由于对编译器实现没有过多经验，本次的编译器实现分为**两个线进行**。

第一条线-AST生成汇编(可执行并通过62个测例)

程序在Compiler文件夹里。

第一条线是从AST生成汇编，这一方式原本准备生成LLVM IR，LLVM IR和汇编很相近，为了节省时间，最后采用了从AST直接到汇编的方式。

由于没有参考文档，本方法使用了很多新方式来达到想要的效果，下面对这种方式进行介绍。

基本组成

由 bindings.cpp，compiler_main.cpp，declaration.cpp，expression.cpp，function.cpp，statement.cpp，translation_unit.cpp，type.cpp ， hshlex.l，hshparser.y 组成，代码总量在5000行左右。

- 在**bindings.cpp**中，通过收集符号和其基本属性，组成了一个符号表，但是，这种符号表并不同于往常的符号表，有很多的创新点。

  > - 创新点1：**采用c++的unorder_map存储符号**，在map容器里，符号和其属性会组成相应的键值对，可以通过符号来索引其属性，这一过程方便快捷且高效。
  > - 创新点2：**使用值传递的方式来实现局部性**。假设两个作用域A，B，B作用域包含在A中，有一个变量x，在A中有声明，B中也有声明，当在A中的时候，查询符号的属性，得到的是在A中声明的值和类型；当进入B中的时候，B中的声明会覆盖A的声明，符号表中的内容就转化为B局部所有的。由于是值传递的因素，在A中的符号表没有发生变化。这一过程也不必添加指向上一级的指针，因为如果上一级的bindings传入，下一级没有改变，bingdings也不会改变，符号所对应的属性就是上一级的。这是我经过很长时间思考想到的，是浪费了一定的存储空间，但是效果是一致的，比起普通的符号表可以节省一些时间。

> - 创新点3：**包含内容广泛**。将当前的栈位置，表达式的栈位置分别记录，以便生成汇编时使用。将break_lable和continue_label也分别记录，通过这一方式，也可以实现循环的多层嵌套

- 在**type.cpp**中，每个类型都有一个类，我实现了int, char, short,还有string，string本来是为了实现printf的翻译，最终发现测例中没有，也就没有实现。每一个类型都有属于自己的load函数和store函数，用于打印此类型的加载和存储回汇编语句。
- 在**expression.cpp**中，是对表达式的处理，包括赋值，加减乘除，单目运算，自增自减，函数调用，数组操作等一些较为基础的表达块组成。每个表达式都有与之对应的汇编语言。
- 在**statement.cpp**中，是对语句的处理，包含ifelse语句，while语句，for语句，switch语句（写了一下，还没测试过），表达式语句，goto语句，break语句，continue语句。经过此文件的处理，识别到相应的语句后，会生成完整的可执行的汇编。
- 在**declaration.cpp**中，对一些声明语句进行了处理，包含基础的变量声明（局部和全局都实现了），数组（仅实现了局部，没有实现全局）。在这些声明语句的识别过程中，即会生成相应的汇编语言，也会在bindings中进行变量的更新。
- 在**function.cpp**中，对函数进行了处理，每个函数的参数会被放入相应的寄存器或者栈中。函数主要有参数，声明列表和语句列表组成，在打印function的过程中也许要对其组成部分进行打印
- 在**translation_unit.cpp**中，对编译单元进行了处理，这一部分仅是将不同需要翻译的函数和全局声明存储在一块，方便依次打印。
- 在**compiler_main.cpp**中，是先生成语法树然后进行遍历，生成相应的汇编。

