

Dans cette séance, nous allons voir diverses utilisations de la pile. En particulier nous verrons comment réaliser des fonctions en assembleur.

Rappelons que la pile désigne une zone mémoire qui se trouve en dessous de l'adresse `0xBFFF FFFF`. En mode protégé 32 bits, on stocke uniquement dans la pile des double-mots. Au début du programme, l'adresse du haut de la pile est contenue dans le registre `esp`. La pile grandie vers le bas: plus on rajoute d'éléments dans la pile plus la valeur de `esp` diminue.

1 Utilisation de la pile

Le but de cette section est de se familiariser avec l'utilisation de la pile et en particulier avec les instructions `push` et `pop`.

Question 1 Donnez des suites d'instructions n'utilisant que `mov`, `add` et `sub` correspondant aux instructions `push eax` et `pop ebx`.

Question 2 Pour les instructions ci-dessous, donnez l'état de la pile et des registres `eax`, `ebx` et `esp` avant l'exécution de la ligne `un`, `deux`, `trois` et `quatre`. On supposera que `esp` vaut initialement `0xBFFF0000`. On donnera la pile comme un tableau de double-mots.

```
        mov eax,0
        mov ebx,0
        push dword 12
        push dword 13
        push dword 15
un :    pop eax
        pop ebx
        add eax,ebx
deux:   push eax
        push ebx
        mov dword [esp+8],9
trois:  pop eax
        pop ebx
        pop ebx
quatre:
```

Question 3 Écrivez un programme qui lit des entiers au clavier tant qu'ils sont différents de `-1` et affiche la liste de nombre lu dans l'ordre inverse de leur lecture. Par exemple, si vous entrez:

```
2
5
6
-1
```

Le programme affichera `6 5 2`.

Astuce Utilisez la pile pour stocker les entiers lus.

Bonus 1 Réalisez un programme qui lit une suite d'entiers terminée par `-1` et qui affiche cette liste triée dans l'ordre croissant.

2 Appel de fonction

Le but de cette section est de voir comment sont gérés les appels de fonctions en assembleur.

Rappel: L'instruction `call label` empile l'adresse de l'instruction suivante sur la pile et fait un saut vers `label`. L'instruction `ret` dépile l'adresse stockée en haut de la pile et effectue un saut à cette adresse.

Question 4 Que fait la fonction `swap` dans le programme `fun.asm` ? Quelle est la valeur des registres `eax`, `ebx`, `ecx` et `edx` avant l'instruction `popa` ?

Bonus 2 Donnez la valeur en haut de pile après l'exécution de l'instruction `call swap`.

Question 5 Expliquez le comportement du programme `failure.asm`.

Bonus 3 Quel est le comportement du programme `failure2.asm` ? Supprimez les instructions `call print_int` et `call print_nl`. Expliquez le comportement du programme et l'influence de la suppression de ces deux lignes.

Toutes les fonctions prennent leurs arguments par la pile et doivent respecter les conventions d'appel du C.

Question 6 Réalisez une fonction qui affiche la chaîne (terminée par un octet de valeur 0) et dont l'adresse du premier octet est passée en premier argument. Vous utiliserez l'appel système `write` vu au premier TP.

Question 7 Modifiez la fonction précédente pour que la valeur des registres généraux reste inchangée avant et après l'appel à votre fonction.

Question 8 Réalisez une fonction qui calcule de manière récursive la somme des n premiers entiers. Le résultat sera renvoyé dans le registre `eax`.

Question 9 Réalisez une fonction qui renverse de manière récursive une chaîne de caractères dont l'adresse est passée en premier argument.

Question 10 Réalisez une fonction qui calcule de manière récursive le n -ième terme F_n de la suite de Fibonacci. Rappelons que cette suite est définie par $F_0 = 0$, $F_1 = 1$ et pour $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$. Le résultat sera renvoyé dans le `eax`.