Compilation

TP 4C/5B : Compilateur simple - Partie 3

G. Iooss & C. Alias

Le but de ce TD/TP est de temporis-... euh... de faire enfin tourner sur Digmips les programmes générés par le compilateur simple vu les semaines précédentes.

Partie I – Compilateur simple et Digmips

Exercice 1 - Rectification du code généré

Manip.

- Dans l'hypothèse que vous n'aviez pas fini l'un des 2 TDs précédents, vous pouvez les finir avec les fichiers sources correspondant, jusqu'à être arrivé à générer du code assembleur Digmips.
- Faites tourner votre compilateur simple sur un programme comportant un tableau et, en utilisant l'exécutable `asm` [1], essayez d'obtenir le code machine. Quel est le problème ?

Vu que plusieurs bugs ont été corrigés depuis le dernier TP, récupérez les sources de ce TP et décompressez-les avant de faire vos modifications dessus.

- Proposez une solution pour réglez le problème sur les tableaux immédiatement et modifiez `parser.ypp` en conséquence. Faites tourner tout ça sur un exemple très simple avec tableau, jusqu'à obtenir le code machine.
- Proposez plusieurs solutions possibles pour s'en sortir, au cas où le problème évité précédemment surviendrait lors d'un calcul.

Exercice 2 - Affichage et bibliothèque d'exécution

À ce stade, on peut faire tourner le simulateur [2] sur quelques programmes simples en C. Le seul hic... c'est que l'on voit rien du tout (à part un `PC` qui se balade, on n'a pas de moyen d'avoir le contenu des registres ou de la pile directement sur le simulateur).

Du coup, il va falloir rajouter des instructions dans le code compilé pour demander au programme d'afficher des valeurs à l'écran. Pour rappel, l'instruction correspondante en code assembleur est :

- `st ri, [rX+255]` avec `X` pouvant valoir ce que l'on veut, affiche la valeur du registre `ri`.
- `ld ri, [rX+255]` (avec `X` ce que l'on veut) lit une valeur du clavier et la met dans `ri`.

La solution que l'on va explorer par la suite est de créer une *bibliothèque d'exécution*, ie une série de fonctions annexes dont le corps peut être constitué de code assembleur à insérer directement dans le code généré. Nous allons donc nous servir de cette bibliothèque pour gérer l'affichage et la multiplication.

Manip.

- [Preprocessing] Allez dans le dossier `"test/test_preprocess"` et exécutez la commande "`cpp main.c`". Quel est le résultat ? À quoi sert `#ifndef` et `#define` ? Que faudrait-il faire pour avoir du code qui pourrait être géré par le compilateur simple ?
- Retournez dans le dossier `"src"` et examinez les fichiers `main.cc` et `lexer.l`. Comment gère-t-on ce preprocessing ?

- [Code inliné] Retournez dans `"test/test_preprocess"` et complétez les trous dans `runtime.c`. Observez comment le code inliné est géré dans `parser.ypp`.
- Modifiez la fonction `mul` dans `CodeDigmips.cc` pour faire un appel de fonction à la fonction `mult`.
- Sauf bug qui m'ait échappé, le tout devrait pouvoir être transformé en code machine via `asm`, puis exécuté par `Digmips`. Si ce n'est pas le cas, tapez sur le TP-man [3].

1. fournit dans `bin` et `/home/calias/M1-Compilation/asm/...`
2. binaire dans `/home/calias/M1-Compilation/diglog/...`
3. Mais pas trop fort...

Partie II – (Bonus) Fonctions imbriquées

Exo 1 - (Échauffement) Lien d'accès et display

Considérons le programme (fonctionnel) comportant les fonctions imbriquées suivantes :

```
let funct1 i =
    let add2i k =
        let addi a =
         let k = (a+i) in
         k
        in
        addi (addi k)
    in
    let mult i j =
        (i*j)
    in
    mult (add2i i) i;;
```

Question :

- Quel est l'état de la pile (et le display) pour chaque fin de fonction ? Notamment, quel est la table de transition pour chacune de ces fins de fonction ?

Exo 2 - Fonctionnel et procédures imbriquées

Question :

- Comment faire pour passer des procédures en arguments/renvoyer des procédures ? Plus précisément, trouvez les informations nécessaires pour pouvoir appeler cette procédure, puis déduisez-en une structure contenant toutes ces informations et le prélude correspondant.
- Votre structure peut-elle gérer le programme suivant ? Si non, adaptez-la.

```
let toto i =
    let my_add k l =
        (k+l+i)
    in
    let f = (my_add i) in
    f;;
```

- Quel (gros) problème survient si une fonction essaye d'utiliser le résultat de `toto` ? (Indice : Essayer d'exécuter le programme correspondant en surveillant l'état de la pile).
- Quel est le critère pour détecter un tel cas ? Proposez une solution pour contourner le problème dans ce cas.

(Bonus du bonus) Si le TP n'est toujours pas terminé, essayez de conceptualiser la galère que sont des "Goto" hors d'une procédure imbriquée [4] [5].

4. Bon courage...
5. Pour en savoir plus, allez regarder "Modern Compiler Design" (ref : D.3.4 GRUN à la bibli), p 485 à 501