Marie-Ange Stefanos
Chloé Poulic

Rapport de projet n°2

27/09/2020

Introduction

Ce livrable 2 permet l'ajout de la commande de désassemblage. Comme pour le livrable précédent, il s'agira aussi de faire attention à la gestion des erreurs et à la génération des traces mais tout d'abord il va falloir permettre le désassemblage des instructions grâce à la gestion complète de la commande disasm. Pour le bon fonctionnement de cette fonction, nous devrons gérer le chargement du fichier texte d'instructions préalablement rédigé afin de créer le dictionnaire associé.

Cette nouvelle étape du projet est aussi une étape de correction et de complétion du livrable 1, les éléments corrigés et ajoutés seront, dans ce rapport, détaillés.

Retour sur le livrable 1

A. La vérification de byte et word

Dans de nombreuses fonctions de commande nous utilisons le fait qu'un entier est un byte ou un word. Cependant dans aucune des fonctions rendues dans le livrable 1 nous avons fait de test d'erreur sur ce type. Nous avons donc créé des fonctions is_byte et is_word dans le module interpreter.

B. Ajustements des commandes disp, set et assert

Comme ces trois commandes sont indispensables pour la suite du projet comme indiqué lors de la dernière séance de BE et qu'elles ne fonctionnaient pas encore comme nous le souhaitions (problèmes d'affichage, certaines cas d'erreur oubliés,...), nous avons pris le temps de rattraper notre retard et de les réajuster afin qu'elles répondent correctement à tous nos fichiers tests. La révision de ces commandes nous a aussi fait penser à de nombreux cas d'erreur que nous traitons mais qu'il reste à ajouter dans nos batteries de tests afin de prouver leur fonctionnement.

Livrable 2

A. Lecture des instructions

Afin de reconnaître une fonction donnée dans le fichier, nous avons créé des fonctions pour reconnaître dans une instruction de type R, I ou J les éléments caractéristiques des instructions. Nous avons opté pour la méthode qui utilise des masques et décalages. La raison de ce choix est une meilleure compréhension du phénomène réalisé. En effet nous comprenons moins bien les champs de bits.

B. Création du dictionnaire

Nous avons choisi comme format de dictionnaire un fichier txt qui contient par ligne le nom de l'instruction puis le code binaire de l'instruction remplit de 0, 1 ou ?, le type de l'instruction (R, I ou J), le nombre de paramètres de l'instruction.

Pour le moment, le dictionnaire contient les instructions suivantes : ADD, AND et BEQ.

C. Fonction chargement du dictionnaire et fonctions associées

1. Créations de listes et des instructions

Afin de stocker les instructions, nous avons défini le type instruction à travers une structure qui prend en compte le nom de l'instruction, sa nature (R, I ou J), son nombre de paramètres utilisés, son ordre, sa signature et son masque. Nous avons alors créé une fonction associée qui permet la création d'une instruction, c'est-à-dire le remplissage des champs de la structure, mais aussi la fonction de destruction pour ne pas créer de fuite de mémoire.

Comme nous allons stocker les instructions lues dans dictionnaire nous allons utiliser un tableau dans lequel nous allons stocker des instructions. Nous avons donc ajouté dans la structure machine un champ qui est ce tableau. Chacune des lignes de ce tableau est une instruction comme définie précédemment. Ce tableau d'instruction fait une taille de 100. Ceci est totalement arbitraire.

2. Analyse d'une ligne du dictionnaire et chargement du dictionnaire dans un tableau

Chacune des lignes du dictionnaire est lue par une fonction, inst_def_parse. La ligne est analysée segment par segment. Les segments sont séparés les uns des autres par des tabulations. Afin de faire la segmentation de la ligne, nous avons créé une fonction en s'inspirant de la fonction

get_next_token utilisée pour scinder une commande rentrée dans l'interpréteur. Au fur et à mesure de la lecture de la ligne, nous stockons les éléments souhaités dans des variables de sortie.

Cette fonction a le comportement souhaité, en effet lorsque nous réalisons des "printf" pour vérifier ce que contiennent ces paramètres de sortie, les affichages donnent les valeurs souhaitées.

Dans la fonction instructions_file_load, nous appelons la fonction d'analyse. Nous affichons comme précédemment les valeurs souhaitées. Ceci fonctionne. Cependant, lorsque nous voulons remplir les champs de la structure, ceci a longtemps créé une erreur de segmentation. Elle n'a été repérée que très tardivement (par rapport à la date du rendu du livrable 2). En effet, le final du livrable 1 nous a permis de remarquer cela, nous avons donc débuggé ceci, mais nous n'avons pas eu le temps de chercher davantage pour la vérification de la fonction cmd_disasm.

D. Fonctions utilisées dans la commande de désassemblage

Nous avons écrit un grand nombre de fonctions auxiliaires qui permettent les différentes étapes de disasm.

Tout d'abord nous avons fait la fonction check_plage_format_disasm. Cette fonction permet de vérifier que les plages de hexadécimaux données à l'interpréteur sont correctes (pas d'adresses inexistantes, la deuxième plage est plus grande que la première…). Une autre fonction de vérification sur les plages est réalisée, c'est check_size_format_text qui permet de détecter si les plages données ne couvrent pas une plage plus grande que le segment .text.

Puis, nous récupérons le contenu de chacune des adresses grâce à la fonction get_mem_word. Celle-ci effectue le changement de poids. Nous avons donc récupéré le mot à désassembler. Ce qui est fait dans la fonction desassemble.

Cette fonction permet d'abord la comparaison du mot relevé par la fonction get_mem_word. Une fois une corrélation retrouvée avec une véritable instruction, nous relevons le type de celle-ci. Puis suivant le type, on effectue une disjonction de cas mais le principe reste le même pour les 3 types. Nous récupérons le nom de l'instruction puis les différentes variables grâce aux fonctions créées préalablement, les get_……

Puis, une fois toutes les valeurs relevées, nous affichons le contenu de chacun des segments grâce à des fonctions d'affichages qui sont spécifiques à chaque type.

E. Les étiquettes

Par souci de temps, nous n'avons pas pu implémenter tout ce qui est relatif aux étiquettes, cependant nous avons compris leur rôle et ce que nous devons rattraper en plus pour le livrable suivant.

Une étiquette peut modéliser un fait une commande, il va donc falloir les repérer dans le code et ensuite faire l'association entre l'étiquette et le symbole qu'elle représente.