代码生成

梁宇钦 151180076

2018.07.07

1. 功能实现

1.1 功能实现情况

实验实现了基本要求以及选择要求的所有要求。对于给出的所有基础例子和选做例子都能正确生成 MIPS 代码，并且生成的中间代码进行了优化，具有较高的效率。

1.2 功能实现方法

在中间代码的基础上进行代码生成，主要解决两个问题：指令选择和寄存器分配。

1.2.1 指令选择

由于中间代码是比较低级的，所以它跟 MIPS 代码几乎对应，因此指令选择十分简单。基本上就是直接对中间代码进行对照翻译就行了。

1.2.2 寄存器分配

寄存器分配是本次实验的难点。

在该实现中，我采用寄存器描述以及变量描述符两个数据结构，对变量以及寄存器进行记录和跟踪。

由于全局的变量分析比较困难，在该实现中，采用较容易实现的局部的变量跟踪。

再进行寄存器分配时，因为没有进行全局的活跃变量分析，所以该实现采用一个类似于老化的方法来进行寄存器的分配。即如果一个寄存器很久没有被使用了，那么它就很可能被覆盖掉。如果一个寄存刚刚被使用过，那么它就不会被选择。

2. 编译和测试

这部分内容在文件“README.MD”中有详细说明, 这里仅做简要说明。

2.1 目录树

```
.
├── 3rdparty
├── intermedCode.c
```

```
├── intermedCode.h
├── lexical.l
├── Makefile
├── objCode.c
├── objCode.h
├── ObjCodeOut.d
├── README.md
├── runtest.sh
├── sample.d
├── semantic.c
├── semantic.h
├── SymbolTab.c
├── SymbolTab.h
├── syntax.output
├── syntaxtree.c
├── syntaxtree.h
└── syntax.y
```

本次实验的所有内容都在 CodeGen.d 目录下面。主要文件包括 lexical.l, syntax.y, syntaxtree.h, syntaxtree.c , SymbolTab.c, SymbolTab.h, semantic.h, semantic.c, IntermedCode.c, IntermedCode.h 和 objCode.h, objCode.c 等文件。其中 lexical.l 是词法文件, 用 flex 编译；Syntax.y 是文法文件, 用 bison 编译；syntaxtree.h 和 syntaxtree.c 是语法树的数据结构文件；SymbolTab.c 和 SymbolTab.h 是符号表数据结构文件；semantic.h 和 semantic.c 是语义分析文件；IntermedCode.c 和 IntermedCode.h 是中间代码生成文件；objCode.h 和 objCode.c 是代码生成相关文件；sample.d 下面是样例。

2.2 编译

直接在终端输入命令:

```
make
```

就可以了编译了。

2.3 测试

2.3.1 测试已经提供的样例

要测试实验指导提供的样例, 也是非常简单的. 直接运行脚本 “runtests.sh” 即可. 建议采用以下的命令运行脚本:

```
sh runtests.sh
```

其结果会会打印屏幕上的同时也会存到 ObjCodeOuput.d 目录下面。

2.3.1 测试其他未提供的样例

命令格式如下:

```
./parser testfilename ObjCodeOutputfilename
```

其结果会打印屏幕上的同时也会存到 ObjCodeOuput.d 目录下面。