



Rapport

Projet Informatique : Assembleur MIPS

Livrable 1

Année Universitaire 2018 - 2019

BRITTON Giovani - Giovani-Crasby.Britton-Orozco@grenoble-inp.org
WRIGHT Gilles - gilles.wright@grenoble-inp.org

Professeur encadrant : Katell Morin-Allory - katell.morin-allory@imag.fr

Filière : SEI

I. Démarche Générale

Le but de ce projet informatique est de réaliser un assembleur MIPS, codé en C, capable de compiler un programme écrit en assembleur. La première étape consiste à réaliser la lecture et l'identification des différents lexèmes ainsi que leurs classement par type : c'est l'analyse lexicale. Pour ce faire, trois débuts de fonctions permettant d'une part la lecture d'un fichier texte - *lex_load_file(char *file, unsigned int *nlines)* - , et d'autre part le découpage du texte en lexème - *getNextToken(char** token, char* current_line)* - et l'identification des lexèmes lues - *lex_read_line(char *line, int nline)* -, étaient à disposition. Les fonctions étant imbriqués, il faut d'abord bien réaliser le découpage du texte en lexèmes afin que ces derniers puissent ensuite être analysés afin de déduire leur nature. Le programme devra aussi gérer les erreurs, c'est-à-dire repérer lorsque la syntaxe du lexème lu ne correspond pas au format admissible en langage assembleur MIPS.

II. Modélisation du problème

Afin de réaliser la découpe des mots en lexème, il a d'abord fallu répertorier l'ensemble des catégories de mot du langage assembleur que l'on peut rencontrer : [COMMENTAIRE], [SYMBOLE], [DIRECTIVE], [REGISTRE], [DEUX-POINTS], [VIRGULE], [DECIMAL], [HEXADÉCIMAL]. Le retour à la ligne n'a pas été pris en compte parmi les états. Ainsi la fonction de découpage en lexème n'en tient pas compte. Cependant, afin de sauvegarder la ligne d'un lexème pour la gestion des erreurs, un décompte des lignes est réalisé.

Dans un premier temps, la fonction *getNextToken* découpe les lexèmes selon le premier caractère du mot, en prenant en compte les différents séparateurs (l'espace, la virgule, le retour à la ligne, les deux-points, le diez, les parenthèses et les guillemets.)

Dans un deuxième temps, les lexèmes sont récupérés dans la fonction *lex_read_line* pour déterminer leur nature. De nouveaux états ont été introduits pour une bonne analyse des lexèmes. Cela peut être modélisé par la machine à états présenté en annexe.

III. Organisation du travail

Au début, il a fallu étudier les différentes catégories de lexèmes du langage assembleur MIPS. Ensuite du temps a été consacré à la compréhension des fonctions de départ. Nous nous sommes alors focalisés sur la réalisation de la machine à états. Pour atteindre cet objectif les tâches ont été organisées comme suit:

- Définition des objectifs de programmation en rapport au MIPS.

- Connaissance des instructions de programmation pour la mise en marche du MIPS nécessaire a les spécifications du projet.
- Conceptualisation des données.
- Analyse des instructions d'assembleur à utiliser.
- Définition de machine aux etats.
- Mise en marche du codage à utiliser par rapport au machine aux etats.
- Période d'essai et de correction.
- Finalisation du premier étape.

En premier lieu, la définition des objectif de programmation consistait en conceptualiser le matériel de soutien du cours, avec les fichiers liées aux composants de MIPS et de visualiser les réquisits nécessaire pour gérer correctement ce qui est demandé. En suite, la connaissance du fonctionnement des instructions et le codage assembleur ont été abordé, c'est dans cette étape là qu'on a visualiser la portée du travaille et la structure du langage ou en d'autres termes, pouvoir connaître la logique des entrées données par l'utilisateur. Une fois finis l'étape d'investigation et conceptualization du projet l'étape de construction et codification de l'entreprise a eu lieu. C'est periode consistait en la définition de la machine aux etats et la caractérisation du comportement de chacun de cas. Le schéma général du machine aux etats se composait d'un statut INIT où le programme initialisé, c'est depuis ici que les entrées sont enregistrés pour pouvoir définir chaque lexème et son correspondant condition lexical, après de la spécification schématique de la machine aux etats la codification de ce qu'a été défini a eu lieu, ici les fonctions ont été travaillés séparément pour chaque membre du group et ensuite ils ont été jointes. Finalement, l'analyse lexicale a terminé avec l'essai de fichiers de test dans le but de garantie que les instructions soit bien enregistrées et associées selon leur categorie lexical, ce dans cette étape là que le codage était corrigé avec les sorties des tests.

IV. Méthode de gestion des erreurs

Pour la gestion des erreurs, nous avons préféré ne pas utiliser un état d'erreur, car il faudrait par la suite identifier cette erreur pour en informer l'utilisateur. Nous avons choisi une méthode similaire au compilateur en C qui n'arrête pas la compilation à la première erreur rencontrée, mais qui analyse tout le code et qui renvoie à l'écran toutes les erreurs rencontrées. Pour réaliser cela, lorsqu'on est dans un état et que le caractère suivant engendre une erreur, un printf est réalisé renvoyant la ligne d'erreur, le lexème posant problème, ainsi que l'état dans lequel se trouvait le lexème au moment de l'erreur.

V. Tests réalisés

Pour tester le programme complet, nous avons utilisé le code en assembleur qui était fourni. Nous avons par la suite, écrit un deuxième code en assembleur regroupant une bonne partie de la syntaxe utilisé en langage assembleur mais aussi des chaînes de caractères sans espaces, ou pouvant générer des erreurs, pour voir si le programme découpe bien tous les lexèmes, et les identifie bien. Nous vérifions nous même si les mots étaient bien découper, et bien associé à leur type. Par la suite, lorsqu'il a fallu utiliser les files pour stocker les lexèmes, nous avons réalisé un programme de test qui vérifie le bon fonctionnement de chaque fonction.

VI. Problèmes rencontrés et travail restant à faire

En ce qui concerne le travail restant à faire, une amélioration de la gestion des erreurs est encore possible. Il serait intéressant par exemple d'identifier les possibles causes d'erreurs et pas uniquement l'état posant problème. D'autre part, le choix de la structure pour stocker les lexème, ainsi que la réalisation de cela, a requis beaucoup de temps. Enfin la complexité du programme pourrait être amélioré, car le code en assembleur est parcouru deux fois : lors du découpage du lexème, et lors de l'identification de sa catégorie. Néanmoins cela ne devrait pas poser problème tant la taille du code en assembleur reste raisonnable.

VII. Conclusion

Pour conclure, un des facteurs les plus importants de l'analyse lexicale est de bien découper les lexème pour qu'ils puissent être identifier par la suite. De plus il faut stocker ces information de telle façon qu'elles puissent être utilisé par la suite lors de l'analyse grammaticale.