

Compilation — TD 8

V. Balat, P. Letouzey, G. Manzonetto

2008–2009

Le but de ce TD est d'écrire une fonction

```
munch : Ir.stm -> Assem.instr list
```

qui traduit un programme en langage intermédiaire en une liste d'instructions en assembleur.

1. Écrivez une paire de fonctions

```
emit : Assem.instr -> unit
getresult : unit -> Assem.instr list
```

telles que `emit` ajoute une instruction à une liste globale d'instructions, et `getresult` retourne la liste complète stockée.

2. Écrivez une fonction

```
munch_exp : Ir.exp -> Assem.temp
```

qui utilise `emit` pour stocker dans la liste globale une suite d'instructions qui calcule le résultat de la r-valeur passée en argument et stocke le résultat dans un temporaire. Vous devrez bien-sûr vérifier que `munch_exp` est capable de traduire toute r-valeur produite par votre générateur de code temporaire.

3. Écrivez une fonction

```
munch_stm : Ir.stm -> unit
```

qui stocke dans la liste globale une suite d'instructions dont l'effet est le même que celui du code intermédiaire passé en argument. Assurez-vous que cette fonction est capable de traduire tout résultat de votre générateur de code intermédiaire.

4. Déduisez-en une définition de `munch`.

5. Écrivez une fonction

```
code_for_fragment :
  Ir.stm list * Frame.frame -> Assem.instr list
```

qui ajoute un prologue et un épilogue au code généré pour une fonction. Intégrez cette fonction au `main`, et testez votre programme.

6. Même s'il est correct, votre programme génère probablement du code très naïf dans certains cas (notamment dans le cas de certains `MOVE` très communs lors d'accès aux variables). Une des techniques pour améliorer le code généré consiste à ajouter des cas plus complexes aux fonctions `munch_exp` et `munch_stm`. N'hésitez pas à vous faire plaisir.