POLYTECHNIQUE MONTRÉAL

INF1600 - Architecture des micro-ordinateurs

Travail pratique 1

Introduction à l'assembleur et au langage machine
Introduction aux outils des laboratoires
Boucle d'exécution sur Code Machine
Introduction à l'assembleur x86

Département de Génie Informatique et Génie Logiciel
Polytechnique Montréal
Hiver 2026

Introduction

Ce travail a pour but de vous familiariser avec le langage assembleur, le langage machine et la boucle d'exécution d'un ordinateur à travers la résolution de problèmes sur divers processeurs. L'objectif est de comprendre, comment un processeur exécute des instructions élémentaires et comment des constructions de haut niveau se traduisent en séquences simples d'opérations arithmétiques, logiques et de branchements.

Nous profiterons de la première séance de laboratoire pour configurer les outils utilisés durant le cours.

CodeMachine

Une partie de ces exercices devront être résolus avec la plateforme **CodeMachine**. Suivre le guide suivant pour l'installation de CodeMachine et pour la liste d'instructions disponibles sur les processeurs. Assurez-vous d'avoir la **version 26.0.1**.

x86 IA-32

L'autre partie des exercices devront être résolus dans un environnement **Linux** sur une architecture **x86-IA32**. Suivre le guide suivant pour la configuration de votre environnement. Une référence des instructions courantes est disponible ici. Un gabarit pour compléter les exercices sur cette architecture sera fourni.

Modalité de remise

Ce travail pratique est formatif. Aucune remise n'est demandée. Néanmoins, les concepts abordés dans ce travail seront évalués au TP2-quiz. Un corrigé sera fourni quelques jours avant le TP2-quiz.

Instructions

- Les exercices identifiés avec [ACC] doivent être réalisés sur le processeur Accumulateur de CodeMachine.
- Les exercices identifiés avec [MA] doivent être réalisés sur le processeur Accumulateur avec registre MA de CodeMachine.
- Les exercices identifiés avec [x86-IA32] doivent être réalisés dans un environnement Linux sur une architecture x86-IA32.

N-ième nombre de Fibonacci [ACC]

Sujets : Boucles, Branchements

Énoncé

Vous êtes surement familiers avec la séquence de Fibonacci où chaque nombre est la somme des deux nombres qui le précèdent. Celle-ci est définie par les relations suivantes :

- Les deux premiers termes de la suite sont initialisés à 0 et 1 :

$$T(0) = 0, \qquad T(1) = 1$$

- Ensuite, chaque terme suivant est la somme des deux termes précédents :

$$T(n) = T(n-1) + T(n-2)\ pour\ n \geq 2$$

Étant donné un entier `n`, retournez dans `ACC` la valeur de $T(n)$.

Exemple 1 :

Entrée : `n = 4`
Sortie : 3
Explication : La suite de Fibonacci est : `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …`

Exemple 2:

Entrée : `n = 10`
Sortie : 55
Explication : La suite de Fibonacci est : `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …`

Contraintes :

- `0 <= n <= 18`
- Il faut minimiser les variables codées en dur
- Pour obtenir tous les points, il faut que le code fonctionne pour un `n` quelconque.
- Pour obtenir tous les points, il faut que l'exemple s'exécute avec ≤ `750` cycles.

Gabarit de code :

```
.text

stop

.data
n: 4
```

Tester si un nombre est une puissance de 2 [ACC]

Sujets : Boucles, Branchements, Arithmétique des ordinateurs

Énoncé

Étant donné un entier n, retournez `1` dans `ACC` s'il s'agit d'une puissance de deux, ou `0` sinon

Un entier n est une puissance de deux s'il existe un entier k tel que :

$$n = 2^k \; et \; k \geq 0$$

Exemple 1 :

Entrée : `1`
Sortie : `1`
Explication : `1 =` 2^0

Exemple 2 :

Entrée : `16`
Sortie : `1`
Explication : `16 =` 2^4

Exemple 3 :

Entrée : `18`
Sortie : `0`
Explication : 18 n'est pas une puissance de 2.

Contraintes :

- `0 <= n <=` 2^{14}
- Pour obtenir tous les points, il faut que le code fonctionne pour un `n` quelconque.
- Pour obtenir tous les points, il faut que **l'exemple 2** s'exécute avec ≤ `150` cycles.

Gabarit de code :

```
.text
stop

.data
n: 4
```

Est pair? [MA]

Sujets : Boucles, Branchements, Arithmétique des ordinateurs

Énoncé :

Étant donné un entier `n`, retournez `1` dans ACC si n est pair, sinon `0`.

