Annexe - Gestion des piles et tas

Introduction

Le segment de pile assure la gestion des données temporaires, en les empilant les unes à la suite des autres. On y stocke, par exemple lors de l'appel d'une fonction, les paramètres, les variables et les résultats. Ces données sont supprimés de l'espace mémoire dès que l'on quitte la fonction. Contrairement à une pile, un tas permet de maintenir en mémoire les données après l'appel d'une fonction. Un tas est nécessaire par exemple lors de l'initialisation et le remplissage d'un tableau. En python, la taille d'un tableau n'est pas défini à sa création. Encore une fois, c'est le ramasse-miettes qui s'occupe de la tâche difficile de libérer de l'espace. Cette étape reste invisible pour l'utilisateur.

I Gestion du segment de pile

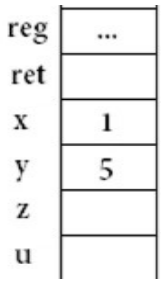
Pour illustrer de fonctionnement d'une pile, prenons en exemple les deux fonctions ci-dessous décrites en python.

```
def g(x, y):
        return 100 + y
def f(x, y):
        z = x + y
        u = g(x-1, y * 2)
```

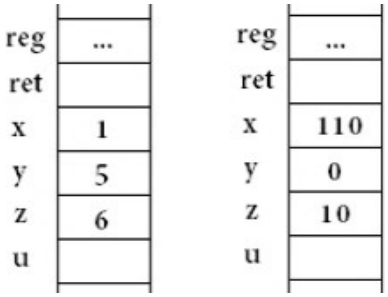
A l'appel de la fonction f(1, 5), une première pile s'initialise comme ci-dessous. On y retrouve nos deux espaces mémoires pour x et y, ainsi que deux autres espaces :

- *reg* pour *registres* qui contient une sauvegarde de l'état du programme.
- *ret* pour *return* qui recevra la valeur à retourner.

La première illustration représente l'état de la mémoire à l'appel de la fonction *f(1, 5)*. Les variables associées à x et à y sont stockés dans les espaces mémoires. On note aussi que les espaces mémoires pour z et u ont été alloués.



Sur cette deuxième illustration, on observe l'organisation de la mémoire lorsque *f(1, 5)* doit appeler la fonction *g(x, y)*. Un nouvel espace mémoire est alors créé (sur la droite de l'image), d'après les arguments passés à la fonction lors de son appel.



Finalement, la pile associé à l'appel de la fonction *g(x, y)* se termine et retourne la valeur 110 qui est stocké dans la variable u, dans la pile de *f(1, 5)*. La valeur finalement retourné est 116.

reg	...
ret	116
x	1
y	5
z	6
u	110

Exercice

Exercice 1

D'après les fonctions définies ci-dessous, décrire les différents états de la pile lorsque :

- L'appel de *g(4)*
- Lorsque *f(x)* s'apprête à renvoyer sa valeur
- Lorsque *g(4)* s'apprête à renvoyer son résultat

```
def f(x):
        z = x * x
        return z - x
def g(y):
        x = y + 1
        t = f(x)
        return x + y + x
```

Exercice

Exercice 2

Décrire le fonctionnement du programme en assembleur ci-dessous :

```
        mov eax, 0
        mov ecx, 100
ici:
        cmp ecx, 0
        je la
        add eax, ecx
        sub ecx, 1
        jmp ici
la:
```

Exercice

Exercice 3

Traduire e, langage assembleur le programme ci-dessous :

```
        x = y + 42
        if x == y:
                z = 1
        else:
                z = 2
```