ZEGGAI Saïd – DEIN Raphaël

Rapport second livrable

Pour ce second livrable, nous nous sommes d'abord focalisé sur l'écriture des fonctions de l'incrément 1 que nous n'avions pas eu le temps de réaliser. Nous sommes maintenant pleinement investis dans l'écriture de la fonction disasm. Nous avons mis en place les grandes lignes de cette fonction.

Travail réalisé

Nous avons fini l'incrément 1, c'est-à-dire :

- écrit les commandes set, assert, debug et resume (les autres étaient écrites) ;

- réécrit la commande disp mem map pour qu'elle affiche le résultat escompté ;

- réécrit la partie qui s'occupait de récupérer les tokens de disp et de load ;

- réorganisé les includes ;

- rectifié tous les warnings ;

- réécrit les fonctions qui vérifient les types des nombres (is_hex, is_oct, is_dec) ;

- mis en place la désallocation ;

- ajouté des tests à toutes ces commandes (76 tests) ;

Cette partie nous a pris un temps conséquent qui nous a empêché de nous concentrer sur la fonction disasm. Néanmoins nous avons :

- mis en place l'analyse syntaxique de la commande (fonction disasm) ;

- mis en place le chargement du dictionnaire au préalable écrit (load_dic) ;

- mis en place la lecture du code .o (disasm_plage) ;

- mis en place la reconnaissance de l'instruction à l'aide de l'opcode et du masque (certaines commandes ne sont pas reconnues cependant) ;

- écrit quelques tests pour cette fonction (certains ne passent pas à cause des instructions non reconnues).

Nous avons réfléchi au parsing des paramètres des instructions, mais n'avons pas eu le temps de le mettre en place. Nous avons bien conscience que cette phase de programmation sera longue.

Compétences acquises

Savoir trouver l'erreur lorsque l'on a un Seg Fault.

Parser un fichier texte.

Désallouer correctement la mémoire (il n'y a pas d'erreurs en tout cas).

Utiliser GitHub et Git.

Travailler en ligne de commande.

Ecrire des scripts.

Points de satisfaction

L'écriture des tests qui nous paraissait d'abord rébarbative et peu pratique permet en fait de gagner beaucoup en efficacité, sitôt qu'on est habitué à leur utilisation et que l'on a écrit quelques scripts pour qu'ils soient utilisables plus facilement (tests.sh, ftests.sh, ces scripts nécessitent des ressources hors de l'archive et ne marcheront pas par conséquent).

Nous n'avons plus de problème extérieur au code. C'est-à-dire que le projet est bien en place et que nous nous focalisons uniquement sur des problèmes liés à la programmation et non pas des problèmes de Makefile, de GitHub, etc.

Problèmes rencontrés

Le temps de déboggage est souvent nettement plus long que le temps d'écriture des fonctions. Il faut donc sans doute que nous passions plus de temps à réfléchir avant d'écrire les fonctions, en anticipant les erreurs probables. Il serait pratique de pouvoir tester une fonction seule avant de l'utiliser, est-ce possible d'intégrer ce genre de fonctionnalité dans un programme ?

Il faut que nous améliorions notre efficacité.

En terme d'organisation, nous avons commis l'erreur de travailler sur la même fonction en même temps. Même si Git est sensé gérer ce genre de problématique, cela a engendré des problèmes et été chronophage.

Conclusion

Nous devons améliorer notre efficacité. Etre plus méthodique afin de réduire le temps de déboggage.