Architecture des ordinateurs

Fiche n°4

ESIPE - INFO 1 2024 –2025

Piles et fonctions

Cette fiche est à faire en une séance (soit 2 h, sans compter le temps de travail personnel), et en binôme. Il faudra

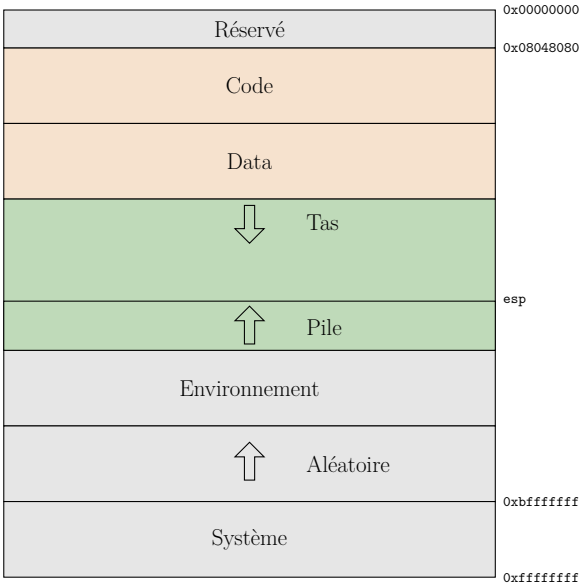
1. réaliser un rapport soigné à rendre **au format pdf** contenant les réponses aux questions de cette fiche (exercices marqués par ■), une introduction et une conclusion ;
2. écrire les différents fichiers sources des programmes demandés. Veiller à nommer correctement les fichiers sources. Ceux-ci doivent **impérativement** être des fichiers compilables par Nasm ;
3. réaliser une archive **au format zip** contenant les fichiers des programmes et le rapport. Le nom de l'archive doit être sous la forme `AO_TP4_NOM1_NOM2.zip` où `NOM1` et `NOM2` sont respectivement les noms de famille des deux binômes dans l'ordre alphabétique ;
4. déposer l'archive sur la plate-forme de rendu.

Tous les fichiers complémentaires nécessaires à ce TP se trouvent sur le site
https://igm.univ-mlv.fr/~derycke/AO/

L'objectif de cette séance est d'apprendre à utiliser la pile et à écrire des fonctions en assembleur. En particulier, nous verrons comment réaliser des fonctions qui suivent les conventions d'appel du C.

1 La pile

La pile désigne une zone de la mémoire qui se trouve avant l'adresse `0xBFFFFFFF` (voir la figure suivante).



Organisation de la mémoire sous Linux en mode protégé.

Par convention, la pile sera utilisée pour stocker exclusivement des `dword` (4 octets). L'adresse du dernier `dword` est enregistrée dans le registre `esp`. Cette zone est appelée *tête de pile*.

Important 1 Plus on ajoute d'éléments dans la pile plus la valeur de `esp` diminue.

On dispose de deux instructions pour faciliter la lecture et l'écriture dans la pile : `push` et `pop`. L'instruction `push eax` ajoute le `dword` contenu dans `eax` en tête de la pile (et modifie donc `esp` en conséquence). L'instruction `pop eax` enlève le `dword` se trouvant en tête de la pile et le copie dans le registre `eax` (et modifie donc `esp` en conséquence).

Important 2 Dans les question qui suivent — sauf mention contraire — on suppose que la pile est initialement vide et que la valeur `esp` est x (où x est une valeur quelconque et inconnue). Lorsque cela est demandé, l'état de la pile doit être donné sous la forme suivante :

Table 1: Exemple de représentation de la pile

Adresse	Valeur
⋮	⋮
$x-8$	`val_1`
$x-4$	`val_2`
x	`val_3`
$x+4$	`val_4`
⋮	⋮

Nous disons alors que ce tableau représente l'état de la pile *relativement à* la valeur x.

Exercice 1 ■ Pour la suite d'instructions ci-contre, donner instruction par instruction les valeurs des registres `esp` et `eax` ainsi que l'état de la pile.

```
mov eax, 0xABCDEF01
push eax
mov eax, 0x01234567
push eax
pop eax
pop eax
```

Exercice 2 ■ Donner des suites d'instructions utilisant uniquement les instructions `mov` et `add` afin de simuler l'instruction.

```
push eax
```

Faire la même chose pour l'instruction.

```
pop ebx
```

Exercice 3 Dans le code ci-contre, le registre `ebp` est utilisé pour sauvegarder la valeur de `esp`. Nous verrons plus tard que c'est en fait le rôle traditionnel de `ebp`.

Donner l'état des registres `eax`, `ebx`, `esp` et de la pile aux étiquettes `un`, `deux`, `trois` et `quatre`. L'état de la pile doit être donné relativement à la valeur de `ebp`.

```
    mov ebp, esp
    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 :
```

Exercice 4 ■ Écrire un programme `E4.asm` qui lit des entiers au clavier tant qu'ils sont différents de `-1`, et affiche la liste des entiers lus dans l'ordre inverse. Par exemple, sur l'entrée

`2 5 6 7 1 3 2 4 -1`,

le programme affichera

`4 2 3 1 7 6 5 2`.

