

Sylvain BALAKRISHNAN

Jules BRANDSTAEDT

Nicolas CIPOLLA

Maxime COLLETTE

## **Rapport : Projet de Compilation**



## **Table des matières :**

- Fonctionnalités et tests 1.
- Points d'intérêts 3.
- Explications additionnelles du projet 5.

## **Fonctionnalités et tests :**

Le compilateur se nomme CcS.

`./CcS -i file.SoS -o file.s` lance le compilateur sur le fichier `file.SoS` et stocke le code en mips dans `file.s`.

Les fonctionnalités implantés sont pour la plupart accompagnées de tests. Si c'est le cas, les commandes en italiques permettent de les effectuer.

Toutes les fonctionnalités ont été implémentées, à l'exception des appels de fonction qui se font différemment (cf point d'intérêt) mais fonctionnent tout de même.

### **Points manquants:**

- Optimisation de code

L'optimisation est un des points intéressant et recherché dans un compilateur malheureusement nous n'avons pas eu le temps de traiter cette partie.

- Conservation des registres `$s`

Les registres se doivent d'être conservés entre les appels de fonction. La conservation au sein même du code SoS n'est pas un

problème : les conventions ne sont juste pas les mêmes. Mais cette conservation n'est pas non plus présente durant l'appel d'un code SoS par un programme tiers ce qui peut poser problème. Cela aurait pu être résolu par exemple en mettant tous les registres sur la pile au début du code et en les dépliant à la fin mais ce n'a pas été fait notamment pour des raisons de simplification du code produit. Mais il est à noter que le registre 31 \$ra, lui, est bien conservé (sur la pile) pour permettre les appels de fonctions imbriquées.

## Points d'intérêts :

### Grammaire :

- La grammaire à été changée au niveau des déclarations de fonction. Le nombre d'arguments doit être connu à la compilation et donc être indiqué au niveau de la déclaration dans les parenthèses qui était vide avant :

*fun(3) {...* déclaration de la fonction qui a 3 arguments.

Cela permet de générer le code plus facilement. notamment pour le *for i in* où *i* prend successivement les valeurs des arguments. Cela permet également de connaître la place des arguments dans la pile (pour pouvoir l'utiliser pour autre chose; par exemple pour les opérations arithmétiques). Mais cela aurait pu être faisable notamment en mettant tout les argument sur la pile puis le nombre d'argument et enfin en mettant le pointeur de pile en haut (pile "vide" au début de la fonction, et les argument sont sous la pile), ainsi on aurait une pile homogène à chaque début de fonction (indépendant du nombre d'argument).

Notre pile est homogène car le nombre d'arguments est connu. Les arguments sont "au dessus" de la pile au début de la fonction

- De plus, des changements ont été faits au niveau des opérations entières. Nous passons de : *somme\_entiere* et *produit\_entier* à un unique *operateur\_entier* qui gèrent les

produits et les sommes grâce aux priorités des opérateurs et des parenthèses. Nous gérons le bon déroulé des opérations grâce à l'utilisation pile avec *setup\_opérateur\_entier*, qui est uniquement dédié à ce rôle.

### Code MIPS:

- En ce qui concerne le code MIPS, nous avons implémenté quelques fonctions pour faciliter la lecture du code : *strtoint*, *strlen* (en 2 versions différentes avec l'argument contenu dans *\$a0* et *\$a1*), *strcompare*, *strconcat* et *intostr*.

- Il existe un fichier qui écrivait les instructions MIPS en binaire, car au début, nous avons cru que le MIPS s'écrivait ainsi et non en ASCII. Nous avons écrit différentes fonctions pour faciliter l'écriture des différentes instructions binaires.

## **Explications additionnelles du projet :**

Toutes les variables utilisateurs sont en mode chaîne de caractères, ducoup il faut les convertir pour les utilisées comme entier, ce qui est coûteux. La traduction est temporaire, il faut la refaire à chaque fois. Il existe cependant des variables internes (non visibles par l'utilisateur (lui même) qui elles peuvent être directement des entiers : `_for1` ).

Pour les boucles for il fallait récupérer le counter pc pour pouvoir faire des jump dessus. Pour cela on fait un jump and link sur place pour récupérer le pc dans le registre \$ra(il faut bien sûr stocker l'ancien \$ra pour ne pas l'écraser).

La traduction entre le code intermédiaire et le mips n'est pas forcément idéal du à une mauvaise organisation du code intermédiaire. `$5 <- $7` en code intermédiaire devient `move $s0,$7` et `move $5,$s0`. Cela est dû au fait que les instructions en code intermédiaire sont trop vastes: il y a trop de cas spécifiques.