Rapport projet de TDL

Korantin Auguste Etienne Lebrun Maxime Arthaud

mai 2014

Table des matières

1 Introduction

Nous avons décidés de s'éloigner un peu du sujet, puisque nous n'avons pas utilisé EGG, mais une modification de ce dernier faite par Maxime (avec de nombreux ajouts intéressants, mais une syntaxe légèrement différente, et donc incompatible). De plus, nous avons souhaité non pas compiler le code seulement en TAM, mais aussi en x86.

Notre compilateur est modulaire et contient donc des « machines » capable de générer du code assembleur. Nous avons donc une machine TAM, et une machine x86.

Ainsi, nous pouvons compiler les programmes en assembleur x86, et les faire tourner sur nos propres machines !

Nous disposons aussi d'un petit système de préprocesseur permettant d'inclure d'autres fichiers.

Pour pouvoir s'amuser un peu, nous avons aussi développé une librairie TAM qui propose quelques fonctions utiles, pour afficher du texte... en appelant une simple fonction (au lieu de mettre du code assembleur). De même, nous disposons d'une petite librairie x86, développée entièrement par nos soins et qui effectue des appels systèmes (via une interruption processeur), afin de lire, d'écrire, de faire de l'allocation dynamique... Nous avons même codé une petite implémentation de la commande « cat » en MOC.

2 EGG

3 La machine TAM

4 La machine x86

La machine x86 est chargée de générer du code assembleur x86 (qui devra être compilé par nasm). Au final, elle est très proche de la machine TAM et je ne m'étendrais pas sur le sujet. Toutefois, un point différe complètement : là ou TAM est une machine à pile pure, en x86 nous devons gérer les registres.

4.1 Gestion des registres

Plusieurs choix se sont offerts à nous pour gérer les registres en x86 :

Tout d'abord, nous aurions pu tout mettre sur la pile, à la façon de TAM. Lors d'un calcul, il suffisait de faire comme TAM en effectuant des pop, de faire le calcul, et de tout remettre sur la pile. Ce système aurait été très simple, mais nous auriont toujours utilisé un ou deux registres, et notre but était d'aller un peu plus loin que la simple machine à pile.

A l'autre extrème, il aurait été possible dans l'idéal de stocker certaines variables uniquement dans des registres, sans jamais les mettre en mémoire, et d'utiliser tous les registres de manière intensive. C'est ce que font les vrais compilateurs, mais il s'agit de quelque chose d'extrêmement complexe.

Au final, nous avons retenu une solution qui est relativement simple, tout en tirant déjà plus partie des registres : les variables sont toutes stockées sur la pile, mais lors

d'un calcul tous les résultats intermédiaire pourront être stockés dans divers registres, et n'auront jamais besoin d'être mis sur la pile.

Pour cela, nous utilisont une pile qui retient les registres contenant les résultats des derniers calculs. Ainsi, un calcul verra ses résultats intermédiaires stockés dans différents registres. Cette solution marche plutôt bien, tire vraiment partie des registres et n'a pas été trop dure a mettre en oeuvre.

4.2 lib x86

Pour faciliter le travail, nous avons développé une petite bibliothèque qu'il suffit d'inclure, et qui se charge de générer un squelette de programme en assembleur, avec une dizaine de fonctions utile pour permettre à nos programme de faire des choses : lire, écrire, quitter, afficher du texte...

Son développement fût très intéressant, puisque nous avons tout fait nous même, sans faire appel a une bibliothèque tierce comme la libc. Pour communiquer avec le système d'exploitation, il suffit d'utiliser l'instruction « int » permettant de faire une interruption, et de mettre les bonnes données dans les registres, pour dire au système d'exploitation ce que l'on attend de lui.

Ainsi, pour afficher du texte nous avons une fonction print, qui fait elle-même appel a une fonction write, qui effectue un appel système :

```
int write(int fd, char* buf, int nbytes) {
    asm("
        ; write syscall
        mov eax, 4
        mov ebx, %fd
        mov ecx, %buf
        mov edx, %nbytes
        int 0x80 ; the result is returned into eax : perfect
    ");
}

void print(char *s) {
    write(1, s, strlen(s));
}
```

5 Tests

6 Conclusion

Ce projet aura été l'occasion d'effectuer un petit compilateur jouet, et nous aura permis de ce rendre compte que c'était une tâche assez longue, mais réalisable.

Toutefois, nous avons aussi mis le doigt sur le fait que s'il est simple de réaliser un petit compilateur qui fonctionne, arriver à lui faire générer du code rapide et optimisé doit être une tâche extrêmement difficile.

Le fait d'arriver à générer du x86 a donné un côté très amusant au projet, puisque nous pouvions lancer nos programmes sur nos machines, et même utiliser un debugger comme gdb pour comprendre certains bugs.

Arriver à effectuer des appels système pour avoir un programme qui fait de vraies actions sur nos ordinateurs était aussi particulièrement sympathique !