编译原理实验四实验报告

171860607

白晋斌

810594956@qq.com

任务号:14

目录

实验要求

实验四任务是在词法分析、语法分析、语义分析和中间代码生成程序的基础上,将 C--源代码翻译为 MIPS32 指令序列(可以包含伪指令),并在 SPIM Simulator 上运行.

“正确”是指该汇编代码在 SPIM Simulator 上运行结果是正确的,因此,以下几个方面不属于检查范围:

1)寄存器的使用与指派可以不必遵循 MIPS32 的约定.

2)栈的管理(包括栈帧中的内容及存放顺序)也不必遵循 MIPS32 的约定.

测试环境

测试环境如下,值得注意的是,gcc 版本为 5.5.0,与助教测试机接近,不再是讲义所述的上古 gcc 版本.

~~gcc version 4.7.4 (Ubuntu/Linaro 4.7.4-3ubuntu12)(~~实验一~~)~~

gcc version 5.5.0 20171010 (Ubuntu 5.5.0-12ubuntu1~16.04)

flex 2.6.0

bison (GNU Bison) 3.0.4

QtSPIM version 9.1.22

运行方法

make clean; make; ./parser xxx.cmm out.s

实验步骤

1.指令选择

数据结构采用讲义所述双向链表串联 InterCode 的方法,并进行适当拓展.

这里我们只要根据讲义上的表 11 和自己的理解,构造 switch 语句,遍历实验三生成的 InterCode 链表,根据 interCode 的 kind 和具体的 Operand 选择具体的指令输出到 out.s,即可完成指令的选择.

2.寄存器分配

这里我们采用了课本上介绍的局部寄存器分配算法,即记录每个寄存器中保存了哪些变量,以及每个变量的有效值位于哪个寄存器中,这里我们构造了 REG_和 VAR_的结构体来表示前述两类记录.

当需要使用一个数据时,我们首先遍历 VAR 表确定该值是否在寄存器中,若在,直接返回寄存器的编号,反之,遍历 REG 表,寻找未使用的寄存器,先将对应的值存于寄存器,

再进行使用.若发现所有的寄存器都被占用时,我们对 REG 再进行一轮遍历,将 REG 中所存的常数、临时变量、临时地址清空(即删除 VAR 表中对应的结点,将 REG 表中对应的是否使用设置为 False).若此时寄存器仍然全部被占用,我们则考虑依次将寄存器压入栈或静态空间中,并设置好他们对应的偏移量,从而实现变量的溢出,当需要该变量时,再从栈中回读到寄存器中,关于变量溢出的寄存器的选择,为了避免在变量溢出上存在未知 BUG 影响正常情况(即为了兼容性),我们的实现为仅循环溢出 9,10 两个寄存器.

3.栈管理

这时候我们考虑实现指令选择中剩下的 ARG、PARAM、RETURN、CALL 等语句.我们将这些语句分为两类,一类是调用者语句,一类是被调用者语句,调用者要保存寄存器并依次按照 ARG 将对应变量存放至寄存器或者栈中,被调用者要依次将寄存器或栈中的变量读取(PARAM),当返回(RETURN)后,调用者要将之前的保存的寄存器依次恢复(恢复现场),值得注意的是调用者和被调用者对变量的处理一定要一一对应.

遇到的问题

这次的问题主要在于对 MIPS 指令的不熟悉,通过课程主页所给资料以及 CSDN,熟悉了 MIPS 常见指令,并搞清楚了某个指令到底是左边赋值给右边,还是右边赋值给左边.