

Neau Guillaume Michael Rigaud | IETA 2017 |

VM\_C

* Introduction

Le projet Vm\_C proposé par Monsieur Le Lann nous amène à reproduire le comportement du microprocesseur MIPS-X. Nous avons choisis de lui adjoindre un compilateur afin que l’on puisse utiliser son jeu d’instruction plus facilement.

* Comportement de la Machine virtuelle

*Cahier des charges*:

La machine virtuelle doit pouvoir exécuter les 19 instructions génériques du MIPS-X.

Pour cela elle a disposition un tableau de registre et un tableau de Data.

Elle prends en entrée un tableau de commandes contenant a chaque case du tableau une instruction a réaliser sous la forme décrite dans la partie Compilateur.

*Fonctionnement :*

Les instructions sont réalisés les unes après les autres comme cela a été demandé dans le sujet. Mais nous avons réalisé quelques choix.

Pour la fonction jump :

Nous avons choisi de ne pas enregistrer l’état suivant dans un registre car quitte a rendre les programmes verbeux il est possible de réaliser toutes les actions imaginables, notamment le retour à cette étape.

*Informations affichés en sorti :*

A la fin de l’exécution du programme plusieurs donnés sont affichées dans le terminal (si l’option -o n’a pas été activé).

Toutes les instructions suivit de l’état des registres aprés chaque instruction est affiché. L’état des registres est affiché d’abord en hexadécimal puis en décimal entre parenthèse. De cette façon il est possible de suivre l’évolution des registres au cour du programme.

Enfin nous avons affiché le temps par instructions. Pour cela nous avons déterminé le temps total nécessaire pour exécuter le programme que nous avons divisé par le nombre d’instructions.

*Comportements supplémentaires :*

Pour rendre notre programme plus facile d’utlisation nous y avons ajouté des options.

Dans un premier temps il possible d’afficher l’aide avec l’option -h

Ensuite il est possible de préciser le fichier qui contient le programme avec l’option -i. Ou le fichier de sorti avec -o.

Enfin, on peut enregistrer l’état des registres à la fin de l’exécution de la machine pour retrouver cet état lors d’une autre application. Il faut pour cela faire -e. Puis on peut récupérer cet état avec -l.

* Compilateur

# Cahier des charges

Le compilateur doit pouvoir compiler les 19 instructions génériques du MIPS-X en binaire.

Le format de compilation doit respecter la nomenclature d’instruction du microprocesseur :

5 bit opcode – 5 bits registre r1 – 1 bit pour le flag – 16 bits de paramètre – 5 bits registre r2

# Fonctionnement

Le compilateur scan un fichier « entrée.txt » et stock les mots dans une pile qu’il va consommer. Les séparateurs sont les espaces ainsi que les retours à la ligne.

Les mots sont séparés en quatre catégories : Opérandes, Arguments, label et flags.

Etant donné la syntaxe très rudimentaire : Un opérande et forcément suivit d’un argument et un label d’un flag.

Dès lors la FSM du compilateur fonctionne comme suit :

Tant que Mot!=’ top’

If mot == Opérande

Décompose Argument

Construit le binaire Opérande+argument

If mot == Label

Retient le nom et la position du label

Return un code ‘nop’

Fin if,if

Mot suivant

Fin tant que

Si un Opérande n’est pas reconnus il est interprété comme étant un ‘nop’. La différence entre un label souhaité et une erreur de syntaxe est le flag ‘\_’ suivant un label proprement déclaré.

# Syntaxe générale

Nous avons souhaité coller au maximum aux instructions tel que décrit dans la datasheet du MIPS-X.

1. Début de programme

Pas de balise de début

1. Syntaxe générique

<Opérande> <espace><Arg1><,><Arg2><,><Arg3>

Exemple : add r1,0,r2

1. Syntaxte label

<Label+’ :’><espace><Flag>

Par défaut le flag est : \_

Exemple : fact: *\_*

Le compilateur ne fait pas de reporting d’erreur et ne préviens donc pas les boucles infinis où erreur de programmation qui pourraient endommager la machine virtuelle.

# Résultat

Nous avons ainsi crée un nouveau langage très proche de l’assembleur et le plus intuitif possible permettant de générer des instructions pour un processeur MIPS-X. Nous avons, pour vérifier le bon fonctionnement de notre compilateur, testé différents jeux d’instruction : différents jeux de load/add/sub/mult et un programme plus complet et complexe : facto.txt.

Cependant, le fonctionnement et la flexibilité des codes peuvent être grandement améliorés en implémentant les standards de compilation automatique vue dans une autre UV.

* Annexe

# Facto.txt

add NAN,10,r1

add r1,0,r2

fact: *\_*

sub r2,1,r2

braz r2,end:

mult r2,r1,r1

jmp fact:

end: \_

stop

En vert : Opérande

En violet : Label

En rouge : Flag

En noir : Argument

En dorée : la virgule, séparateur d’arguments