Deuxième livrable – Projet Info SEI

SIMULATEUR MIPS

Nicolas SCHOEMAEKER
Éric FELTRIN

Grenoble INP - Phelma
2A – SEI

Introduction

À l'issu du premier livrable de ce projet info, nous avons mis en place l'interpréteur de commandes, qui permet d'interagir avec le simulateur. Nous avons implémenté les commandes basiques liées à l'environement et les fonctions basiques de gestion de mémoire.

Ce deuxième livrable a un objectif plus concrêt : celui de fournir la première véritable interaction avec des fichiers binaires ELF. En particulier, nous avons implémenté la fonction de chargement de programme LP, dont le rôle est de peupler la mémoire de notre structure de MIPS avec des données réelles. Maintenant que notre structure de MIPS comporte des véritables données, la fonction DM affiche les véritables valeurs (en ce qui concerne notre binôme, aucun ajustement de cete fonction n'était à faire, puisque nous avions codé de manière assez générique).
Enfin, la fonction DA a également été implémentée, de sorte à afficher le code héxadécimal de l'instruction assembleur présente à cette adresse, puis de la désassembler afin d'afficher l'instruction en clair (*i.e.* en assembleur).

Implémentation de l'incrément

Le chargement de fichiers ELF

Dès le premier incrément, nous avions établi une structure saine et robuste de notre architecture MIPS, en définissant des sections de mémoire pour stocker nos données (factices et aléatoires dans un premier temps, réelles maintenant) ; de ce fait, le chargement de fichers ELF se trouvait être simplifié : nous devions faire l'interface entre la fonction mipsloader donnée (et les SectionELF) et notre architecture.

Pour ce faire, nous commencons par allouer les SectionELF afin de fournir un conteneur de réception à la fonction mipsloader, puis nous récupérons la taille des différents segments depuis ce conteneur pour allouer notre propre espace de mémoire. Nous transferons les données dans notre structure de mémoire de notre architecture MIPS et nous détruisons la mémoire allouée par les SectionELF. Bien entendu, des tests sont réalisés à chaque étape de tentative d'allocation de mémoire.

Désassemblage d'instructions (commande DA)

Nous avions, lors du précédent livrable, parsé et affiché le code assembleur sous forme hexadécimale, l'objectif de ce livrable était de désassembler le code pour retrouver les instructions assembleur.

Pour réaliser cette opération, nous avons tout d'abord implémenté un dictionnaire contenant l'ensemble des instructions. Ce dictionnaire est chargé dans notre structure mips à partir d'un fichier texte où sont listées les instructions ainsi que l'opcode correspondant.

Chaque instruction est placée dans le tableau correspondant à son type (R, I, J) avec son nom, son opcode, ainsi que l'ordre des opérandes. Pour coder l'ordre des opérandes, nous avons ajouté un entier à notre structure qui définie l'ordre des paramètres à lire dans le code d'instruction (ex : order = 312, l'ordre d'écriture des paramètres est 2, 1 puis 3).

Une fois notre structure chargée, nous avons réalisé la fonction de désassemblage. Les codes d'instruction de type R commencent systématiquement par 000000 ce qui nous a permi de les traiter à part ; ensuite on parcourt les tableaux à l'aide d'une boucle for afin de

trouver l'instruction correspondante en comparant les opcode. Ensuite il ne reste plus qu'à interpréter l'entier codant l'ordre des opérandes pour écrire la la ligne assembleur correspondante.

Conclusion

À l'issu de ce deuxième incrément, nous sommes désormais capables de charger un fichie ELF sans relocation, de mapper toute sa mémoire et d'afficher correctement toutes les données qu'elle contient. Nous sommes également capable de parcourir la mémoire du segment .text et de désassembler les instructions qui y sont stockées afin de faire l'état des instructions.
C'est la première étape concrête de notre simulateur, puisqu'elle fait le lien avec des fichiers réels, compilés. Bien sûr, tout ceci s'appuie sur le premier incrément, puisque toute l'implémentation se fait dans l'interpréteur de commandes.
La notion d'incrément montre ici toute son importance.