LARBI Adel, M1 STL
LARBI YOUCEF Mohamed Reda, M1 STL

RAPPORT DU PROJET SUR L OPTIMISATION D'UN CODE ASSEMBLEUR
COMPILATION AVANCEE

I- Introduction

Le but de ce projet est d'implémenter un programme permettant d'optimiser un code assembleur. Cette optimisation a donc pour but de réduire le temps d'exécution du code assembleur, en passant par des techniques tels que le calcul du graphe des bloc de bases, le calcul du graphe des dépendences entre les instructions, ou encore le renommage de registres.

Ce programme est implémenté en C++.

II- Fonctionnalitées implémentées

Nous avons implémentés, testé et vérifié manuellement toutes les fonctions à implémenter, à savoir :

1) Le calcul des blocs de base.
2) Le calcul des successeurs et prédecésseurs des bloc de bases.
3) Le calcul du CFG.
4) Le calcul des bloc dominants.
5) Le calcul des dépendences.
6) Le calcul du nombre de cycles.
7) Le calcul des registres USE et DEF, LiveIn et LiveOut.
8) Le calcul des registres DefLiveOut.
9) Le renommage de registres (Nous avons tout de même un doute sur le résultat obtenu, car nous n'avons pas vérifié manuellement les résultat pour cette fonction).

II- Programme de test

Le programme qui permettra de tester les optimisations sur un code assembleur devra, prendre en paramètre un fichier contenant le code assembleur et l'afficher, et réaliser, pour chaque bloc de base, les opérations suivantes :

1) Afficher le bloc de base.
2) Calculer le temps d'exécution du bloc sans aucune optimisation et l'afficher.
3) Renommer tout les registres qui peuvent être renommés, et afficher le bloc de base.
4) Réordonnancer le bloc de base, et afficher le nouveau bloc de base.
5) Recalculer le temps d'exécution, et l'afficher.
6) Afficher le gain du temps d'exécution.

III- Code assembleur de test

Nous pouvons tester notre programme avec n'importe quelle code assembleur. En ce qui nous concerne, nous avons testé avec le code assembleur du fichier *sample2.s* (fourni dans le compte rendu). Ce fichier contient le code suivant :

```
        .text
        .ent main
main:
(BB0)   lw $4, 0($6)
        lw $2, 0($4)
        add $5, $14, $2
        ori $10, $6, 0
        sw $5, 0($10)
        lw $2, -12($10)
        addi $5, $2, 4
        bne $5, $2, $L5
        nop

(BB1)   lw $4, 0($6)
        lw $2, 0($7)
        add $5, $4, $2
        sw $5, 0($6)
        addiu $12, $8, 2
        addiu $7, $12, 1
        bne $7, $0, $L5
        nop

(BB2)   sub $6, $0, $5
        sll $6, $3, 4
        addiu $5, $6, -2
        sw $15, 12($7)
        sw $10, -4($6)
$L5:
(BB3)   sub $8, $10, $15
        sll $10, $10, 4
        sw $8, 8($7)
        add $10, $8, $10
        sw $10, 12($7)
        jr $31
        nop

        .end main
        .set reorder
```

Qui n'est autre que le programme étudié en TD 2.

IV- Resultats

Nous obtenons, avec le code assembleur du fichier *sample2*, les resultats suivants :

Pour BB0 :
-Sans réordonnancement et renommage: 13 cycles
-Avec renommage et sans réordonnancement: 12 cycles
- Avec réordonnacement : 11 cycles

Pour BB1 :
-Sans réordonnancement et renommage: 10 cycles
-Avec renommage et sans réordonnancement: 10 cycles
- Avec réordonnacement : 8 cycles

Pour BB2 :
-Sans réordonnancement et renommage: 5 cycles
-Avec renommage et sans réordonnancement: 5 cycles
- Avec réordonnacement : 5 cycles

Pour BB3 :
-Sans réordonnancement et renommage: 7 cycles
-Avec renommage et sans réordonnancement: 7 cycles
- Avec réordonnacement : 7 cycles

Nous obtenons donc une amélioration du temps d'exécution sur certain bloc de bases, notamment lorsqu'il ont une taille «assez grand»

IV- Problèmes rencontrés

Comme dit ci-dessus, nous ne sommes pas sûrs du renommages des bloc de bases. On obtient bien une amélioration mais comme nous n'avons pas vérifié manuellement, nous ne pouvons pas nous assurer que le résultat est bon. Même si nous avons tout de même vérifié que nous obtenons les bons régistres disponibles pour le renommage.

Autre incertitude, celle dans la fonction de calcul des USE et DEF. Nous avons vu que l'instruction «nop» en deleyed slot est remplacé par l'instruction «add $0 $0, $0», nous devrions donc ajouter dans DEF le registre 0, seulement nous ne l'avons pas ajouté, car nous ne sommes pas sûrs du fait que nop soit forcément remplacée par cette instruction.