Exemple 1:

Entrée : `n = 4`
Sortie : `1`
Explication : 4 est pair

Exemple 2:

Entrée : `n = 7`
Sortie : 0
Explication : 4 est impair

Contraintes :

- $-10^5 <= n <= 10^5$
- Il faut minimiser les variables codées en dur
- Pour obtenir tous les points, il faut que le code fonctionne pour `n` de taille quelconque.
- Pour obtenir tous les points, il faut que tout exemple s'exécute avec ≤ `25` cycles.

Gabarit de code :

```
.text

stop

.data
n: 4
```

Compter l'occurrence d'un nombre dans un tableau [MA]

Sujets : Boucles, Branchements, Adressage indexé, Tableau

Énoncé :

Étant donnée une liste d'entiers `nums` de taille `n` et un entier `target`, calculez combien de fois `target` apparaît dans `nums`. Retournez ce `nombre` d'occurrences dans `ACC` avant que le programme ne termine.

Exemple:

Entrée : `nums = [1,2,3,2,2,4], target = 2`
Sortie : `3`
Explication : Le nombre 2 apparaît trois fois aux indices 1, 3 et 4.

Contraintes :

- `1 <= n <= 50`

- Il faut minimiser les variables codées en dur
- Pour obtenir tous les points, il faut que le code fonctionne pour un tableau `nums`, de **taille `n` quelconque.**
- Pour obtenir tous les points, il faut que l'exemple s'exécute avec ≤ `275` instructions.

Gabarit de code :

```
.text

stop

.data
n: 6 # nbr elements dans nums

nums: 1   # nums[0]
2   # nums[1]
3   # nums[2]
2   # nums[3]
2   # nums[4]
4   # nums[n - 1]

target: 2 # chiffre à compter
```

Suite de Golomb [MA]

Sujets : Boucles, Branchements, Arithmétique des ordinateurs, Adressage indexé

Énoncé

Dans la suite de Golomb chaque entier apparaît un nombre de fois égal à sa valeur.

Les premiers termes sont : `1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, …`

Étant donné un entier `n`, sauvegarder dans la mémoire les termes d'une suite de Golomb. Retournez dans `ACC` **l'adresse mémoire** où débute la suite.

Exemple 1 :

Entrée : `n = 4`
Sortie : variable
Explication : La mémoire devrait contenir `1, 2, 2, 3`.

Exemple 2 :

Entrée : `n = 12`
Sortie : variable
Explication : La mémoire devrait contenir `1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5`.

Contraintes :

- `1 <= n <= 50`
- Pour obtenir tous les points, il faut que le code fonctionne pour un `n` quelconque.
- Pour obtenir tous les points, il faut que **l'exemple 2** s'exécute avec ≤ `525` cycles.

Gabarit de code :

```
.text
stop

.data
n: 4
```

Calcul d'une puissance [x86-IA32]

Sujets : Boucles, Branchements, Opérations arithmétiques

Énoncé : Écrire un programme qui calcule la valeur de

$$a^b$$

où

- `a` est un entier positif (la base)
- `b` est un entier positif ou nul (exposant)

Données :

- La base `a` est stockée dans le registre `%esi` au début du programme.
- L'exposant `b` stocké dans le registre `%edi` au début du programme.

Résultat attendu : Placer le résultat calculé dans le registre `%eax` avant la fin du programme.

Exemple :

Entrée : `a = 3, b = 4`
Sortie : `81`

Contraintes :

- `a > 0`
- `b ≥ 0`
- Le résultat a^b entrera toujours dans 32 bits.

Gabarit de code : sur Moodle; fichier `todo_puissance.s`

Compilation et exécution :

