TAURAND Sébastien
BERTRAND Antoine

PROJET INFORMATIQUE 2ème année

ASSEMBLEUR MIPS 32 BITS

Livrable 3 : Analyse grammaticale

Introduction

La troisième partie de ce projet consiste à réaliser l'analyse grammaticale du fichier assembleur. L'objectif étant de déterminer la validité du code et d'effectuer l'encodage des instructions ainsi que de préparer la gestion des relocations.

Organisation du travail

La répartition du travail a été la suivante : l'un s'est occupé de compléter la gestion des instructions tandis que l'autre s'est afféré à la gestion des relocations et pseudos instructions.

Les Outils

Nous avons utilisé un désassembleur en ligne pour vérifier la validité de l'encodage de certaines instructions.
Comme toujours Valgrind a été d'une grande utilité afin d'identifier les erreurs liées à la gestion dynamique de la mémoire et des pointeurs associés.

Améliorations du livrable 2

Les points à améliorer identifiés dans le livrable 2 ont été implémentés dans le code. Cela inclut :

- Gestion des caractères et chaines de caractères.
- Gestion des .half (16 bits).
- Gestion des erreurs mémoires avec sortie propre sur la majorité du code (en cas d'insuffisance mémoire, on libère toute la mémoire déjà utilisée de façon propre avant de sortir en erreur).
- Nettoyage du code machine d'état.

L'analyse grammaticale

L'analyse grammaticale du fichier assembleur est réalisée à partir des données de trois dictionnaires :

- Le dictionnaire de registre qui contient les noms acceptés ainsi que les valeurs entières correspondant à chacun de ces noms.

  Une ligne du dictionnaire de registre est de la forme suivante :

Nom du registre	Valeur du registre
$zero	0

- Le dictionnaire d'instructions qui contient le nom de l'ensemble des instructions et pseudo instructions reconnues, le nombre de leur opérande, le type d'opérande attendu, ainsi que le type de relocation associée. De plus pour les instructions, le dictionnaire contient le code opérande vierge, ainsi que la définition des opérations à effectuer sur les opérandes pour les insérer dans le code opérande (nombre de bits, bit de destination, donnée signée ou non, décalage à effectuer).

Une ligne du dictionnaire d'instruction est de la forme suivante :

Nom de l'instruction	Nombre opérande	Type instruction	Type relocation	Code opérande vierge	Nombre de bits pour coder l'opérande 1	Valeur signée ou non opérande 1	Bit de décalage opérande 1	Bit de destination opérande 1	A complét si plus d'un opérand
J	1	N	2	0x08000000	26	1	0	0	…

- Le dictionnaire de pseudo instructions qui contient quant à lui le nombre d'instructions de remplacement ainsi que l'encodage des opérandes à passer.

Une ligne du dictionnaire de pseudo instruction est de la forme suivante :

Nom pseudo-instruction	Nombre d'instruction de remplacement	Nom de l'instruction de remplacement	Opérande 1 de l'instruction de remplacement	Opérande 2 de l'instruction de remplacement	Opérande 3 de l'instruction de remplacement	A compléter si plus d'une instruction de remplacement
MOVE	1	ADD	1	2	"$zero"	…

Remarque : Pour le dictionnaire de pseudo-instructions, les opérandes passés entre double cotes sont retranscrits textuellement tandis que ceux passés sans double cotes sont les indices des opérandes de la pseudo-instruction.

Les pseudo-instructions sont remplacées par leurs instructions de remplacement ainsi que leurs opérandes associés tels que défini dans le dictionnaire.

Le traitement des instructions LW et SW a lui aussi été implémenté afin de différencier les cas où un symbole est utilisé au lieu d'un adressage offset(base). Les simplifications d'écriture omettant base ou offset ont aussi été traitées.

L'insertion d'instructions de remplacement pour les pseudo-instructions et pour LW et SW nécessite la création de lexèmes qui ne font pas partie de la liste de lexème générée par l'analyse lexicale. Afin de libérer la mémoire associé à ces lexèmes à la sortie du programme que ce soit en conditions normales ou en conditions d'erreurs, ils sont stockés dans une liste chainée générique disposant de son destructeur.

Quatre types de relocations ont été implémentés :

- R_MIPS_LO16 pour les instructions prenant en compte des opérandes 16 bits signés (branch, LW, SW)
- R_MIPS_HI16 pour charger la partie haute d'une adresse (LUI)

- R_MIPS_26 pour les instructions de saut (J, JAL)
- R_MIPS_32 dans le cadre des données de type .word

L'encodage des instructions a été bien entamé, il ne reste plus qu'à compléter le code opérande.

Test réalisés

Nous avons réutilisé les fichiers de tests précédents en y rajoutant les pseudo-instructions et les différentes syntaxes possibles pour les instructions LW et SW.

Problèmes rencontrés

Les modifications du code (incluant la mise à niveau prévu de step 2) ont été assez longues à implémenter ce qui ne nous a pas laisser suffisamment de temps afin de nous construire un fichier de test assembleur exhaustif incluant l'ensemble des instructions données, mode d'adressage, etc. nécessaire à la validation du programme.

Résultats

Le programme documenté a été testé de façon plus succincte que nous aurions voulu mais semble bien fonctionner avec nos fichiers d'exemples.

Points à améliorer

La génération du fichier de listage n'est pas aussi complète et esthétique que l'exemple du sujet.

Certaines parties de code peuvent encore bénéficier d'un « nettoyage ».