Astuce 1 Il est possible (et recommandé) d'utiliser la pile pour enregistrer les entiers lus.

Exercice 5 Réaliser un programme `E5.asm` qui lit une suite d'entiers terminée par `-1` et qui affiche cette liste triée dans l'ordre croissant. Un simple tri à bulles ou un tri par sélection suffira.

---

2 Appels de fonction

Nous allons introduire le mécanisme de base pour écrire des fonctions en assembleur. Les deux instructions que l'on utilise sont `call` et `ret`. Voici leur description :

1. l'instruction `call label` empile l'adresse de l'instruction suivante sur la pile et saute à l'adresse `label`. Le fait de se souvenir de cette adresse permet de reprendre l'exécution des instructions après que les instructions correspondant au saut aient été exécutées.
2. L'instruction `ret` dépile le `dword` de la tête de pile et effectue un saut à cette adresse. C'est l'adresse empilée précédemment par l'instruction `call`.

Exercice 6 ■ Pour le code ci-contre, donner une suite d'instructions n'utilisant que `push` et `jmp` qui soit équivalente à l'instruction `call` dans l'instruction `call print_int` de la ligne 1.

```
    call print_int
suite :
    mov eax, 0
    ...

print_int :
    ...
```

Exercice 7 Donner l'ordre dans lequel sont exécutées les instructions ci-contre.

Pour cela, mentionner pour chaque étape de l'exécution, le numéro de la ligne exécutée ainsi que l'état de la pile si celle-ci a été modifiée par l'instruction en question.

```
main :
    call f
l1 :
    call g
l2 :
    mov ebx, 0
    mov eax, 1
```

```
    int 0x80
f :
    call g
l3 :
    ret
g :
    ret
```

Exercice 8 Expliquer si la suite d'instructions ci-contre est correcte.
Dans le cas contraire, expliquer pourquoi.

```
fonction :
    call read_int
    push eax
    pop ebx
    push ecx
    ret
```

Exercice 9 Expliquer le comportement des programmes `Probleme1.asm` et `Probleme2.asm`. On y trouve des instructions que nous n'avons pas rencontrées auparavant. Voici leur explication :

- `pusha` empile le contenu de tous les registres, c'est à dire `eax`, `ebx`, `ecx`, `edx`, `esp`, `ebp`, `esi` et `edi`. Cette instruction est utile pour effectuer une sauvegarde générale des valeurs des registres avant de les modifier ;
- `popa` restaure les valeurs de tous les registres dont les valeurs ont été enregistrées au préalable dans la pile par l'intermédiaire de `pusha` ;

Exercice 10 ■ Réécrire la fonction `print_string` en utilisant l'appel système `write`. Faire en sorte qu'après l'exécution de la fonction, les registres soient dans le même état qu'avant l'exécution de la fonction. La fonction prend en argument une adresse d'une chaîne de caractères qui se termine par le caractère nul (de code ASCII `0`, encore appelé marqueur de fin de chaîne). La fonction se nommera `print_string2` et sera écrite dans un fichier `E10.asm`.

---

3 Convention d'appel du C

La convention d'appel du C est essentiellement la suivante :

- les paramètres de la fonction sont passés par la pile et non dans les registres comme pour les fonctions `print_int` et `print_string`
- au début de la fonction, on empile la valeur de `ebp`, et on sauvegarde la valeur `esp` dans `ebp`. À la fin de la fonction, la valeur de `ebp` est restaurée
- l'éventuelle valeur de retour de la fonction doit être placée dans `eax`

- les valeurs des registres `eax`, `ecx` et `edx` peuvent être modifiées à l'envie. Les valeurs des autres registres doivent être les mêmes avant et après l'appel à la fonction. (on parle de registres *caller-save* et *callee-save*)

Un squelette possible de fonction est le suivant :

```
fonction :
    push ebp      ; Valeur de ebp empilee.
    mov ebp, esp ; Sauvegarde la tete de pile dans ebp.

    ; Debut du corps de la fonction.
    ; argument 1 @ ebp + 8
    ; argument 2 @ ebp + 12
    ; argument k @ ebp + 4 * (k + 1)
    ; Fin du corps de la fonction.

    pop ebp      ; Valeur de ebp restauree.
    ret
```

Pour appeler cette fonction, on utilise une suite d'instructions de la forme :

```
push arg3       ; On empile les arguments dans l'ordre inverse.
push arg2
push arg1
call fonction   ; Appel de la fonction.
add esp, 12     ; Depile d'un coup les trois arguments.
```

Au début de l'exécution du corps de la fonction, l'état de la pile, relativement à `ebp`, est

Table 2: État de la pile

Adresse	Valeur
⋮	⋮
`ebp`	ancienne valeur de `ebp`
`ebp` + 4	adresse retour
`ebp` + 8	argument 1
`ebp` + 12	argument 2
`ebp` + 16	argument 3
⋮	⋮

Exercice 11 ■ Écrire une fonction `longueur` qui respecte les conventions d'appel du C et qui prend en argument l'adresse d'une chaîne de caractères terminée par un octet 0 et calcule sa longueur.

Exercice 12 ■ Réécrire la fonction `print_string2` (dans un nouveau fichier) en suivantla convention d'appel du C. La fonction devra utiliser la fonction `longueur` précédente.
