|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《编译原理》实验报告** |
|  |
| 学 院: 计算机科学与技术   |  |  | | --- | --- | | 姓 名: | 王靳 | | 学 号: | 220111012 | | 专 业: | 计算机科学与技术 | | 日 期: | 2024-11-17 | |

# 1 实验目的与方法

**实验目的为本次实验的实验目的，方法为所使用的语言，软件环境等。**

实验目的：将简单的语言（赋值、计算）翻译成riscv64汇编。本次使用使用java语言编写。个人在生成汇编的时候做了一些操作：可以被gcc交叉编译器编译成elf文件，并且可以在qemu user mode下运行

## 词法分析器

体验手搓正则表达式分析器（确定有穷自动机）的过程，将raw text输出为一系列的token

## 语法分析

体验LR实现语法分析的过程，识别文法，给语义动作用的

## 典型语句的语义分析及中间代码生成

体验L-SDD对应的语法制导翻译，实验二识别id, imm, 产生式，实验三根据实验二识别的内容，执行对应的语义动作，也就是向一个vector中添加TAC中间表示语句

## 目标代码生成

体验指令选择、寄存器分配。根据TAC生成rv64汇编。我这里做了多余的工作：我可以将输出的汇编通过gcc汇编器，生成elf文件，并且这个文件可以在qemu user mode下执行。或许可以更好的自动化测试。

# 2 实验内容及要求

**每次实验室的实验内容和要求描述清楚。**

## 2.1 词法分析器

词法分析就是：字符串进行划分，划分成一个一个的词素，并识别词素，赋予相应的词法语义，比方说：如果是id，那么就会加入到符号表中，如果是立即数，那么就会将立即数转换成java的立即数。最终准换成token

## 2.2 语法分析

使用编译工作台：将上下文无关文法转换成对应的LR状态转移表，输出到.csv文件中。实验读取这个LR状态转移表，这个LR状态转移表被读进一个数据结构里面，可以通过status, token得到LR状态转移表中的状态。LR(1), SLR(1), LALR(1)这些只是构造LR状态转移表的方法有些许不同，但是最终都可以得到比较标准的LR状态转移表，以及为状态转移表抽象出来的api，这让语法分析的过程非常的方便。最终将分析的结果用一个观察者记录下来。

## 2.3 典型语句的语义分析及中间代码生成

这个语言正好满足L-SDT，根据实验二生成的文法，生成TAC中间表示。这个语义分析、生成中间代码的过程是与语法分析同时发生的，这得益于观察者模式：语法分析中，如果发生了reduce, shift, 那么就会通知对应的观察者，语义分析中的whenShift, whenReduce，实际上就是对应观察者的callback函数。语义分析中whenShift, whenReduce会匹配id, imm, reduced的产生式，匹配到后，执行对应的寓意动作，生成TAC中间表示，这个中间表示与SSA不同，SSA是一个图，而TAC直接放到一个列表中就行了。我们可以通过IREmulator验证实验三的正确性

## 2.4 目标代码生成

目标代码生成就是编译器后端该做的事了，最简单的目标代码生成实际上也就两个步骤：指令选择、寄存器分配。指令选择，因为我们的目标平台是rv64, rv64的汇编正好是三地址形式的，实际上直接可以一一对应的翻译，如果是CISC，就会有许多复杂的算法，FastISel, GLobalISel之类的。寄存器分配，也有许多的算法，图着色、弦图、线性规划，我这里非常的懒，干脆就不考虑寄存器溢出的情况，如果溢出，那么就直接抛异常

# 实验总体流程与函数功能描述

## 词法分析

词法分析实际上就是写一个识别正则语言的自动机。将raw text拆分成一序列的Token

### 编码表

定义了每个词法单元的枚举值，以及文字描述

### 正则文法

比方说int就是需要在raw text中匹配“int”, 需要注意的是：id是: 以letter或者是\_开头的字符串，也就是^(\_|letter)(\_|number|letter)\*, 而IntConst就是number+。

### 状态转换图

TODO

### 词法分析程序设计思路和算法描述

我是采用了类似于libc中的处理，一个while循环(last)，不回头的遍历整个raw text，并用一个向前看指针(cur)来向后识别，那么raw[last, cur-1]就是对一个的词素了，通过对这个词素进行匹配，生成对应的Token（路子比较野）。当然了，这里也是硬编码的了，并不会用到coding\_map.csv



