TAURAND Sébastien
BERTRAND Antoine

PROJET INFORMATIQUE 2ème année

ASSEMBLEUR MIPS 32 BITS

Livrable 4 : Génération de la liste d'assemblage et Du code binaire

Introduction

La dernière étape de ce projet consiste à générer la liste d'assemblage et le code binaire d'un fichier assembleur MIPS.

Organisation du travail

Ne restant plus que l'encodage des instructions et la génération du code binaire le partage du travail a été complexe, nous nous sommes organisé de la façon suivante, l'un effectue ces différentes tâches et l'autre vérifie ces procédures et les corrige en cas de besoin.

Les Outils

La documentation des structures et des procédures a été effectuée à l'aide de Doxygen. Comme toujours l'outil valgrind a été indispensable pour contrôler les fuites de mémoires en particulier dans les cas très nombreux de gestion d'erreurs.

La commande : `testing/simpleUnitTest.sh -e ./as-mips -b test_ref/*.s` nous a permis de tester automatiquement l'ensemble de fichier de référence en comparant les listes d'assemblage.

Pour effectuer la comparaison des fichiers objets la commande `diff miam.o miam.obj` est suffisante et en cas d'erreur la commande `od -t x1 miam.o` permet de visualiser l'ensemble du code en hexadécimal et de réaliser une analyse octet par octet.

Génération de la liste d'assemblage

Lors de l'étape précédente il ne restait plus qu'à faire :

- L'encodage des instructions
- Organiser la table d'étiquette dans l'ordre des lignes d'apparition des symboles correspondants
- Faire la mise en forme exacte de la liste d'assemblage (bon nombre de tabulations, espaces, retour chariots, etc.).

L'encodage des instructions s'effectue à l'aide de la table des définitions d'instructions où toutes les informations nécessaires sont listées, pour encoder une instruction on considère dans un premier temps son code opérande vierge (qui définit une instruction) une boucle est réaliser sur l'ensemble des opérandes de l'instruction, le code opérande est rempli au fur et à mesure.

Pour afficher la table des étiquettes dans le bon ordre, une liste d'étiquette a été créée, dès que l'on rencontre un symbole, ce symbole est ajouté à la liste (sauf si ce dernier est déjà dans la liste) il reste ensuite simplement à afficher les éléments de la liste dans l'ordre naturel de la liste.

Génération du code binaire

La structure du code binaire est très simple, les longueurs de sections sont gardé en mémoire après l'analyse syntaxique et les informations sur les instructions et les données sont les mêmes que pour la liste d'assemblage. Notre choix a été d'implémenter un buffer pour stocker les informations de la section data et ainsi gérer facilement les trous éventuels dus au réalignement des structures de données .word et .half.

Tests réalisés

Nous avons conçu différents fichiers de tests. Dans un premier temps nous nous sommes attachés à vérifier si l'ensemble des instructions fonctionnaient pour l'intégralité des instructions avec différents opérandes, l'étape suivante étant de tester toutes les instructions avec des valeurs d'opérandes minimales et maximales (par exemple 0x0000FFFF pour un opérande de 16 bits).

La vérification des erreurs est elle aussi indispensable nous avons réalisé différents fichiers de test incluant des erreurs lexicales, un autre sur les erreurs syntaxiques de premier niveau, un autre sur les erreurs syntaxique de niveau supérieur.

La dernière étape étant de vérifier l'exécution du programme sur les tests de références fournis renvoie les bonnes listes d'assemblage et les bons codes binaires.

Problèmes rencontrés

Nous avons perdu beaucoup de temps suite à l'analyse entre nos fichiers générés et les fichiers références fournis sur le site dont quatre sur huit étaient erronés. Nous avons signalés ces erreurs à Monsieur François Portet afin qu'il puisse corriger le site et informer les autres binômes.

Le format de la liste d'assemblage n'étant pas réellement spécifié, nous avons pris comme règles tout ce qui était contenu dans les fichiers référence. Nous avons dû supposer certaines règles d'écriture pour des cas non contenus dans les fichiers référence. Cette dernière partie devra donc peut-être être modifiée le jour de l'examen en cas d'hypothèses différentes.

Points à améliorer

Le sprint final a été particulièrement chronophage, l'envie de perfectionner et raffiner l'écriture de chaque partie du programme pouvant prendre énormément de temps.

La clarté et le détail contenu dans les messages d'erreurs pourraient très certainement être bien améliorés, même s'ils se révèlent suffisant à l'utilisation pour identifier et corriger les erreurs dans le fichier assembleur.

Il aurait fallu aussi commenter chaque ligne des fichiers de tests, en particulier ceux contenant des erreurs afin de préciser ce que l'on cherchait à vérifier pour chaque ligne. Nous n'avons malheureusement pas eu le temps de le faire.

Résultats

L'ensemble du projet est doté d'une documentation basique mais qui suit les règles de syntaxe de Doxygen et documente toutes les fonctions et structures.

Nous avons aussi gérer de façon extensive sur les erreurs de traitement qu'elles soient liées à des problèmes d'insuffisance mémoire ou d'erreurs dans les fichiers sources.

Notre programme semble fonctionnel aussi bien dans les cas de fichiers assembleurs sans erreurs qu'avec sur les tests que nous avons pu faire, qu'ils soient sur les fichiers référence ou sur nos fichiers de tests personnels.

Le verdict final sera bien sûr les résultats de comparaisons le jour de l'examen.