Davy Nil
Besson Thomas

Livrable 3
Analyse Syntaxique
Partie 2
Projet Informatique : Assembleur MIPS
SEI 2018 - 2019

1. Introduction

L'analyse syntaxique est l'opération qu'effectue un compilateur après l'analyse lexicale. Elle permet de vérifier la structure du programme : si les instructions sont situées à leur place, si leur paramètres sont du bon type etc... Précédemment nous avions déjà conçu une partie de l'algorithme qui structurait le code en 4 grandes sections : les instructions, les symboles, la section .data et le section .bss. Pour ce qui est de l'analyse syntaxique en elle même il nous restait à vérifier les types des opérandes passés aux instructions dans la section .text. De plus dans l'objectif de réaliser du code relogeable nous avons mis en place une table de relocation.

2. Analyse des opérandes

Après avoir séparé le code entre les différentes sections, il est nécessaire de vérifier la section .text et les opérandes de chaque instruction. Pour cela, il a d'abord fallu modifier la table de hachage mise en place. Les informations sur le nombre d'opérande après une instruction et le type de chacune d'elles ont été ajoutés.

La première vérification effectuée porte sur le nombre des opérandes. Si le nombre attendu ne correspond pas au nombre d'opérande détecté alors il y a une erreur.

Le seconde vérification vient à la fin de l'analyse syntaxique et porte sur le type de chaque opérande. Grâce aux informations rajoutées dans la table de hachage, chaque opérande après une instruction est vérifiée. Le type de l'opérande et sa valeur sont examinés.

Cas particulier : Les pseudo-instructions

Lors du remplissage de la file .text, contenant toutes les instructions, l'équivalence entre les pseudo-instructions et les instructions réellement effectuées est réalisées. Lorsqu'une instruction capable d'être une pseudo-instruction est détectée, une fonction est appelée

pour vérifier s'il s'agit bien d'une pseudo-instruction et si c'est le cas effectue le remplacement de celle-ci.

Exemple d'exécution de cette partie du programme :

Figure 1 : Exemple de la gestion des erreurs sur les bases offset avec le fichier test_base_offset.s

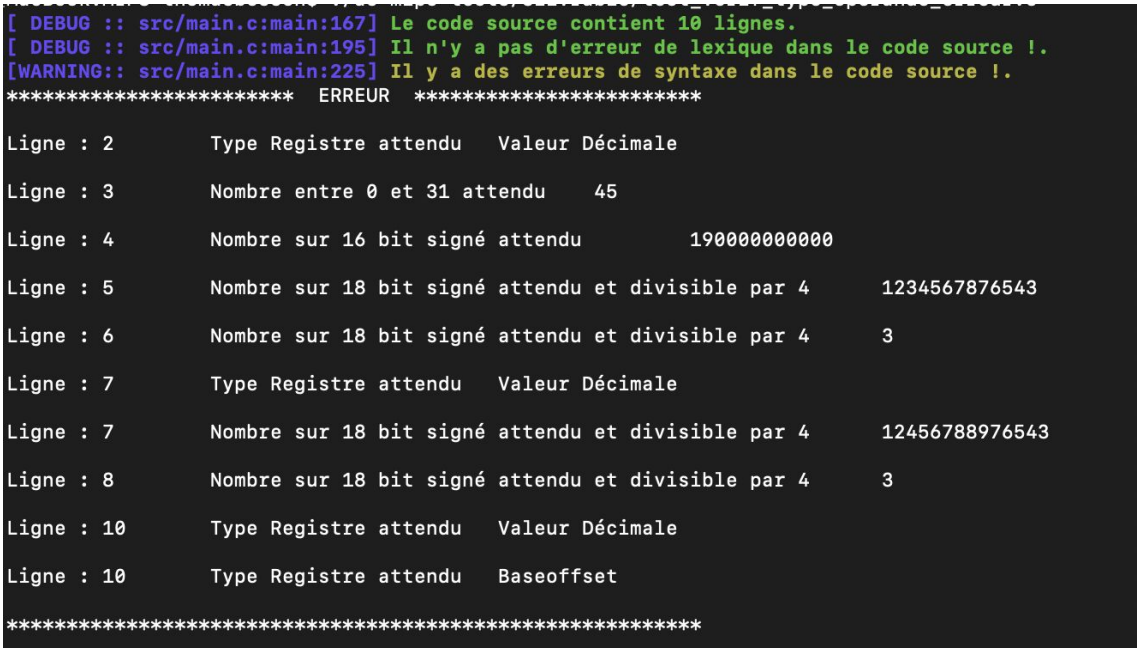

Figure 2 : Exemple de la gestion des erreurs sur les bases offset avec le fichier test_verif_type_opérande.s

3. La table de relocation

L'utilisation de symbole dans le code assembleur pose quelque soucis pour la transformation du code en binaire. En effet en utilisant des étiquettes on manipule en réalité des adresses mémoires qui ne seront connues uniquement au moment de l'édition des liens. Afin d'avoir un code qui puisse être utilisé sur n'importe quelle machine et sans connaître les adresses en mémoire des différentes section (.text, .data et .bss) nous transformerons le code assembleur en un fichier de type Elf qui permet d'avoir un code relogeable. Cela impose la mise en place d'une table de relocation. Ça construction se fait après avoir construit nos différentes collections : celle des instructions, celle des symboles etc...Il suffit ensuite de parcourir la liste des instructions et celle de data à la recherche des renvoie vers des étiquettes. Cela impose de résoudre différents cas de l'utilisation des étiquettes afin de connaître le type de relocation à utiliser (R_MIPS_32, R_MIPS_26…).