## 3.2 语法分析

将Token序列识别为一个产生式序列，当然这个产生式序列是虚拟的了，他的生命周期很短，识别到了产生式后，通过观察者，将信息传递给对应的callback函数。

### 拓展文法

拓展文法就是对LR分析的拓展，因为带有语义动作。就是这个txt语言并非一个完全的上下文无关语义，有许多地方会有上下文：比方说上面定义了一个变量，下面使用。

### LR1分析表

LR1分析表是由编译工作台生成的，但可惜的是，这个是windows程序，在我的机器上跑不起来，但是好在项目中有自带LR1\_table.csv。分析表将LR, LALR, SLR标准化了，统一成了一个状态转移表。

### 状态栈和符号栈的数据结构和设计思路

状态栈采用栈结构，java STL

符号栈其实并不需要，因为他会在语义分析中保存。

我用一个队列保存纸带上剩余的输入符号

### LR驱动程序设计思路和算法描述



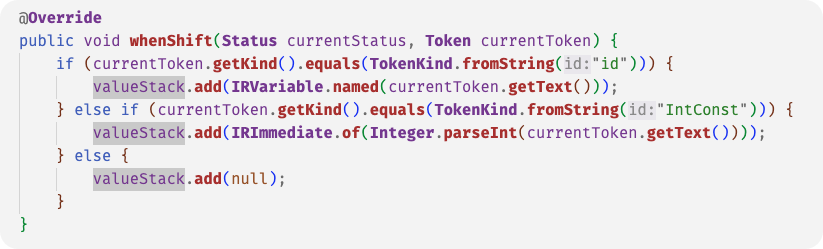
循环，直到纸带上没有剩余的符号。状态栈中push初始状态。根据状态栈栈顶，前瞻符号，得到对应的动作、状态。执行相应的动作(reduce, shift)，并跳转到对应的状态（状态栈pop若干状态、状态栈push action对应的状态）。把识别到的文法传给对应的观察者。

## 语义分析和中间代码生成

实验二将识别到的文法传给了对应的观察者。观察者执行对应的语义动作。TXT语言是L-SDD的，但是其实几乎就是S-SDD，因为只有一种数据类型，也就是int，几乎可以按照S-DD的SDT分析方法，只不过需要检查符号表之类的。

