Université Gustave Eiffel

Travaux Dirigés de Compilation n°2

Licence d'informatique

———

Prise en main de l'assembleur

Le but de ce TD est de prendre en main **nasm** 64 bits, un assembleur pour processeurs x86. Vous allez compiler des programmes et faire des opérations arithmétiques sur les registres puis des boucles.
Ces exercices sont conçus pour les machines de l'IGM. Sur d'autres machines, il peut y avoir des différences dans l'accès aux registres (pas d'accès à **rax**, **rbx**...), les conventions d'appel de fonctions, les instructions, les conditions d'utilisation de la bibliothèque fournie, la compilation.

▸ Exercice 1. *Compiler du code en assembleur*

Le fichier `utils.asm` *est un module qui vous permet de faire des affichages. Le fichier* `lazy.asm` *contient le code minimal en assembleur pour créer un exécutable qui ne fait rien mais ne provoque pas d'erreur à l'exécution.*

1. *Créez un fichier objet* `lazy.o` *avec la commande*
   `nasm -f elf64 -o lazy.o lazy.asm`
   et de même pour `utils.asm`*. Créez un exécutable à partir de* `utils.o` *et* `lazy.o` *par la commande*
   `gcc -o lazy lazy.o utils.o -nostartfiles -no-pie`
   Testez. Vérifiez la valeur de retour du programme par la commande `echo $?`*.*
2. *Faites un* `makefile` *pour automatiser les opérations précédentes.*

▸ Exercice 2. *Registres, appels de fonction*

1. *Dans le programme* `registers.asm`*, à chacun des 6 commentaires dans le code, déduisez les valeurs des registres* `rbx`*,* `r12`*,* `r13` *et* `r14`*.*
2. *Ajoutez des appels à la fonction* `show_registers` *et testez pour vérifier vos résultats. La fonction* `show_registers` *qui vous est fournie affiche* `rbx` *et* `r12` *en hexadécimal, et* `r13` *et* `r14` *en décimal. Si vous préférez un autre affichage, vous pouvez modifier la valeur de* `format_registers` *dans le module* `utils.asm` *.*

Dans les exercices qui suivent, utilisez `call_registers` pour afficher les valeurs des registres.

▸ **Exercice 3.** ***Instructions conditionnelles, valeur de retour***

1. *Faites un programme en* `nasm` *qui met les valeurs de votre choix dans* `rbx` *et* `r12` *puis qui renvoie 1 si* `rbx` *est strictement plus grand que* `r12` *et -1 sinon. (La valeur de retour d'un programme est codée sur 1 octet, et Linux l'interprète comme entier signé, ce qui donne 255.)*
2. *Modifiez le programme pour qu'il renvoie 0 si* `rbx` *et* `r12` *sont égaux.*

▸ **Exercice 4.** ***Boucles***

1. *Faites un programme en* `nasm` *qui incrémente* `rbx` *de 1 indéfiniment en utilisant l'instruction* `jmp`*, sans condition d'arrêt.*
2. *Modifiez votre programme pour qu'il s'arrête quand* `rbx` *vaut 100.*
3. *Modifiez votre programme pour que l'incrémentation se fasse par un incrément de 9.*
4. *Modifiez votre programme pour qu'il s'arrête quand* `rbx` *est supérieur ou égal à 100.*

▸ **Exercice 5.** ***Accès à la mémoire***

1. *Dans le programme* `tdc2_memory_access.asm`*, à chaque commentaire dans le code, déduisez les valeurs des registres* `rbx`*,* `r12`*,* `r13` *et* `r14`*, sauf pour le registre qui contient une adresse, imprévisible.*
2. *Ajoutez des appels à la fonction* `show_registers` *et vérifiez vos résultats.*

▸ **Exercice 6.** ***Calcul d'adresses***

1. *Faites un programme qui alloue de la mémoire statique pour une liste d'entiers de votre choix, comme* `seq_numbers` *dans* `tdc2_memory_access.asm`*, tous strictement positifs sauf le dernier qui sera 0. Votre programme doit ensuite mettre successivement chaque entier strictement positif dans* `rbx` *jusqu'à tomber sur 0.*
2. *Ajoutez du code qui parcourt les entiers dans l'ordre des adresses décroissantes, donc à partir du 0, et qui copie les entiers dans* `rbx` *les uns après les autres.*
3. *Faites un programme qui place dans* `r14` *le plus grand entier de la liste.*