目标代码生成 实验报告

151220151 殷瀚 411629507@qq.com

1.实现的功能

完成了必做任务：为 cmm 代码生成 MIPS32 汇编代码。

2. 实现方法及数据结构

2.1 寄存器分配

采用朴素寄存器分配法，所以实现起来就比较简单，每次要用之前就去栈上取值，赋值后立刻写回到栈上。

需要定义结构体 Var 和 Reg 来分别代表变量和寄存器。Var 中记录了变量的 name，在栈上的偏移量，以及现在存在哪一个寄存器中；Reg 中就记录了当前存储的 Var。

由于朴素寄存器分配法只需要几个寄存器就行了，所以只使用了 8-16 号寄存器，以及$v1 用来作为某些指令的临时寄存器。每次从中依次挑一个寄存器分配给变量。除了首次在中间代码中出现的变量，其余的变量在写入寄存器之后，立刻保存到栈上。

2.2 函数调用和栈管理

因为采用了最简单的寄存器分配策略，本次试验的主要工作量就是函数调用的实现。

2.2.1 栈操作

为了方便，全部采用栈底指针$fp 来进行栈操作，需要记录当前函数使用栈的大小，这个量保存在 MemManager.spOffSet 中，在函数头部初始化，每次有变量入栈或出栈时，修改这个变量，(spOffSet+$fp)就代表了栈顶指针。

这样的一个好处时，由于 c--程序对栈空间的使用都是静态的，所以在编译时记录下当前栈的大小即可，不需要生成代码去不停地在运行时修改$sp 的值。所以实际上，我生成的目标代码，除了在开始用来为$fp 赋值以外，就没有再使用$sp 了。

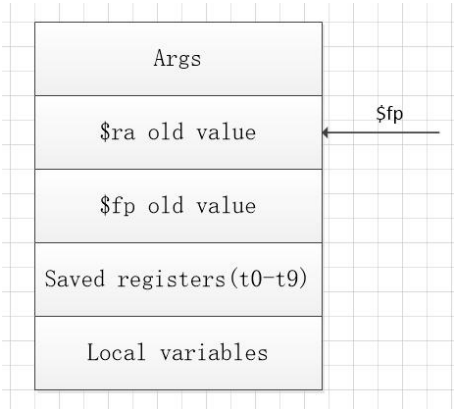
2.2.2 程序头

在开始翻译中间代码前，需要先生成一些代码，比如实现 read 和 write 函数的代码，这部分直接参考了示例代码中的开头部分。

另外，QtSPIM 默认没有使用$fp 作为栈底指针，所以我们在 main 函数的开头需要将当前$sp 的值移入$fp 中。并且，由于 main 函数没有调用者为其保存返回地址和$fp 旧值，我们需要在 main 函数的开头将这两个量入栈，以和本试验中设计的函数活动记录保持一致。

2.2.3 活动记录和函数调用

我的函数调用活动记录如下：



由于没有做活跃变量分析，所以活动记录中需要把所有寄存器都保存下来。并且，由于没有使用到需要由被调用者保存的寄存器（$s0-$s8），所以只需由调用者来保存和恢复$t0-$t9 即可。

在调用函数时，先将参数放入寄存器或入栈，再将返回地址、$fp 的旧值入栈。

2.2.4 参数传递

对于 ARG 类型的中间代码，由于我们要将前四个参数放入寄存器中，其余的放入栈中，但是中间代码中的 ARG 指令的顺序是倒过来的，也就是说，解析中间代码时，后面的参数会先被解析，此时我们无法知道该将其入栈还是放入寄存器中。

所以我对中间代码生成的部分做了修改，类型为 ARG 的中间代码，除了包含参数对应的 Operand，还新增了一个 argIndex，记录了这是该函数中的第几个参数。这样在生成目标代码时，就可以正确处理多个参数了。

对于 PARAM 类型的中间代码，不会将其翻译为目标代码，对其的解析实际上是维护了另一个“符号表”，告诉我们这个参数的值保存在$a0-$a3 寄存器中，还是在栈上。

3. 编译运行方法

直接在 Code 目录下 make 即可，将依次调用 flex 和 bison 生成.c 文件，再寻找 Code 目录下所有的.c 和.cpp 文件进行编译链接得到可执行文件 parser（生成的 parser 在 **Code 目录**）。

在 Code 目录下执行 make test 将调用 parser，对 Test 目录下所有.cmm 文件进行分析，生成同名的.s 文件，在项目目录的 MIPSCodes 文件夹中（需要先 mkdir）。

在 Code 目录下执行 make clean 将 Code 目录下的临时文件清理，同时删除可执行文件。