### 翻译方案

Shift 时, 如果是 id, IntConst, 那么就会读取对应的词法值, 包装成operand



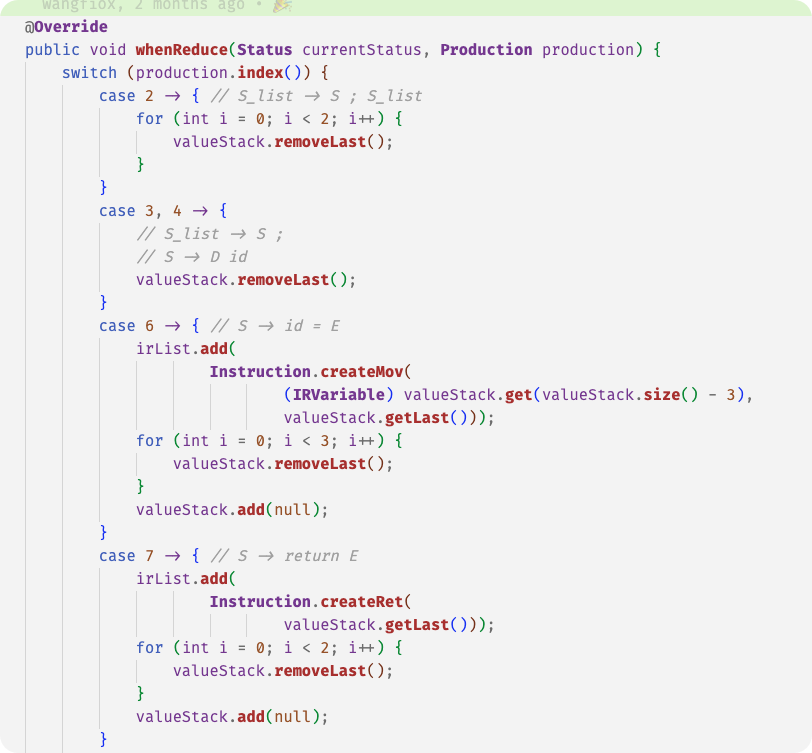
Reduce时

S\_list -> S ; { 从 valueStack 删除 semicolomn }

S -> D id { 从 valueStack 删除 id }

S -> id = E { 从valueStack删除 id = E, 并在irList插入mov id, E }

S -> return E { 从 valueStack 删除 return E, 并在 irList 插入 ret E }



E -> E + A { 从valueStack中删除 E + A, 产生一个res, 并在 irList 中插入 add res, E, A }

与8同理



与8同理

B -> ( E ) { 从 valueStack 中删除 ( ) }



### 语义分析和中间代码生成的数据结构

irList, TAC列表

valueStack, 语义分析对应的栈

语义分析会检查对应的符号在符号表中是否存在



### 语法分析程序设计思路和算法描述

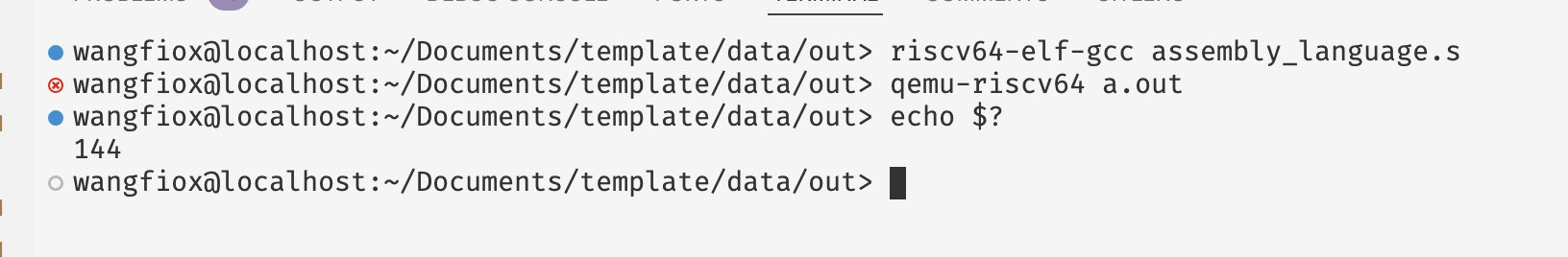
实际上也就是两个观察者的实验，一个是语义检查，一个是语法制导翻译。语法制导翻译，这里面有valueStack，实际上也就是符号栈。语法分析在实验二基本上就完成了，能识别产生式并传递给相应的观察者（回调函数），观察者获取到相应的信息，并在栈中保存句柄，并且执行相应的语义动作

## 目标代码生成

指令选择、寄存器分配。

**3.4.1 设计思路和算法描述**

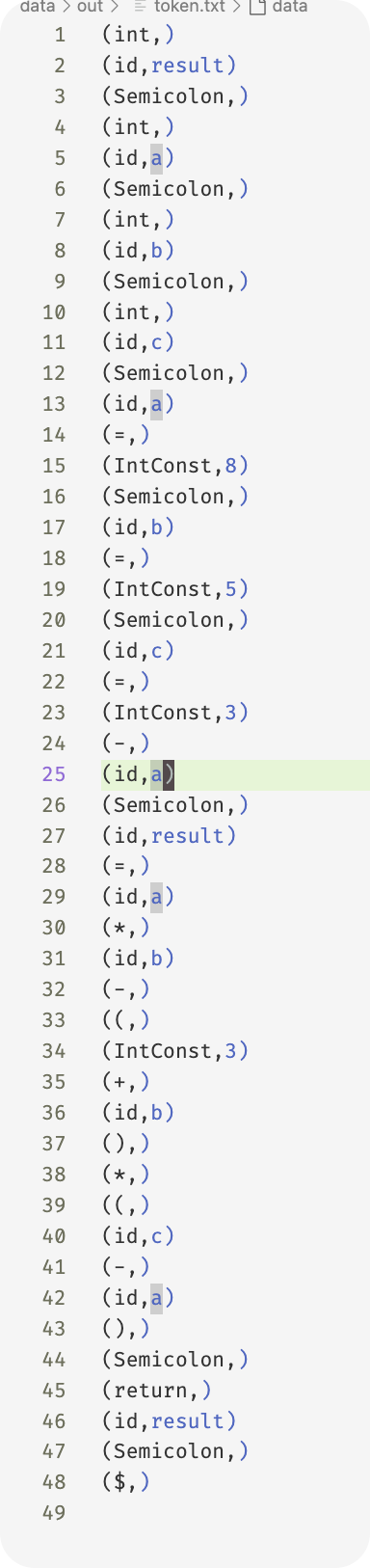
指令选择非常的简单直白，基本上是一对一的翻译。寄存器分配也非常简单，依次给IR的virtual reg分配对应的physical reg，如果需要的寄存器个数超过31，那么就抛出异常，也就是根本不考虑寄存器溢出的情况。我输出的汇编可以被gcc汇编器识别，输出到riscv64 elf文件，并且能用qemu user mode执行。

