Architecture des Ordinateurs

Licence 3

TP n°3 : Assembleur

Tous vos programmes réalisés en assembleur doivent impérativement être présentés comme suit :

- Les commentaires et mnémoniques doivent être aligné(e)s à la même colonne au sein du code
- Chaque sous-programme doit être précédé d'un en-tête décrivant sa fonction et ses paramètres d'entrée/sortie.

Dans ce TP, nous utiliserons les programmes suivant :

- **nasm** : le compilateur d'assembleur
<http://nasm.sourceforge.net/doc/html/nasmdoc3.html>
- **ddd** : le programme d'analyse et de mise au point (débugueur)
<http://www.stat.ohio-state.edu/current/support/programming/ddd/>
- **nm**, pour lister les symboles utilisés dans votre programme, **objdump**, pour afficher le contenu des objets binaires obtenus et **ld**, l'éditeur de liens GNU => cf. pages 'man'.
- Appels système du noyau Linux (i386/PC)
<http://www.lxhp.in-berlin.de/lhpsysc0s.html>

Ex. 1 : Récupérez `hello.s` et compilez-le avec la commande : `nasm -f elf -g hello.s`

Liez le fichier avec la commande : `ld hello.o -o hello` et exécutez-le.

Suivez ensuite son exécution avec `ddd` (cf. fin de page pour le fonctionnement de `ddd`).

Analyser les fichiers obtenus avec `objdump -d hello.{o,a}` ainsi que `nm -p hello.{o,a}`

A quoi correspondent les sorties et les paramètres de `objdump` et `nm` ?

Ex. 2 : Changer le point d'entrée standard `__start:` du programme par le symbole `Debut:`.

Observer les changements provoqués avec `ld` et `nm`.

`man ld` pour changer le symbole du point d'entrée du programme.

Ex. 3 : Ecrivez le même programme en langage C, comparez la taille des exécutable obtenus (asm, C).

Auparavant, utilisez `strip fic` pour éliminer les informations superflues des exécutable.

Ex. 4 : Ajouter au programme deux variables `a` et `b`, initialisées, entières sur 32 bits, ainsi qu'une variable de même type `s`, non initialisée (dans une section `.bss`). Ajouter également le calcul dans `s` de la somme de `a` et `b`.

Compiler et exécuter le programme avec `ddd`.

Ex. 5 : Mettre la fonction d'affichage d'une chaîne de caractères sous la forme d'une procédure (cf. mnémoniques `call` et `ret`) prenant en paramètre l'adresse de la chaîne et sa taille, respectivement dans les registres `eax` et `ebx`.

Observer l'état de la pile avec `ddd`. Qu'effectue la mnémonique `call` ?

Ex. 6 : Créer une procédure qui affiche le chiffre correspondant à un nombre (stocké dans `eax`) entre 0 et 9 à l'aide de la chaîne `'0123456789'` et d'un décalage par rapport au début de cette chaîne.

Ex. 7 : Même question, mais sans utiliser la chaîne `'0..9'`, et en passant le nombre sur la pile.

Ex. 8 : Afficher les nombres de 0 à 9, un par ligne, à l'aide d'une boucle.

Ex. 9 : Créer une procédure qui affiche en base 10 un nombre entier 16 bits passé par la pile.

Ex. 10 : Même question mais en spécifiant la base de représentation du nombre.

Quelques commandes DDD

<code>set disassembly-flavor intel</code>	affichage des mnémoniques en représentation ix86
<code>graph display `x /8cb \$ecx`</code>	affiche les 8 bytes à l'adresse <code>ecx</code> sous forme de caractères
<code>graph display `x /3ib \$eip`</code>	affiche les 3 prochaines instructions à exécuter
<code>graph display `x /1sb \$ecx`</code>	affiche la chaîne de caractères stockée à l'adresse <code>ecx</code>
<code>graph display `info registers`</code>	affiche l'état des registres
<code>graph display `info registers \$eax`</code>	affiche uniquement le registre <code>eax</code>