在compiler/include文件夹中，是头文件。

在compiler/build文件夹中，有可执行文件COMPILER。

在compiler/test文件夹中，是我自己写的测试文件，这些全都可以过。官方测例可以通过62个，具体情况在systest.xlsx中有体现。

第二条线-AST生成IR（仅仅生成4元组，没有生成汇编）

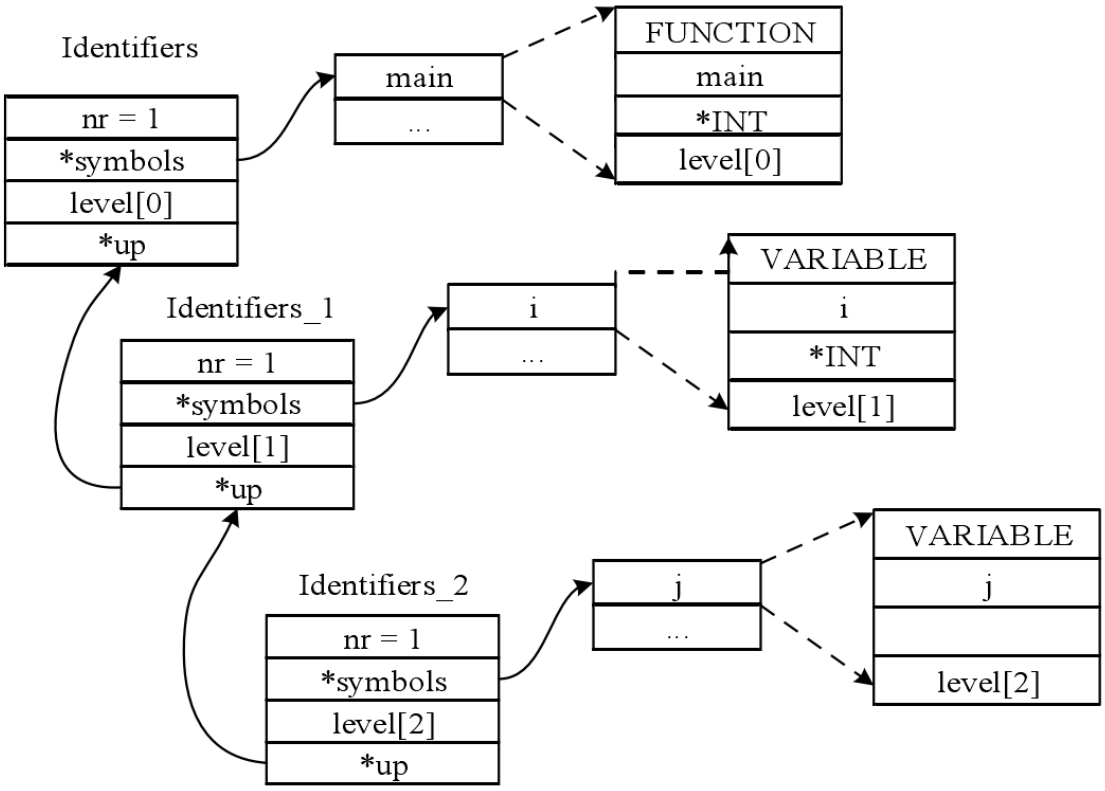
这条线是按照标准的流程进行相应的实现，不过并没有完成从c语言到汇编的全过程。

基本组成

types.cpp和types.hpp中定义了相关的结构，包括符号表和四元组表，也定义了相应的查询插入修改的函数，一百年对这些数据结构进行操作。

hshlex.l和hshparser.y分别是flex和bison文件，对c语言进行语法和词法分析。

符号表的具体组成如下图



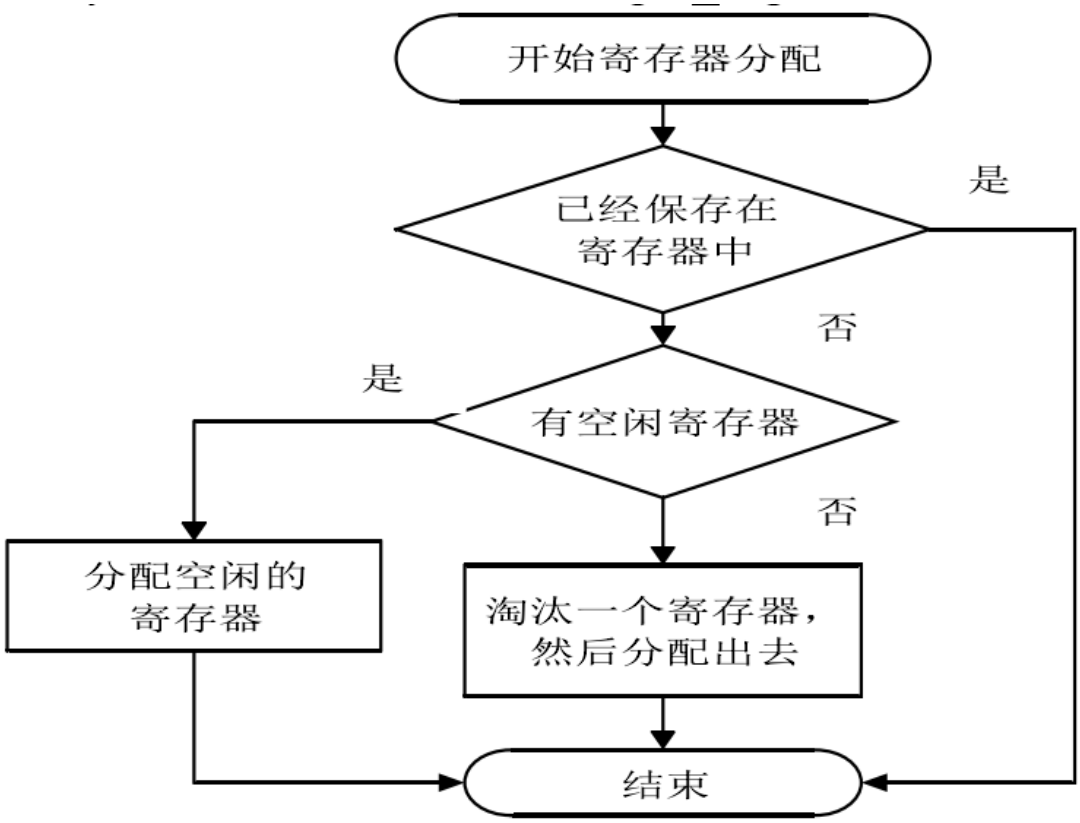
四元组表的设计如下图所示

	label	opcode	opds[0]	opds[1]	Opds[2]
1	0	+	t1	a	1
2	0	>	128	b	t1
.	0	=	c	b	NULL
.	...	...	...	...	...
12	128	=	c	a	NULL
.	...	...	...	...	...