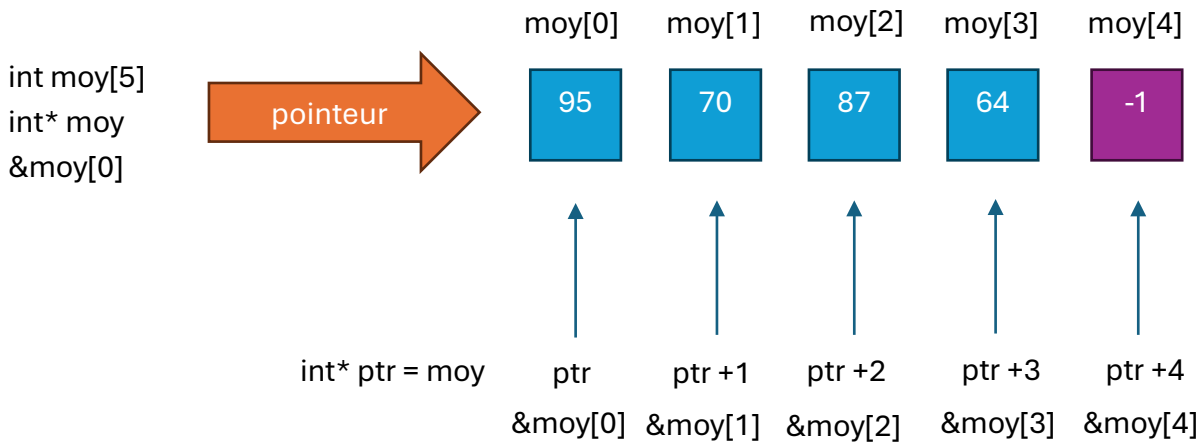
Ouvrir un terminal dans le dossier du TP et d'exécuter la commande :

```
make puissance
```

Moyenne d'une liste [x86-IA32]

Sujets : Boucles, Branchements, Opérations arithmétiques, Tableaux, Types d'adressage

Rappel: Un tableau est une suite contiguë d'éléments du même type dans la mémoire.



Énoncé : Faites la moyenne de tous les éléments dans un tableau. Le nombre d'éléments dans le tableau est inconnu, mais pour faciliter la tâche, le dernier élément du tableau est `-1`. Il ne faut pas compter ce dernier `-1` dans le calcul de la moyenne. Une division entière est suffisante pour le calcul.

Données : Le tableau est **pointé** par l'adresse contenue dans `%esi` au début du programme.

Résultat attendu : Placer la moyenne calculée dans le registre `%eax` avant la fin du programme.

Exemple :

Entrée : `[95, 70, 87, 64, -1]`
Sortie : `79`
Explication : (95 + 70 + 87 + 64) / 4 = 79

Contraintes :

- Le résultat entrera toujours dans 32 bits.

Gabarit de code : sur Moodle; fichier `todo_moyenne.s`

Compilation et exécution :

Ouvrir un terminal dans le dossier du TP et d'exécuter la commande :

```
make moyenne
```

Palindrome [x86-IA32]

Sujets : Boucles, Branchements, Chaines de caractères, Types d'adressage

Énoncé :

Un palindrome est un mot, une phrase, un nombre, ou une séquence de mots qui peut être lu de la même manière dans les deux sens, de gauche à droite comme de droite à gauche, en ignorant les espaces, les ponctuations et les majuscules. Les palindromes sont souvent utilisés dans des jeux de mots et des énigmes, et leur caractéristique unique les rend intrigants et intéressants à étudier.

Par exemple, le mot "kayak" est un palindrome car il reste le même qu'on le lise de l'avant vers l'arrière ou de l'arrière vers l'avant.

Écrivez un programme qui trouve si une chaine de caractère est un palindrome. Le code ne doit **PAS** être **sensible à la casse**.

Données: Le pointeur vers la chaîne de caractère est placé dans `%esi` au début du programme

Résultat attendu : Placer 1 dans le registre `%eax` avant la fin du programme si la phrase est un palindrome et 0 sinon.

Rappel : Une chaine de caractère est toujours terminée par le caractère nul (0x00 ou '\0 ').

Exemple 1 :

Entrée : `KaYak`
Sortie : `1`

Exemple 2:

Entrées : `"inf1600"`
Sortie : `0`

Contraintes :

- Il n'y aura aucun espace ni de caractères spéciaux dans la chaîne.

Gabarit de code : sur Moodle; fichier `todo_palindrome.s`

Compilation et exécution :

Ouvrir un terminal dans le dossier du TP et d'exécuter la commande :

```
make palindrome
```

Inverser la pile [x86-IA32]

Sujets : Boucles, Branchements, Opérations arithmétiques

Énoncé : Écrire un programme qui inverse l'ordre des éléments présents sur la pile. À la fin de l'exécution, le sommet de la pile devra contenir l'élément qui se trouvait initialement au bas de la pile, et inversement.

Données : Les éléments seront dans la pile au début du programme.

Résultat attendu : Inverser les éléments de la pile et terminer le programme.

Exemple :

```
Pile initiale :
10 ← sommet
9
8
7
6
5
4
3
2
1
```

```
Pile finale :
1 ← sommet
2
3
4
5
6
7
8
9
10
```

Gabarit de code : sur Moodle; fichier `todo_pile.s`

Compilation et exécution :

Ouvrir un terminal dans le dossier du TP et d'exécuter la commande :

```
make pile
```