TP 2 : Programmation en assembleur (2012 – version 4)

Les programmes de ce TP sont à écrire en assembleur x86, en syntaxe Intel, sans faire appel à la librairie standard du C. Les programmes et fonctions sont à tester au fur et à mesure.

Exercice 1 – *Premier programme*

1. Recopiez le programme ci-dessous dans un fichier appelé `bonjour.asm`.

```
section .data
message: db 'Bonjour chez vous !',10
longueur: equ $-message
section .text
global _start
_start:
  mov eax, 4
  mov ebx, 1
  mov ecx, message
  mov edx, longueur
  int 80h
  mov eax, 1
  mov ebx, 0
  int 80h
```

2. Compilez-le avec les commandes :

- `nasm -f elf bonjour.asm`
- `ld -m elf_i386 -o bonjour.elf bonjour.o`

puis exécutez le avec la commande :

- `./bonjour.elf`

et vérifiez qu'il fonctionne correctement. Que fait ce programme ? A quoi sert la commande `ld` ?

3. Comment modifieriez-vous la deuxime ligne de code pour que le programme affiche "Bonjour" puis "chez vous" sur deux lignes successives ?

Exercice 2 – *Appels systèmes*

Les appels systèmes sont des fonctions que le système d'exploitation met à disposition des applications et que les programmes assembleur peuvent utiliser. Sous Linux comme sous MSDos, ces appels systèmes utilisent des interruptions. Une interruption est caractérisée par un numéro d'interruption associé à un numéro d'appel qui est placé dans le registre `EAX` *avant le lancement de l'interruption ; les autres paramètres utiles sont placés dans les autres registres* `EBX`*,* `ECX`*, etc.*

1. Quelles sont les interruptions et numéros d'appel utilisés dans le programme `bonjour` de l'exercice précédent ?

2. Cherchez la liste des appels systèmes de Linux, en tapant la commande `man syscalls`, ou en faisant une recherche sur Internet avec les mots-clés `syscalls Linux`. A quoi correspond l'appel système 0x80 ?
A quoi correspondent ses numéros d'appel 1 et 4 utilisé dans le programme `Bonjour` ?

3. Modifiez le programme `Bonjour.asm` en un programme `Hithere.asm` qui effectue un deuxième appel système afin d'afficher "Bonjour chez vous" puis "Hi there !" sur deux lignes successives. Combien de fois votre programme utilise-t-il la fonction `INT` ?

4. (*plus difficile*) Il est possible de trouver la correspondance entre un numéro d'appel système et la fonction qui fait cet appel système en consultant les fichiers d'entête du C (c'est-à-dire, les fichiers `.h`). Par exemple, quel appel système permettrait de créer un répertoire (`mkdir`) ?
N.B. *En recherchant dans les différents fichiers* `.h` *du système, on peut trouvez cette correspondance. Le mieux est de partir du fichier* `syscall.h` *et de suivre les inclusions de* `.h` *; il est aussi possible que les appels systèmes se trouvent dans un fichier* `unistd.h` *(souvent inclus dans* `syscall.h`*). Pour savoir si un fichier contient les appels systèmes, on peut faire une recherche du mot-clef* `SYSCALL` *(souvent en majuscules).* Gardez une trace de votre recherche.

5. A l'aide des informations précédemment obtenues, Ecrivez un programme en assembleur qui affiche à l'écran "Je pense donc j'ecris" puis quitte, en utilisant les appels systèmes `write` et `exit`.

6. Sur le même principe, écrivez un programme en assembleur qui lit un caractère au clavier, l'affiche, puis quitte.

Exercice 3 – *Avec des boucles...*

1. Écrivez un programme en assembleur qui affiche une ligne de n étoiles, n étant une constante de votre programme. Modifiez ce programme pour que n soit un chiffre saisi au clavier.

2. Écrivez un programme en assembleur qui affiche un carré de $n \times n$ étoiles. n est une constante de votre programme.

