L4实验报告

191250145 王子鉴

0 文件说明

新增`asm.h`和`asm.c`，将中间代码翻译成目标代码

1 分配

本次试验寄存器分配方法采用了朴素分配方法。除了$a0用来传递函数调用的返回值，只用到两个寄存器$t0和$t1。中间代码中的临时变量、变量、地址变量都在栈上分配相应内存。

栈管理与实验讲义上活动记录结构类似。最小的栈为8个字节，记录了$ra和$fp中内容。

2 处理

处理分为两步

第一步`ASMPrepare`先扫描整个中间代码，通过记录函数体中所有变量(`ASMRecordVariable`)，对每一个函数计算其所需要的栈空间的大小和传递参数所需要的大小。在记录的过程中通过传递之前的变量大小的总和，给每一个变量分配在栈空间中相对于$fp的偏移量，并挂到中间代码上(IROperand内)

第二步`ASMTransList`扫描整个中间代码，根据代码和前一步生成的偏移量，生成目标代码。

对于一般的表达式如add之类的算术运算处理逻辑是先把需要用到的变量读取到寄存器中去，再对寄存器进行操作，最后再写回栈中。注意在写回栈中是根据$fp加偏移量来决定地址，同时栈向低地址延伸。所以offset为正时偏移量为负。对于parameter，如果想要其对于$fp偏移量为正，那么只能在预处理过程中将其偏移量置为负。

本次实验采取被调用者来管理参数、寄存器和返回过程的方法。在处理Arg时，不断给$sp - 4，并且讲参数写入相应内存。之后Call的时候直接jal 到目标地址即可。但在函数开始的阶段，需要先存起来$fp和$ra的值，然后把$fp置成$sp(此时$sp指向最后一个参数的后面一个字节，正好是$fp的位置)，最后把$sp - function.size(即伸展function中所有的变量大小的总和的bytes)。Return与call的过程相反，先把$a0置为返回值，再恢复$fp和$ra，最后删除被调用者的栈。

3 bug

本次实验遇到的bug：

1. 处理`*x = y`时寄存器写反了，一直读到不该读到的位置上去
2. 对于DEC中间代码，其operand是`Variable`类型的，会多分配一部分内存，并且DEC的v1和后面的&v1指向的地方不同，所以在预处理时要把类型改为`Address`。(并不会影响中间代码的生成，因为outputIR先于目标代码生成)
3. 在本地跑的时候，测试脚本对多个输入无法处理。疑似为和mac不兼容导致的，换到linux服务器上跑就可以通过。

感谢孙伟杰同学的debug指导，感谢鄢振宇学长的测试脚本