POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

LOG3210

Travail pratique 6

Génération de code machine

Hiver 2017

Professeur : Ettore Merlo

Chargés de laboratoires :

Nicolas Cloutier

Alexandre Pereira

1. Introduction

Enfin ! Après plusieurs étapes intermédiaires, vous pouvez finalement générer du code machine ! Dans ce dernier laboratoire, nous allons nous concentrer à convertir un bloc de code de base comme au précédent TP en du code machine. Si vous réussissez votre travail, vous pourrez utiliser le simulateur afin de calculer correctement un élément de la suite de Fibonacci.

Le langage machine que vous avez à supporter est celui décrit à la section **8.2.1** (p. **512**). Cependant, vous n'avez pas à supporter toutes ces commandes puisque vous n'avez pas besoin de gérer les tableaux et le flux de contrôle. Vous aurez surtout besoin des « LD », « ST » et « OP ». Bref, simplement ce qui est capable de supporter n'importer quel bloc de base valide. Pour effectuer vos traductions, vous pouvez vous baser sur la figure **8.20** à la page **561**.

2. Travail à réaliser :

Pour votre TP, vous devrez principalement écrire le visiteur qui générera le code machine. Tenez compte des particularités suivantes :

- Vous avez un maximum de 256 adresses mémoires disponibles. Mais on ne va jamais vous fournir des blocs qui risquent de dépasser la limite.
- Vous devez supporter un nombre variable de registres.
- Comme décrit dans la description du langage, vous ne pouvez pas utiliser les variables ailleurs que dans les « LD » et dans les « ST », vous allez devoir transférer vos valeurs dans vos registres et vice-versa.
- De plus, votre fichier d'entrée contient du code intermédiaire et des informations sur les variables vives comme calculées au dernier TP. Regardez l'exemple fourni.

On vous donne aussi deux fichiers « .code » qui sont deux blocs utilisés pour le calcul de Fibonacci. Ces deux blocs doivent être transformés en code intermédiaire et les variables vives doivent être calculées pour chaque bloc. Vous pouvez utiliser vos anciens travaux et la librairie fournit pour calculer ces informations. (Vous pouvez aussi utiliser l'optimisation que vous avez faîtes!). Ceci est une petite étape intermédiaire pour générer le code de Fibonacci[1]. Vous pouvez aussi effectuer ce calcul manuellement si vos algorithmes ne fonctionnent pas correctement.

Une fois le tout créé, ajoutez deux fichiers « .ci » dans le dossier « data » pour générer le code machine de ces blocs. Finalement, vous pouvez manuellement insérer ces deux blocs dans le fichier « examples/fibb.asm » présent dans le simulateur. Lancez le simulateur et si votre code est correct, vous pourrez calculer n'importe quel nombre de la suite de Fibonacci!

On vous demande donc de :

- Créer le visiteur.
- Générer le code machine pour le code de Fibonacci pour trois registres et pour cinq registres. Laissez les deux fichiers à la racine de votre projet.

[1] https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Fibonacci_Number_Program#Exponentiation_by_squaring

- Écrire un court rapport qui explique les différences entre le code généré pour les nombres de registres différents et l'utilité qu'ont eue les variables vives dans ces situations.

3. Simulateur

Le simulateur fourni a été écrit en Python 3.5, alors vous devrez utiliser la commande « python3.5 run_tests.py » pour tester tous les fichiers « .asm » présents dans le dossiers « examples ».

Vous avez besoin des librairies externes « arpeggio » et « numpy ». Si elles ne sont pas installées, vous pouvez les ajouter avec « pip install --user **nom_de_la_librairie** ».

Au début la méthode « simulate » du fichier « simulator.py », l'environnement est créé avec les contraintes du nombre de registre, c'est ici que vous pouvez le changer pour vos expériences.

Remarquez que ce simulateur a été écrit encore avec un parseur de grammaire ! Il y a le fichier « .peg » qui décrit la grammaire et un ensemble de visiteurs qui prépare et effectue la simulation. Vous avez plusieurs alternatives possibles si vous désirez créer des parseurs !

5. Fichiers fournis

- `LangageH17.jjt`
- `compile.sh`.
- `test_files.sh`: Ce fichier fait rouler le parseur sur les fichiers d'exemple, et génère dans res les fichiers contenant le code machine.
- `./data`: Contient les fichiers de test. Chaque fichier de test est séparé en deux parties. Une première contient les variables vives pour chaque instruction suivie d'un bloc de code intermédiaire.
- `Makefile` : Utilisez les commandes `compile` pour compiler et pour tester utilisez simplement `test`
- `./ast` : contient les classes java représentant les nœuds de l'AST
- `PrintMachineCodeVisitor.java` : visiteur de la grammaire, permet d'effectuer l'impression du code optimisé.
- `MachineCodeSimulator.zip` : Librairie pour la simulation du code machine.
    - `examples` contient les tests exécutés
    - `run_tests.py` est le script à lancer pour les tests.

6. Barème

- 15 points pour l'implémentation
    - 8 points pour la génération de code machine.
    - 5 points la réutilisation des registres. C'est-à-dire réutiliser une valeur chargée en registre au lieu de toujours la recharger.
    - 2 points pour l'utilisation des variables vives dans l'allocation de registres.
- 3 points pour le bon fonctionnement des deux programmes de Fibonacci.
- 2 points pour le rapport.
- -50% par jour de retard.
- -10% pour non-respect des consignes de remise.

Le tout pour un total de 20 points.

7. Remise

Remettez dans un fichier compressé nommé **TP6_matricule1_matricule2.zip** contenant simplement le dossier du projet fourni ainsi qu'un rapport au format PDF avec vos réponses pour les questions. Faites un « make clean » avant la remise. L'échéance pour la remise est le dimanche 16 avril 2017 à 23h55.