1. Le renvoie à l'étiquette est un relatif

Si l'étiquette est un relatif (les différentes instructions de branchement) alors il suffit simplement de la remplacer par la différence entre l'offset de la déclaration de l'étiquette (par rapport au début de la section) et l'offset de l'instruction qui l'utilise comme relatif. Il ne faut pas oublier de vérifier que cette valeur tient sur 18 bits et est divisible par 4.

2. Le renvoie à l'étiquette est un immédiat

Dans ce cas il faut procéder à une relocation. Un renvoie vers une étiquette utilisé en tant qu'immédiat consiste à une relocation de 16 bits (en l'occurrence les 16 bits de poids faible de l'adresse de l'étiquette). Il suffit alors d'ajouter à la table de relocation l'offset de l'instruction, le type de relocation : R_MIPS_LO16 ainsi que la section où est déclarée l'étiquette (si elle est déclarée dans le même fichier, on mettra son nom si ce n'est pas la cas).

3. Le renvoie à l'étiquette est un absolu

Lorsqu'un renvoie à une étiquette est utilisé comme opérande d'une instruction de type J c'est qu'il s'agit d'un absolu. Les absolu sont codé sur 26bits. On l'ajoute donc à la table de relocation en précisant cette fois ci qu'il s'agit d'une relocation de type R_MIPS_26.

4. La section data

Il est possible d'utiliser des renvoie à des étiquettes dans .data. Il s'agit souvent des étiquettes qui pointe vers un espace mémoire réservé dans la section .bss. Comme dans les autres cas si on a un renvoie à une étiquette on l'ajoute à la table de relocation en précisant cette fois qu'il s'agit d'une relocation de type R_MIPS_32.

5. Le cas des pseudo-instructions

Le cas des pseudo-instructions est géré de manière indépendante. En effet les pseudo-instructions sont transformées en 2 instructions (pour les instructions SW et LW). À la suite de cette transformation chacune des 2 instructions nécessitent une relocation de type différentes. Dans l'analyse syntaxique on identifie ces 2 cas au moment de la transformation des pseudo-instruction. Dans la fonction de remplissage de la table de relocation on identifie alors facilement la relocation de type R_MIPS_HI16 et celle de type R_MIPS_LO16.

6. Exemple de l'exécution de ce code

L'exécution du code suivant (figure 3) nous donne le résultat souhaité (figure 4).

```
# TEST_RETURN_CODE = PASS
#allons au ru
.set noeorder
.text
add $8,$9,$10
Lw $t0 , lunchtime
Lw $12, -200($7)
ADDI $t1,$t2,8
boucle :
add :
BeQ  $t0 , $t1 , byebye # test
    NOP
addi $t1 , $t1 , 1
J boucle
    NOP

byebye:
JAL boucl

.data
lunchtime:
test:
    .word 12
.asciiz "ils disent : \"au ruuu!\""
    .word menu

.bss

menu:
    .space 24
```

Figure 3 : Le code du fichier tests/miam_sujet.s modifié

```
[.rel.text]
Offset    Type        Value
4         R_MIPS_HI16 DATA
8         R_MIPS_LO16 DATA
32        R_MIPS_26   TEXT
40        R_MIPS_26   BOUCL

[.rel.data]
Offset    Type        Value
12        R_MIPS_32   BSS
```

Figure 4 : Le table de relocation du code de la figure

Un autre test est aussi effectué, il se situe dans tests/3livrable/test_relocation.s
Dans ce fichier on fait un branchement sur une étiquette située dans .bss, on obtient le résultat suivant (la table de relocation se situe dans le dossier récapitulatif : elle ne présente pas d'erreur) :

```
[nil@nil-pc MIPS]$ ./as-mips tests/3livrable/test_relocation.s
[ DEBUG :: src/main.c:main:167] Le code source contient 35 lignes.
[ DEBUG :: src/main.c:main:195] Il n'y a pas d'erreur de lexique dans le code source !.
[WARNING:: src/main.c:main:225] Il y a des erreurs de syntaxe dans le code source !.
**********  ERREUR  **********

Ligne : 15      Saut relatif à une étiquette qui n'est pas dans la section .text        MENU

******************************************************
```

Figure 5 : Erreur sur l'utilisation d'étiquette en tant que relatif