
pop eax
generer(exp2)

exp1 < exp2
generer(exp1 - exp2)
pop eax
jl vrai
push 0
jmp fin
vrai : push 1
fin :