3. Écrivez un programme qui lit un par un (avec l'appel système `read`) des caractères, jusqu'à trouver le caractère espace (code ASCII 32) , et les affiche à mesure sur la sortie standard. Que se passe-t-il à l'exécution si vous donnez une chaîne de plusieurs caractères ? une chaîne de plusieurs caractères terminée par un espace ?

4. Écrivez une fonction assembleur `Ecrirenombre` qui prend en paramètre un nombre entier (présent dans le registre EAX par exemple), et l'affiche.

5. Écrivez une fonction assembleur `Palindromeur` qui prend en paramètre un nombre entier et le considère comme un mot pour l'afficher en ordre inverse (i.e. de droite à gauche).

6. Écrivez une fonction assembleur `Lirenombre` qui :

a. Lit au clavier des caractères un par un et s'arrête dès que le caractère saisi n'est plus un chiffre ;
b. Interprète cette suite de caractères comme un nombre entier (à placer dans EAX par exemple) ;
c. Affiche ce nombre interprété.

Comment ferez-vous pour que le nombre saisi ne dépasse pas la capacité du registre ?

Exercice 4 – *Calculs rapides avec des entiers*

1. Ecrivez un programme qui détermine une approximation entière de la racine carrée d'un nombre n en énumérant tous les entiers x en partant de 1, jusqu'à en trouver un tel que $x^2 > n$. L'approximation entire par défaut est alors $x - 1$.

2. Pour déterminer si un nombre entier n est premier, on peut tenter de le diviser par tous les entiers entre 2 et $n - 1$; si l'une de ces divisions euclidiennes (entières) donne un reste nul, alors le nombre n'est pas premier.
a. Ecrivez un programme `Premier` qui détermine si un nombre n, lu au clavier, est premier ou non.
b. Modifiez ce programme pour qu'il affiche le premier facteur de ce nombre s'il en existe ou sinon le message "nombre premier".

Exercice 5 – *Format IEEE 754*

1. Ecrivez un programme qui indique si un nombre, codé au format IEEE 754 simple précision, est positif ou négatif.

2. Ecrivez un programme qui calcule l'exposant (non biaisé) d'un nombre écrit au format IEEE 754 simple précision. Si cet exposant est noté e, on pourra en déduire que la valeur absolue du nombre appartient à l'intervalle $[2^e; 2^{e+1}[$.

Exercice 6 – *Fonctions récursives*

1. Programmez la fonction factorielle en assembleur en utilisant une fonction récursive.

2. Programmez la suite de Fibonacci en assembleur en utilisant la définition récursive suivante : $F(0) = 1$, $F(1) = 1$ et pour tout $n \geq 2$: $F(n) = F(n-1) + F(n-2)$.

Exercice 7 ***Racines d'un polynôme du deuxième degré***
Dans cette partie, nous allons utiliser le coprocesseur arithmétique, une unité de calcul indépendante du processeur spécialisée dans les calculs (notamment, à virgule flottante). Avant d'aborder cet exercice, il est recommandé de se documenter sur :

- *les registres du coprocesseur arithmétique :* `st0`*,* `st1`*, ...,* `st7`*,*
- *les instructions du coprocesseur :* `fild` *pour charger une valeur entière dans un registre du coprocesseur,* `fist` *pour transférer une valeur entière d'un registre du coprocesseur en mémoire,* `fadd` *pour ajouter,* `fsub` *pour soustraire,* `fmul` *pour multiplier,* `fdiv` *pour diviser,* `fsqrt` *pour calculer la racine carrée, et* `frndint` *pour arrondir à l'entier le plus proche.*

Ecrivez un programme qui lit trois entiers a, b et c, et calcule une approximation entière des racines du polynôme $a.x^2 + b.x + c$. a, b et c seront lus comme des entiers, en utilisant `fild`) ; les racines seront affichées sous la forme de l'entier le plus proche du résultat exact, en utilisant `frndint` et `fist`).