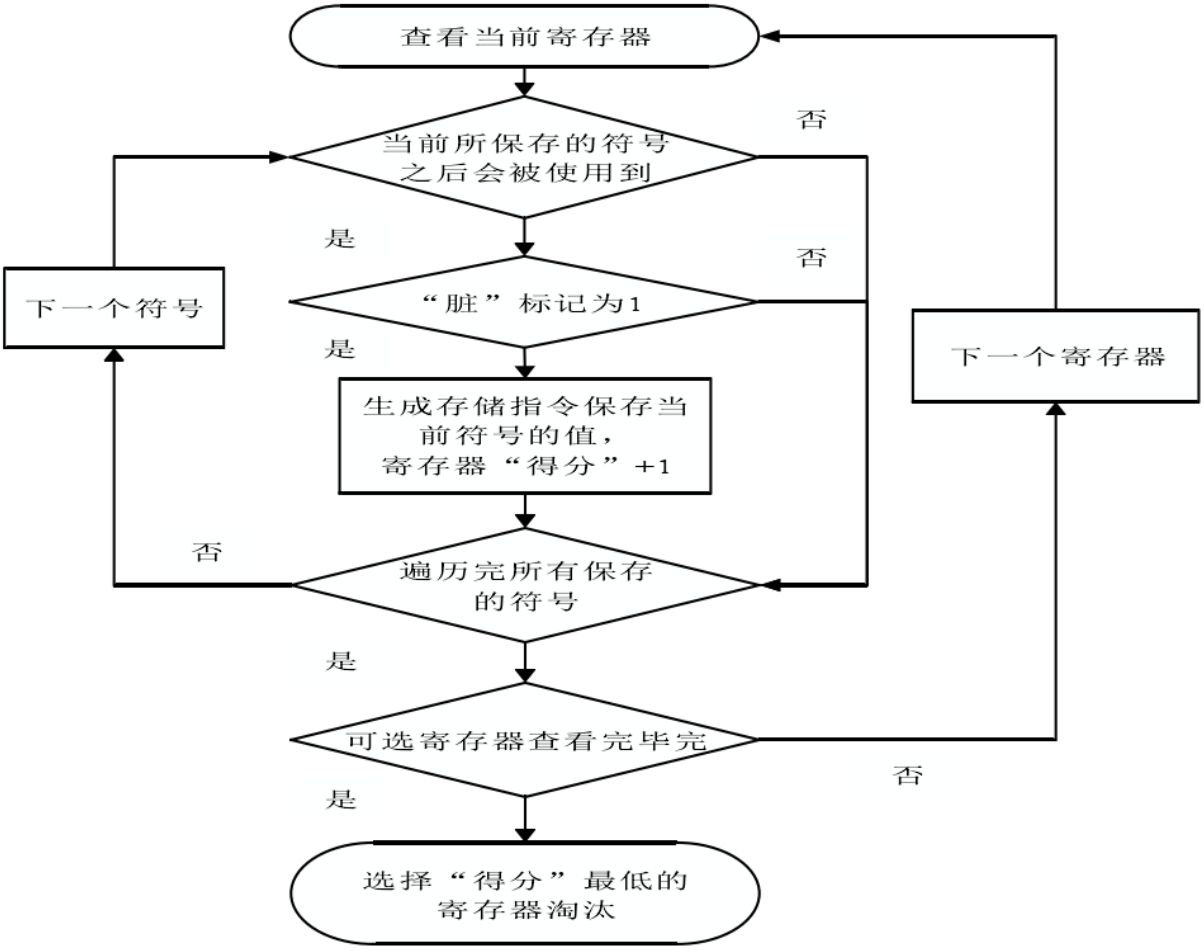
这一部分没有进行优化和汇编生成以及寄存器分配，但是，对这些内容都进行了学习。

寄存器分配

对寄存器的分配要按如下流程进行：



淘汰寄存器时没有使用图着色算法，通过遍历，来计算寄存器得分，选取最低分的寄存器抛弃



汇编生成

汇编生成和在第一条线中一样，我已经掌握了RiscV的使用方法。

中间代码优化

目前没有实现，但是对北大编译原理实验文档进行了学习