UNIVERSITÉ LIBRE DE BRUXELLES ULB

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-F105 - Deuxième projet d'assembleur

Jacopo De Stefani

Année académique 2017–2018

Résumé

Ceci est l'énoncé du troisième projet du cours « Langages de programmation 1 ». Il s'agit de vous y familiariser avec les appels à fonction en langage d'assemblage, dans les conditions vues au cours et aux séances d'exercices. Veillez à le réaliser soigneusement et à respecter scrupuleusement les instructions présentées ci-dessous.

1 Introduction

Etant donné la diffusion et l'utilisation croissante des crypto-monnaies, les chercheurs du Département d'Informatique de l'ULB ont décidé de créer leur propre crypto-monnaie, dont le nom de code est Double Word Coin (DWCoin, en bref).

Avant de pouvoir effectuer des transactions avec des DWCoins, les utilisateurs doivent générer une séquence de 32 bits, sur base des informations contenues dans la transaction. Lors de la validation d'une transaction, il est nécessaire de vérifier son intégrité (c'est-à-dire l'absence de manipulation). Cette opération est effectuée en vérifiant l'égalité entre la séquence de 32 bits générée avant et après la transmission de la transaction.

Afin d'optimiser la gestion de la mémoire et d'atteindre une efficacité maximale, nous vous demandons d'implanter la procédure de validation d'une séquence en langage assembleur. Cet outil que nous vous demandons d'implanter s'appelle *DWTransactionValidator*.

2 Description du problème

Le générateur de séquences prend comme entrée une chaîne de caractères `message` (représentant les informations de la transaction) en format ASCIIZ [1] de taille maximale de 256 octets (y compris le terminateur nul). Le générateur (Figure 1) se compose de deux blocs principaux : une somme de contrôle et une fonction de projection.

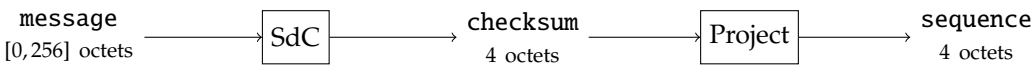

FIGURE 1 – Schéma de l'architecture du validateur des séquences

2.1 Somme de contrôle

Pour calculer la somme de contrôle, une variable checksum c est initialisée à `0x00000000`. La somme de contrôle est ensuite calculée de manière itérative (*cf.* Figure 2).

Le message m est décomposé en n blocs de 4 octets (soit 32 bits). Ensuite une opération de XOR (ou exclusif) est effectuée entre le i^e bloc m_i de 4 octets du message et le checksum actuel c_i. Le résultat est ensuite stocké

1. Voir cours IV.5.4.b ou `https://en.wikipedia.org/wiki/Null-terminated_string`

dans l'état interne du système c (la valeur précédente est écrasée) après un décalage circulaire d'un octet vers la gauche.

Si la taille du dernier bloc du message m_n est inférieure à 4 octets, une extension de 0 pour atteindre la taille de 4 octets sera effectuée avant le XOR.

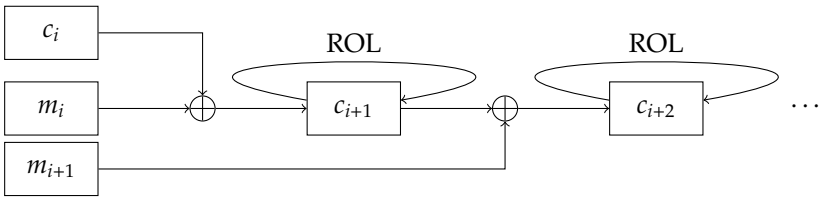

FIGURE 2 – Exemple de deux itérations de l'algorithme pour le calcul de la somme de contrôle

2.2 Fonction de projection

En ce qui concerne la fonction de projection, nous disposons déjà d'une implantation efficace en langage C++ :

PROGRAMME 1 – Fonction de projection

```
1  unsigned project (unsigned checksum) {
2    unsigned tmp = checksum;
3    // tmp = checksum mod 216+1 (a Fermat prime) ; O codes 226
4    tmp = (tmp & 0xFFFF) - (tmp>>16 & 0xFFFF);
5    if (tmp & 0xFFFF0000) ++tmp; // carry
6    tmp &= 0xFFFF;
7    // add-multiply (by PI decimals)
8    tmp += 0x6A88;
9    tmp *= 0x243F;
10   return tmp ^ checksum;
11 }
```

Par conséquent, nous vous demandons d'intégrer ce code source dans (la partie C++ de) votre projet et d'appeler cette fonction depuis votre code en assembleur.

3 Description du logiciel

Vous disposerez d'un fichier de tests écrits en C++ (`SequenceTests.cpp`), ainsi que d'un canevas du code assembleur (`ValidateSequence.asm`).

Nous vous demandons de réaliser la fonction de vérification d'intégrité d'un message.

La signature de cette fonction sera la suivante :

```
bool validate_sequence(char message[], unsigned sequence);
```

dont les paramètres sont

- `char message[]` : La chaîne de caractères à utiliser pour la génération de la séquence.
- `unsigned sequence` : La séquence de 32 bits à valider .

La fonction renverra `true` si la sequence générée à partir de `message` correspond à `sequence`, `false` autrement.

Vous écrirez également un « Makefile » qui produit un fichier exécutable `SequenceTests` à partir de ces sources.

Indications complémentaires

Vous devez soumettre un fichier compressé `.zip` qui contient un dossier. Ce dossier doit s'appeler `<nom>_<prenom>` (pas de majuscules ni de caractères accentués) ; le fichier zip doit porter le même nom. Par exemple, Alan Turing doit soumettre un fichier `turing_alan.zip` qui contient le dossier `turing_alan` avec son projet. Les fichiers à soumettre :

- `ValidateSequence.asm` qui contient l'implantation assembleur de la fonction pour valider une séquence ;
- `SequenceTests.cpp` qui contient les tests pour vérifier l'exécution correcte de l'algorithme ;
- `Makefile` qui produit un fichier exécutable `SequenceTests`.

Bon travail !

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Votre projet doit comporter **votre nom** et **votre section**.
2. Votre code doit être **commenté**.
3. Vous devez suivre les modalités de dépôt sur l'UV : un fichier `.zip` tel que décrit ci dessus.
4. Et respecter le délai : avant le vendredi 6 avril 2018 à 16 heures.

Passé ce moment, les projets sont considérés comme en retard et **ne seront plus acceptés**.
De plus, tout code présentant des erreurs pendant la phase de compilation ou d'exécution sera sanctionné avec un note de 0/20.