Devoir 1
INFO3201 - Architecture d'ordinateur

Remise : Lundi 6 octobre.

Prof. Andy Couturier

Automne 2025

Avertissement : Bien que l'utilisation de dialogueurs (*chatbots*) soit autorisée pour la réalisation de ce devoir, il est important de réfléchir à la manière dont vous les utilisez. Les exercices proposés ici sont de niveau « très facile » et il est trivial pour un dialogueur d'en fournir directement les solutions. Une partie importante de la pondération de l'examen intra 1 sera justement constituée de ce genre d'exercices, lesquels devront être réalisés en environnement contrôlé, sans accès aux dialogueurs. Je vous recommande donc de n'utiliser les dialogueurs qu'en cas de blocage réel, et uniquement en dernier recours. Recourir systématiquement à ces outils risque de vous priver de l'expérience nécessaire à la réussite de l'examen. Pour progresser et vous préparer efficacement, tentez d'abord de résoudre les exercices par vous-même.

Instructions pour le livrable

Le devoir doit être remis via un répertoire Git qui doit être synchronisé sur le serveur git.couturier.prof avant la date et heure de remise.

- Sur le serveur Git, votre répertoire devra être exactement « `INFO3201-A2025-Devoir1` ».
- Les fichiers devront être nommés comme suit :
  - `ex1.asm`
  - `ex2.asm`
  - `ex3.asm`
  - `ex4.asm`
  - `ex5.asm`

  - `ex6.asm`
  - `ex7.asm`
  - `ex8.asm`
  - `ex9.asm`
  - `ex10.asm`
- Chaque code source doit être exclusivement écrit en assembleur pour l'architecture x86_64, en suivant la syntaxe propre à l'assembleur `NASM`. Aucun autre langage de programmation ne sera accepté.
- Chaque code source doit être soigneusement commenté. L'apparence du code doit être propre, avec une bonne aération (lignes vides entre les différentes parties du programme).
- Chaque fichier de code source doit pouvoir être assemblé sans erreur à l'aide de l'assembleur `NASM`, version 2.16.03. Le code source doit également être compilé avec la commande suivante sans erreur fatale :

  ```
  nasm -felf64 <nom_de_votre_fichier_source>
  ```

  Assurez-vous de remplacer `<nom_de_votre_fichier_source>` par le nom de votre fichier source.

Exercices

Exercice 1. Écrire un programme assembleur qui effectue un compte à rebours de 10 à 0 inclus. Chaque valeur doit être affichée sur une ligne différente à l'aide de `printf`, puis, à la fin de la boucle, afficher « Décollage ! » avec `puts`.

Exercice 2. Écrire un programme assembleur qui analyse la valeur d'un entier signé sur 8 bits A stocké en mémoire, et affiche avec `puts` :

- « A est strictement positif » si $A > 0$;
- « A est nul » si $A = 0$;
- « A est strictement négatif » si $A < 0$.

Exercice 3. Écrire un programme assembleur qui demande à l'utilisateur de saisir deux entiers non signés de 8 bits chacun (en indiquant la plage de valeurs valides). Lire les deux valeurs à l'aide de `scanf`. Additionner les deux valeurs puis :

- si la somme dépasse 255, afficher avec `puts` : « Il y a eu un dépassement. » ;
- sinon, afficher la somme en décimal et en hexadécimal (chacune sur une ligne différente) à l'aide de `printf`.

Exercice 4. Écrire un programme assembleur qui affiche le plus petit de trois entiers signés de 8 bits (A, B et C), stockés en mémoire. Afficher ce minimum en décimal à l'aide de `printf`.

Exercice 5. Écrire un programme assembleur qui vérifie si un entier non signé A (stocké en mémoire) appartient à l'intervalle strictement compris entre 10 et 100. Afficher, à l'aide de `puts`, soit « Dans l'intervalle », soit « Hors intervalle » selon le cas.

Exercice 6. Écrire un programme assembleur qui demande à l'utilisateur de saisir un entier signé de 32 bits, en précisant la plage de valeurs valides. Lire la valeur à l'aide de `scanf`. Après la lecture, déterminer si cet entier est positif ou négatif selon chacune des méthodes suivantes :

- en utilisant `TEST` ;
- en utilisant `CMP` ;
- en utilisant `BT` ;
- en utilisant `SHR` ;
- en utilisant `SAR`.

Pour chaque méthode, afficher avec `puts` soit « Positif » soit « Négatif ».

Exercice 7. Écrire un programme assembleur qui demande à l'utilisateur de saisir dix entiers non signés de 8 bits chacun (en indiquant la plage de valeurs valides). Lire chaque valeur avec `scanf` et les stocker dans un tableau en mémoire. Parcourir ensuite le tableau pour déterminer la valeur maximale, la valeur minimale, puis calculer leur différence (`max` − `min`). Afficher à l'aide de `printf` :

- la valeur maximale (en décimal) ;
- la valeur minimale (en décimal) ;
- la différence (en décimal).

Exercice 8. Écrire un programme assembleur qui demande à l'utilisateur de saisir dix entiers non signés de 8 bits chacun (en indiquant la plage de valeurs valides). Lire chaque valeur avec `scanf` et les stocker dans un tableau en mémoire. Utiliser une double boucle pour vérifier si le tableau contient au moins une valeur en double. Afficher, avec `puts`, soit « Il y a des doublons », soit « Tous les éléments sont uniques » selon le cas.

Exercice 9. Écrire un programme assembleur qui additionne deux entiers non signés de 64 bits, chacun stocké dans une variable de 8 octets dans la section `.data`. Utiliser une boucle : à chaque itération, additionner les octets correspondants des deux entiers avec l'instruction `ADC` afin de propager la retenue d'un octet à l'autre. Avant de commencer la boucle, remettre le drapeau `CF` à 0 avec l'instruction `CLC`. À chaque itération, écrire le résultat dans une variable de 8 octets dans la section `.bss`. Afficher ensuite la somme totale en décimal à l'aide de `printf`.

Exercice 10. Écrire un programme assembleur qui lit, un à un, des entiers non signés de 8 bits. L'utilisateur peut entrer autant de valeurs qu'il le souhaite. Après chaque saisie (avec `scanf`), ajouter immédiatement la valeur à une somme cumulée. Le programme s'arrête dès que la somme cumulée dépasse 255, puis affiche, à l'aide de `printf` :

— la somme totale atteinte (en décimal) ;
— le nombre de valeurs qui ont été additionnées avant le dépassement (en décimal).