**编译原理实验 lab4**

1. 你的程序实现了哪些功能？简要说明如何实现这些功能

在本实验中，程序主要实现了从中间代码向基于RISCV汇编指令的转换。在lab3 中，我并没有为中间代码建立任何数据结构，仅仅是使用string类型将其存留并输出至文件；因而在本实验中，我需要按行将中间代码生成的string进行切割，再根据具体的中间代码类型，进行相应的RISCV汇编语言转化。

因为中间代码的语句局限性，使得转化为RISCV汇编指令的指令选择过程变得较为简洁，只需要根据当前string行的已有信息，转化为RISCV汇编string输出即可。在寄存器分配方面，暂且仅使用了最为朴素的寄存器分配方法，即对于任何中间代码而言，只使用了t0，t1，t2三个寄存器用于数据运算；在函数调用时则考虑a0 - a7寄存器进行参数的传递；另外包括sp、s0、ra等必要寄存器的使用。

在最开始的正确性测试中，程序仅仅只能测试正确性，对数独计算的测试文件，并不能达到5s的时间运行要求；经输出，发现汇编语言大约进行了38w步，从而只能在7s左右完成测试任务。在此基础上，我对中间代码的生成过程做了一些细微的优化，主要是数组取值时简化了对立即数的一些使用，使汇编语言的步数减少到27w左右，从而到达了5s的时间运行要求。

2.你的程序应该如何被编译？请详细说明应该如何编译你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数

解压相应文件后，在最外层文件夹使用 “make compiler”命令即可获得compiler二进制文件，区别于lab1、lab2的compiler文件，本实验生成的compiler文件不仅接受一个文件的输入，同时输出一个具有RISCV汇编指令高的.s的文件。

若使用“python3 test.py ./compiler lab4”测试命令，只可判断compiler的通过性正确率；使用“./compiler xxx.sy xxx.s”测试命令，则可以获得词法分析器（每一个token）的输出结果；语法分析树（有缩进结构）的打印结果；(暂时在代码中被注释）语义分析与语法分析阶段的正确与否，分别显示“Successfully！”与“Failed!”及相应错误原因以及基于经编译SysY语言后转化为IR中间代码而产生的RISCV汇编指令文件。

1. 所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，尤其要避免大段地向报告里贴代码

3.1 jump main的巧用

在实验指导方案中，提及了我们需要将main函数放到汇编语言文件的最开始，这才可以顺利的执行我们输出的RISCV文件。但在编写SysY语言文件时，我们设定了所有函数都需要在main函数前定义的语义，并没有支持函数声明；从而迁移main函数主体，可能带来极大的工程量与错误可能性。从而我选择在.text后直接加入语句jump main，使程序运行时直接跳转至main函数处，这很好的解决了main函数整体的迁移需求。

3.2堆栈的提前规划

将中间代码转变为RISCV汇编语言，最大的工程是对寄存器的分配。因为使用了最朴素的寄存器使用策略，这对于中间代码数以百计的寄存器使用而言，是无法支持的；因而需要大量的使用栈来存储这些临时变量。

本实验中我选择用提前规划sp寄存器的方式，仅在进入函数时开辟栈空间，退出函数时回收栈空间，不在函数执行内部随意改变栈指针的位次。因而在进入函数时，我会提前遍历本函数的所有中间代码，一方面记录临时变量的使用个数，便于开辟合适的栈空间；另一方面记录使用函数时会传入的参数个数，便于处理参数个数大于8的情况。

在函数参数个数大于8时，a0 - a7寄存器已不能满足参数传递需求，我们需要借助栈来实现这一目的。此时，我们不得不移动栈指针，来存储这些数据。因为进行参数传递时，我们无法预知参数个数，从而只能对于每个溢出的参数进行让sp进行 -4的操作开辟空间，并在call 函数指令结束后，使sp指针指回原始位置。在接受参数的函数一方，因为我们事先统计了参数的传递个数，从而可以直接进行sp指定位置的数据读取。

3.3 函数的调用

本次实验中，函数的调用也存在着一些注意点。首先是函数“迭代”调用的问题，这会导致ra寄存器正常返回位置的数据丢失，导致第一层函数调用的返回失败。从而在开辟栈存储空间时，我们不仅要考虑s0寄存器、临时变量的存储，也同样要考虑ra寄存器的存储与读取。

另一方面，数组的使用也是一个大问题。在中间代码的实现中，在函数体内定义的数组和int变量，均使用了统一的临时变量标识符表示。但对于int变量而言，标识符记录的是对映栈空间中存储的数据；而对于函数内定义数组而言，标识符记录的应该是其数组头对映地址位置，而栈空间则还是存放数据相应的首个元素。这在实际使用数组与函数参数传递数组时，产生了选择栈空间地址还是栈空间内容的干扰。

这里我便将map结构（临时变量序号与栈空间位置对映图）中数组临时变量的栈空间位置记录为其对映的相反数，从而作为数组头的标识，指导当前临时变量标识符是对栈地址的读取还是对栈地址空间内容的读取。