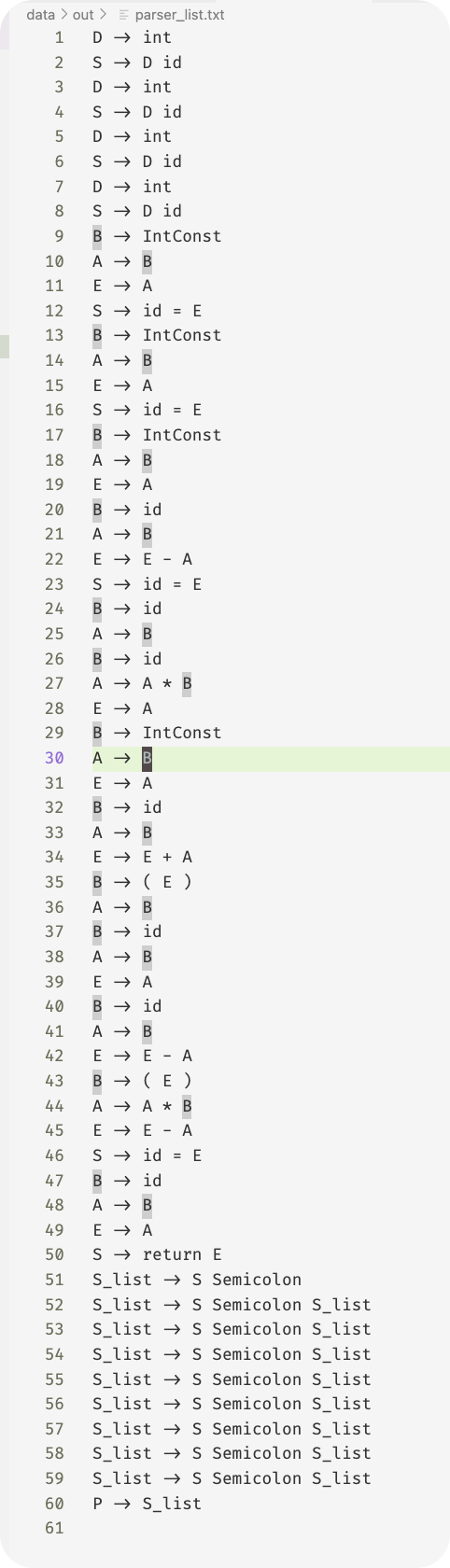


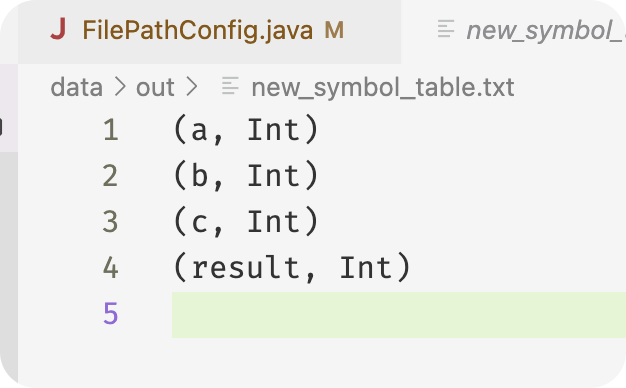
# 4 实验结果与分析

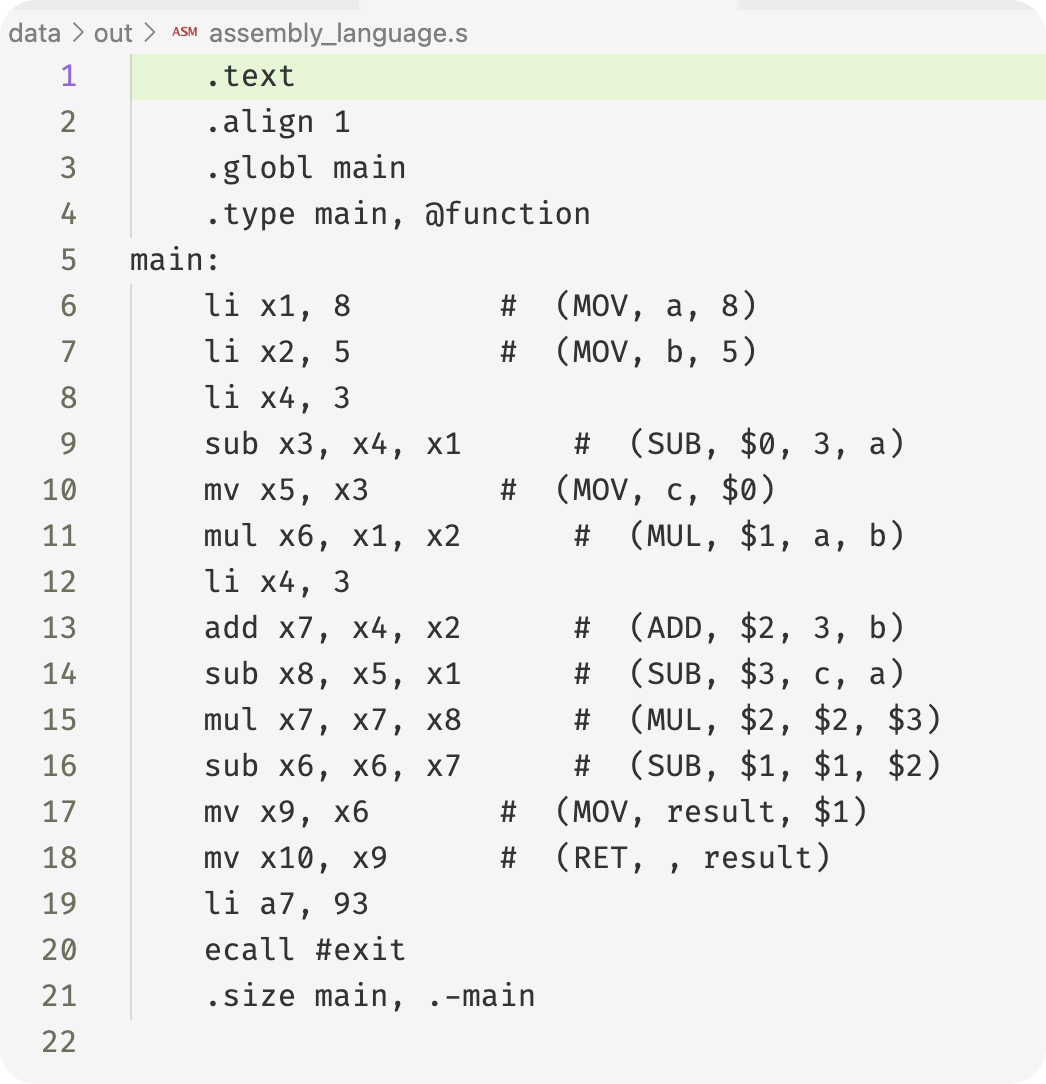
**对实验的输入输出结果进行展示与分析。注意：要求给出编译器各阶段（词法分析、语法分析、中间代码生成、目标代码生成）的输入输出并进行分析说明。**

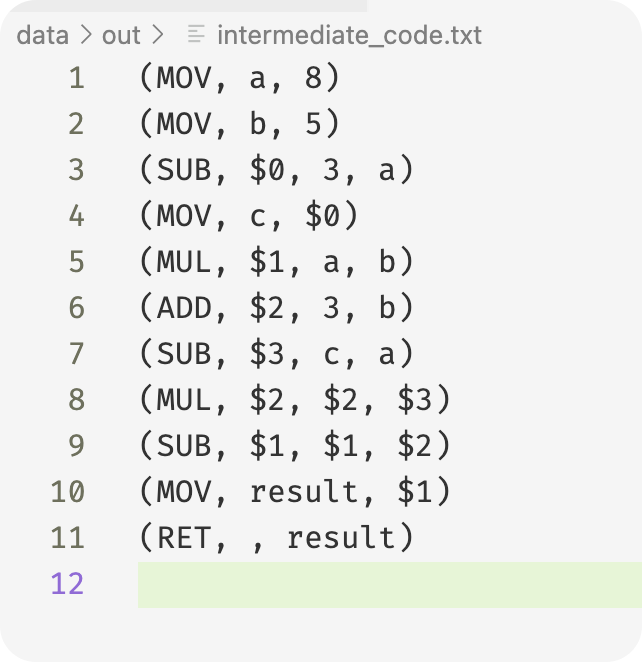
词法分析

****



中间代码生成

目标平台代码生成



# 5 实验中遇到的困难与解决办法

**描述实验中遇到的困难与解决办法，对实验的意见与建议或收获。**

让我遇到最难受的一个问题就是：我实验二调试出现了问题，结果这个问题是实验一词法分析错误导致的。解决方法：先用正则表达式库对我实验二的代码进行差分测试，验证了我实验二的代码是对的，再重写了我实验一的词法分析（可以看到，我实验一和实验二提交的lexer有些许不同）