

# Compilation – TP 11 : De RETROLIX à MIPS

Université Paris Diderot – Master 1

(2014-2015)

Cette feuille de TP vous donne les étapes à suivre pour traduire RETROLIX et MIPS. Il s'agit de compléter le module `RetrolixToMIPS`.

Le code source correspondant à ces travaux pratiques se trouve sur le GIT, dont on rappelle l'URL :

`http://moule.informatique.univ-paris-diderot.fr:8080/Yann/compilation-m1`

On rappelle que vous devez faire des *commits* réguliers (à chaque modification de votre code) pour que nous puissions suivre votre avancement.

## 1 Structure d'un programme MIPS pour GCC

La production d'un fichier assembleur compatible avec le compilateur GCC nécessite le respect des contraintes suivantes :

- les conventions d'appel doivent être respectées;
- le point d'entrée du programme doit être une fonction `main`;
- les variables globales doivent être déclarées.

### Exercice 1 (Structuration du programme)

1. Complétez la définition de `main`. On doit s'assurer que cette fonction initialise correctement les variables globales du programme.
2. Complétez la définition de code. Le code compilé du programme doit contenir l'ensemble des définitions du programme source ainsi que le code du `main`.

□

## 2 La pile

Dans cette dernière passe, on doit expliciter les opérations d'extension, de réduction, de lecture et d'écriture qui s'appliquent sur la pile du programme. Il s'agit de mettre à jour le registre `sp` et de l'utiliser comme base pour calculer la position des variables locales en mémoire.

### Exercice 2 (Implémentation des routines de modification de la pile)

1. Comment doivent être implémentées les extensions et les réductions de la pile ? Complétez les fonctions `allocate_stack_frame` et `free_stack_frame` en conséquence.
2. Sachant que les variables locales à stocker dans la pile sont représentées par la liste `locals`, choisissez une façon de positionner chaque variable dans le bloc d'activation. Complétez la fonction `variable_address` en conséquence.

□

### 3 Compilation des instructions

À l'aide des primitives définies dans les deux exercices précédents, vous devriez pour compléter la fonction de compilation des instructions de RETROLIX vers MIPS. Vous pouvez ignorer l'instruction `TailCall` car elle sera l'objet du TP suivant.

Pour implémenter les fonctions `block_create`, `block_get` et `block_set`, on va utiliser un petit environnement d'exécution qui implémente ces fonctions en C dans un fichier nommé `runtime.c`.

#### Exercice 3 (Compilation des instructions)

1. Implémentez les fonctions `block_create`, `block_get` et `block_set` (dans le fichier `runtime.c`).
2. Implémentez une fonction `print_int` qui affiche un entier et saute une ligne (dans le fichier `runtime.c`).
3. Complétez la fonction `instruction` du module `RetrolixToMIPS` en ne traitant pas le cas `TailCall`. N'oubliez pas d'utiliser `load_rvalue` pour éviter toute explosion du nombre de cas à traiter.

□

### 4 Comment tester le code compilé ?

Pour tester votre code compilé, il faut se préparer un environnement d'émulation d'une machine Linux-MIPS32 comme suit :

1. Installer `qemu` sur votre système et s'assurer que `qemu-system-mips` est bien accessible.
2. Téléchargez les images `qemu` suivantes :
  - [https://people.debian.org/~aurel32/qemu/mips/debian\\_wheezy\\_mips\\_standard.qcow2](https://people.debian.org/~aurel32/qemu/mips/debian_wheezy_mips_standard.qcow2)
  - <https://people.debian.org/~aurel32/qemu/mips/vmlinux-3.2.0-4-4kc-malta>
3. Lancez la commande suivante :

```
qemu-system-mips \  
-M malta -kernel vmlinux-3.2.0-4-4kc-malta \  
-hda debian_wheezy_mips_standard.qcow2 \  
-append "root=/dev/sda1 console=tty0" \  
-net user,hostfwd=tcp::10022-:22 -net nic
```

4. Connectez-vous en `root` (mot de passe : `root`) et installez `gcc` avec la commande `apt-get install gcc`.
5. Copiez votre `runtime.c` sur la machine à l'aide de la commande :

```
scp -P 10022 runtime.c root@localhost:
```

6. Compilez votre `runtime.c` sur la machine à l'aide la commande :

```
gcc -Wall -c runtime.c
```

Ensuite, compiler puis exécuter un programme `prog.fopix` sur la machine émulée peut se faire à l'aide d'une unique commande :

```
../flap --gcc true -s fopix -t mips -! retrolix prog.fopix && \  
scp -P 10022 prog.mips root@localhost:prog.S && \  
ssh -p 10022 root:root@localhost 'gcc -o prog runtime.o prog.S && ./prog'
```

### 5 Comment débbuguer le code compilé ?

Pour débbuguer le code compilé, il faut commencer par installer `gdb` sur la machine émulée à l'aide de la commande `apt-get install gdb`.

Puis, on lance le programme dans `gdb` à l'aide de la commande :

```
% gdb ./prog
```

On peut poser un point d'arrêt sur le `main` en faisant :

```
(gdb) b main
```

Puis, lancez le programme à l'aide de la commande `run` (ou simplement `r`).

On active le désassemblage automatique à l'aide de la commande :

```
(gdb) set disassemble-next-instruction on
```

On peut voir la valeur d'un registre à l'aide la commande :

```
(gdb) info registers sp
```

Enfin, on avance instruction par instruction dans l'exécution, à l'aide de la commande `stepi` ; on saute un appel de fonction sans rentrer dedans à l'aide de la commande `nexti` ; et on continue l'exécution jusqu'au prochain point d'arrêt à l'aide de la commande `cont`.

Lisez le manuel de `gdb` pour plus de renseignements.