Projet Informatique

Émulateur de Microprocesseur MIPS

Livrable 2 : Désassemblage des instructions

Livrable 1 : Registres et mémoire

Ismail Guedira

Hugo Vallée

Introduction

L'objectif de ce second livrable du projet a été le désassemblage des instructions c'est à dire la conversion d'un mot situé à l'adresse .TEXT en une instruction assembleur qu'on va afficher. Cependant nous avons également eu beaucoup de travail par rapport au premier livrable sur lequel nous avons eu du retard. Nous avons ainsi pu désassembler les instructions et bien avancer par rapport au retard accumulé sur le premier livrable.

I. Démarche générale du projet

La plupart des premières tâches que nous avons eu à réaliser ont d'abord été les parties manquantes du premier livrable. On a ainsi travaillé sur les registres et les fonctions sur la mémoire -load, affichage...- qui sont en partie nécessaires au désassemblage final des instructions. Ensuite nous avons pu faire les vérifications des paramètres d'entrée de l'interpréteur -valeur de l'adresse, aligment sur 4 octets...-. Une fois ces vérifications faites, nous pouvons exécuter la commande de désassemblage de la partie .TEXT de la mémoire.

1. Désassemblage

Le désassemblage a été réalisé en cinq fonctions pour éviter de trop rallonger notre code, il y a donc **cmd_disasm** qui gère les appels à toutes nos fonctions associées au désassemblage : test_cmd_disasm, erreur_fonction_disasm, execute_cmd_disasm et return_operande. Ainsi la fonction **test_cmd_disasm** s'assure que les adresses demandées correspondent bien à des adresses effectives, une plage d'adresse... Ensuite la fonction

execute_cmd_disasm va réaliser le désassemblage effectif en différenciant les cas avec deux grilles de lecture : une en fonction du type et l'autre en fonction du nombre d'opérande. On utilise dans cette fonction **return_operande** qui définit le type de l'instruction, on utilise pour ça une structure avec trois éléments : R, I et J. Enfin la fonction **erreur_fonction_disasm** centralise toutes les erreurs possble, et renvoie selon le type d'erreur le message correspondant. Nous avons ensuite testé le désassemblage à partir des fichiers ELF qui nous ont été donnés pour nous assurer que les désassemblages correspondaient bien à ce qu'on attendait.

2. Lecture du dictionnaire

Pour la lecture du dictionnaire, on crée une variable "définition" qui pointe sur une structure composée de la signature, le masque, le nom, le type de l'instruction, la map des opérandes et le nombre d'opérande. Il s'agit ensuite d'une fonction assez classique qui prend en paramètre un nom de fichier -permettant éventuellement de créer des dictionnaires différents-. Pour le test de la lecture, nous avons simplement affiché la zone mémoire où on a chargé le dictionnaire pour nous assurer que les éléments en mémoire correspondent bien à ce qu'on attendait.

3. Fonctions associées à la mémoire et aux registres

Pour permettre le désassemblage des instructions, nous avions besoin d'une fonction de chargement du fichier ELF dans la zone .TEXT de la mémoire et de deux fonctions de renvoi et affichage d'un mot stocké en mémoire.

Pour les fonctions associées aux registres, nous avons à rajouter un certain nombre de fonctions de manipulation. Pour plus de simplicité, nous avons décidé de travailler quasi exclusivement avec le numero entier des registres. Si on nous renvoie une mnemonique ou un numero type '$3', nous avons crée des fonctions assurant la conversion en un numero entier avant d'appeler nos fonctions utilisant nos numeros entiers. Il est probable que ces fonctions seront amenés à évoluer légèrement notamment par rapport aux droits en écriture et lecture de certains registres qui peuvent ou ne peuvent pas être modifiés suivant les fonctions.

Les tests associés aux registres nous ont permis de nous assurer que nos fonctions s'exécute en modifiant, copiant... les registres les uns dans les autres. Le debuggage de la fonction n'a pu être terminé mais les fonctions associées aux numeros entiers et mnémoniques fonctionnent. Certaines des fonctions citées précédemment n'ont pas été implementées au sein des fichiers exécutables mais sont présents dans les fichiers de test.

II. Problèmes rencontrés

Nous avons rencontré plusieurs problèmes au cours de la réalisation de ce livrable.

1. Affichage de certains nombres signés

En effet, lors du désassemblage avec la fonction réalisée, on peut remarquer des différences que nous n'arrivons pas à expliquer. En effet, pour le fichier boucle.o on obtient : 300c :: 1549fffe BNE $t2,$t1,-2 , 0xfffe est bien le code hexadécimal pour -2 et, s'il s'agit bien d'un entier signé, nous avons -8 comme valeur pour "immediate" -dans le fichier assembleur donné sur le site-. De la même façon, nous avons pour le fichier procedure.o : 300c :: 0c000009 JAL 9 qui ne correspond pas à la valeur théorique qu'on est sensée avoir.

2. Affichage de l'ensemble de l'instruction désassemblé

L'un des soucis qui nous a pris le plus de temps à résoudre a été l'affichage complet de l'instruction. En effet, s'il a été relativement facile d'afficher le nom de l'instruction -ADD, SUB...-, afficher les opérandes à poser quelques problèmes.

3. Fin du code à désassembler

La fonction de desassemblage ne présente pas une limite pour indiquer la fin du code, ainsi si le code est sur 24 octets, on peut toujours désassembler jusqu'à la fin de la zone .TEXT, il n'apparaitra que des NOP, indiquant finalement que rien ne se passe.

III. Répartition du travail

Pour la répartition du travail, Ismail Guedira a principalement travaillé sur le coeur du livrable 2 avec la fonction de désassemblage et quelques fonctions traitant la mémoire tandis que Hugo Vallée s'est principalement concentré sur les fonctions associées aux registres, le reste de ce qui touchait à la mémoire et quelques points associés au désassemblage (dictionnaire notamment).