Semaine 3-4 (Compilateur-IR) - Projet IFT3150

Philippe Caron

29 août 2017

1 Objectif

Le but du compilateur-IR est assez direct : produire du code assembleur compilable avec gcc. Cependant, le code produit doit respecter les standard de C pour que les deux langages puissent être interfaçable. Ce qui implique :

— Appeler des fonctions avec les registres
— Aligner adéquatement la pile

2 Le compilateur-IR

Le compilateur-IR lit une liste d'instructions comportant au plus un paramètre, la question du *parsing* ici est complètement éliminée. Les instructions sont basées sur le principe d'une pile d'exécution. Bien entendu on pourrait utiliser la pile de l'ordinateur, mais cela produirait du code illisible, plutôt lent, mais surtout incroyable long. À tous les niveau, l'utilisation de registres est favorable à celle de la pile. De plus si l'objectif est un jour non seulement d'appeler du C, mais de pouvoir être appelé PAR du C, alors le code se doit de respecter une certaine norme.

2.1 Pile virtuelle

Pour utiliser les registre sans faire d'erreur, **Luna** utilise une pile virtuelle (`v_stack`). Sur cette pile, chaque case correspond à un registre précis. Il ne peut pas y avoir plus qu'un registre par case sur le stack, ceci réduit grandement le risque d'erreur et de conflit. Dans le cas de **Luna**, il y a 8 registres de calculs, donc chaque registre correspond à la place qui a son numéro (mod 8). En plus de la pile virtuelle, le compilateur-IR conserve la valeur de la taille de la pile, de manière à pouvoir la protéger lors d'un appel. Il conserve aussi la dernière position connue de la pile ; ceci lui permet de faire des ajustements relatifs (si la base n'est pas connue) ou des ajustement absolu dans l'autre cas.

3 Déroulement

Évidemment, lorsqu'on est rendus aussi près de la machine, il y a beaucoup d'accident, et la phrase «Segmentation fault (Core dumped)» revient souvent. Malgré tout, l'écriture du compilateur-IR s'est déroulée sans trop d'obstacle majeurs ; la plupart des problèmes était réglés rapidement et l'écrire général du compilateur avançait a un vitesse intéressante.

4 Faits-saillants

L'écriture du compilateur-IR fut un travail colossal, mais certains acquis valent plus peine d'être mentionnés que d'autres

4.1 Nombres à virgules

En commençant à écrire le compilateur IR, il est devenu évident que supporter les nombre à virgule demanderait un changement majeur. Au lieu d'être pris avec plus tard, la décision à été prise de commencer tout de suite en supportant les nombre à virgule, de sorte qu'il n'y aurait pas de mauvaise surprise plus tard.

4.2 Fonctions externes importantes

Afin d'améliorer la lisibilité du code et pour éviter les fragment important redondant, à partir du moment où les opérateur prennaient plus qu'environ 5 lignes assembleur, ils étaient incorporés au fichier utils.asm puis appelés.

4.2.1 utils.s

Puissance

L'opérateur de puissance en Lua est défini dans le langage lui-même il a donc fallu implémenter un algorithme en assembleur afin de calculer une puissance arbitraire d'une base arbitraire. Cette fonction est assez volumineuse et vaut mieux être définie dans la librairie que dans le code produite.

4.2.2 mem.s

Transfert

À chaque fois que des variables sont reféfinies, le compilateur fait appel à la fonction de transfert. C'est en fait une routine assembleur qui fait corresondre chaque valeur à un index en mémoire, en décant un index jusqu'à ce que le pointeur des valeurs atteinge la première case des destination.

4.2.3 array.s

Indexage

Les fonction d'indexage écrites en assembleur sont très fiable et permettent une gestion rapide et efficace des table. Le plus difficile a été d'écrire la fonction de redimensionnement dynamique des table lorsqu'on accède à un nouvel index, mais maintenant cette fonction fonctionne très bien et les tables sont une structure fiable malgé leur complexité

Chargement de table

La fonction de copie des tables est très importante car elle permet de charger une table depuis la pile. Cette implémentation a été priorisée car il est impossible de prévoir la taille d'une table quand le dernier élément est une fonction ou le stack d'argument variable. Cette fonction va simultanément définir la taille de la table produite.

4.2.4 special.s

L'indexage spécial est également un des succès agréables de cette étape. Grâce à cette fonction, le programme peut avoir accès à toute les sous-fonctions d'un type, sans devoir appeler sa fonction de librairie associée.

5 Défis

Certaines choses ont évidemment poser un plus défi qu'initialement envisagé. Un des plus gros défi a sans doute été la gestion des tables. Il y a tellement de manière de définir des éléments de table.

```
x = { allo = 1, ["bonjour"] = 2, n = 3; [1] = 2, 3, 4}
x["b"]= x.allo + x[3]
```

Et la contrainte auto-imposée d'avoir des tables à double format a commencer à peser lourd. Au final cependant tout est rentré dans l'ordre et le compilateur-IR rempli exactement la fonction qu'on attend de lui. Le seul problème est le retard pris, cela risque d'impacter sur le temps à dédier au GC.

6 Conclusion

Cette étape du projet quoi que frustrante par moment s'est vraiment déroulée parfaitement en y repensant. Tous les objectifs ont été atteint, et le code produit est compatible avec gcc, l'appel de fonction C fonctionne.