Projet Génie Logiciel

Compilateur Decac

Manuel de conception

Équipe GL2

Étudiants : Élina Houdé, Julien Pinto Da Fonseca,

Léa Solo Kwan, Yin Xu

Année 2019/2020

Sommaire

Introduction 3

1. Architecture 4

1.1. Gestion des erreurs 4

1.2. Codegen 4

1.2.1. RegisterManager 5

1.2.2. LabelManager 5

1.2.3. ErrorLabelManager 5

1.2.4. StackManager 5

2. Spécifications sur le code 7

3. Algorithmes et structures de données 9

Introduction

Ce manuel de conception s'adresse à tout développeur qui souhaiterait maintenir et/ou faire évoluer le compilateur. Celui-ci est essentiellement composé de la description de l'organisation générale de l'implémentation.

1. Architecture

1.1. Gestion des erreurs

Afin de simplifier la gestion des messages d'erreurs et la maintenabilité du code, il a été décidé de créer une classe dont le rôle est la centralisation des différents messages d'erreur pouvant être retournés à l'utilisateur. De ce fait, lorsqu'une erreur doit être déclenchée au sein du code, le message d'erreur approprié provenant de la classe statique ***ErrorMessages*** est passé en paramètre de l'erreur.

Chemin depuis la racine du projet :

src/main/java/fr/ensimag/deca/ErrorMessages.java

1.2. Codegen

Afin de mener à bien l'implémentation des différentes méthodes entrant dans le cadre de l'étape de génération de code, plusieurs classes personnalisées ont dû être créées. On retrouvera notamment les classes suivantes :

- ***RegisterManager*** : permet de gérer tout ce qui touche aux registres ;
- ***StackManager*** : permet de gérer tout ce qui touche à la pile ;
- ***LabelManager*** : permet de gérer la génération des labels ;
- ***LabelType*** : énumération composée des différents types de label ;
- ***ErrorLabelManager*** : permet de gérer les labels pour les erreurs concernant l'exécution du code. A la fin de la génération du code principal d'un programme, cette classe permet également de générer le code assembleur permettant la gestion de chaque erreur du programme ;
- ***ErrorLabelType*** : énumération composée des différents types de label d'erreur.

Chemin depuis la racine du projet :

src/main/java/fr/ensimag/deca/codegen/

1.2.1. RegisterManager

Cette classe comprend les méthodes publiques suivantes :

- ***isTaken****(int regNumber)* : renvoie un booléen pour indiquer si le registre passé en paramètre est utilisé (true) ou libre (false) ;
- ***take****(int regNumber)* : réserve le registre qui est passé en paramètre ;
- ***free****(int regNumber)* : libère le registre passé en paramètre ;
- ***nextAvailable****()* : renvoie le numéro du premier registre disponible ;
- ***getSize****()* : fonction qui renvoie le nombre total de registre.

1.2.2. LabelManager

Cette classe comprend les méthodes publiques suivantes :

- ***getLabelValue****(LabelType label)* : renvoie l'entier associé au label passé en paramètre ;
- ***incrLabelValue****(LabelType label)* : incrémente de un la valeur associée au label passé en paramètre.

1.2.3. ErrorLabelManager

Cette classe comprend les méthodes publiques suivantes :

- ***addError****(ErrorLabelType error)* : met à true le booléen associé à l'erreur passée en paramètre pour indiquer qu'elle devra être gérée ;
- ***errorLabelName****(ErrorLabelType error)* : retourne une chaîne de caractère correspondant au nom associé à l'erreur passée en paramètre. Ce nom est notamment utilisé comme label pour les instructions de saut ;
- ***printErrors****(DecacCompiler compiler)* : pour chaque erreur signalée précedemment par la fonction *addError(error)* on demande son affichage grâce à la fonction suivante ;
- ***printError****(DecacCompiler compiler, ErrorLabelType error)* : ajoute le code assembleur pour gérer l'erreur passée en paramètre.

1.2.4. StackManager

Cette classe comprend les méthodes publiques suivantes :

- ***getGB()*** : renvoie le numéro associé à l'adresse de la prochaine variable globale (adresse de la forme x(GB) avec x un entier) ;
- ***getLB()*** : renvoie le numéro associé à l'adresse de la prochaine variable locale (adresse de la forme x(LB) avec x un entier) ;
- ***incrGB()*** : ajoute 1 au numéro associé à l'adresse de la prochaine variable globale (adresse de la forme x(GB) avec x un entier) ;
- ***incrLB()*** : ajoute 1 au numéro associé à l'adresse de la prochaine variable locale (adresse de la forme x(LB) avec x un entier).

2. Spécifications sur le code

Afin de déterminer rapidement le but de chacune des méthodes d'une classe, celles-ci respectent les conventions de nommage suivantes concernant la génération de code assembleur :

- ***codeGenInst****(DecacCompiler compiler)* : méthode réservant les registres nécessaires pour la génération du code assembleur d'une instruction produite grâce à l'appel à la fonction suivante ;
- ***codeGenInst****(DecacCompiler compiler, GPRegister register)* : méthode qui génère le code assembleur correspondant à une instruction et stocke le résultat de l'instruction dans le registre *register* ;
- **codeGenInstArith***(DecacCompiler compiler, GPRegister register1, GPRegister register2)* : méthode similaire à *codeGenInst* mais spécialisé pour les expressions binaires arithmétiques, car celle-ci nécessite l'utilisation de deux registres ;
- ***codeGenPrint*** : méthode qui affiche le résultat d'une expression, après avoir générer, si nécessaire, le code assembleur correspondant à cette instruction avec la méthode *codeGenInst* ;
- ***codeGenCMP****(DecacCompiler compiler, Label label, boolean reverse)* : méthode vérifiant si une condition est respectée lorsque *reverse* vaut false, et si elle n'est pas respecté lorsque *reverse* vaut true, i.e. réalisant la comparaison adéquate entre deux expressions et effectuant un saut conditionnel à un label donné ;
- ***codeGenError****(DecacCompiler compiler)* : méthode générer la condition de saut en cas d'erreur et signalant au ErrorLabelManager qu'il faudra générer le code permettant la gestion de cette erreur à la fin du fichier assembleur.

Concernant les vérifications contextuelles, les méthodes respectent les conventions de nommage suivantes :

- ***verifyClass*** : méthode vérifiant contextuellement que la déclaration d'une classe est correcte lors de la passe 1 ;

- ***verifyXXXMembers*** : méthodes vérifiant contextuellement que l'élément de type XXX est valide lors de la passe 2. L'élément de type XXX peut représenter un champ, une méthode ou les paramètres d'une méthode ;
- ***verifyXXXBody*** : méthodes vérifiant contextuellement l'élément de type XXX lors de la passe 3. L'élément de type XXX peut représenter un champ, une méthode ou les paramètres d'une méthode ;
- ***verify(List)Inst*** : méthode qui vérifie contextuellement une instruction ou une liste d'instruction lors de la passe 3 ;
- ***verifyExpr*** : méthode qui vérifie contextuellement une expression lors de la passe 3 ;
- ***verifyYYY*** : méthode qui vérifie vérifie contextuellement un élément de type YYY lors de la passe 3. Cette méthode concerne toutes les éléments pour lesquels un des trois derniers types de méthodes ne peut être utilisé.

3. Algorithmes et structures de données

Afin de correspondre aux spécifications, les attributs suivants ont été ajoutés à la classe *DecacCompiler* :

- *RegisterManager registerManager ;*
- *StackManager stackManager ;*
- *LabelManager labelManager ;*
- *ErrorLabelManager errorLabelManager.*