

**本科生《编译原理》课程实践报告**

**题　　目： 目标代码生成实验**

**学 院： 徐特立学院**

**专业名称： 计算机科学与技术**

**姓 名： 陈照欣-1120191086**

## 实验目的

（1）了解编译器指令生成和寄存器分配的基本算法；

（2）掌握目标代码生成的相关技术和方法，设计并实现针对x86/MIPS/RISCV/ARM 的目标代码生成模块；

（3）掌握编译器从前端到后端各个模块的工作原理，目标代码生成模块与

其他模块之间的交互过程。

## 实验内容

基于 BIT-MiniCC 构建目标代码生成模块，该模块能够基于中间代码选择合适的目标指令，进行寄存器分配，并生成相应平台汇编代码。如果生成的是 MIPS 或者 RISC-V 汇编，则要求汇编代码能够在 BIT-MiniCC集成的 MIPS 或者 RISC-V 模拟器中运行。需要注意的是，config.xml 的最后一个阶段“ncgen”的“skip”属性配置为“false”,"target"属性设置为“mips”、“x86”或者“riscv”中的一个。如果生成的是 X86 汇编，则要求使用 X86 汇编器生成 exe 文件并运行。

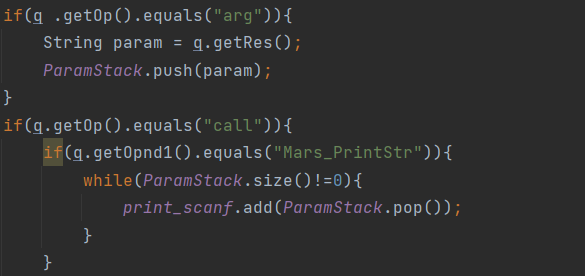
## 实验步骤

全局维护一个变量stringBuilder用来存储生成的汇编代码。

由于本次实验提供的测试用例的输入输出函数，Mars\_PrintInt，Mars\_PrintStr，Mars\_GetInt在x86中并未定义。所以需要通过借用对应的c语言函数，又由于x86中对应的printf和scanf的调用处理较为特殊，所以在此处对于这一类函数调用进行特殊处理。所以在完成lab8时对lab7中间代码部分增加了内置函数的识别。

本实验中新建了一个类MyQuat，与lab7中建立的结构类似，只是元素由AST节点换成了String类。在生成目标代码前先使用LoadIC转换，方便后续的读取。

首先由于我生成的四元式中的函数调用形式为：先用arg表示要传入的变量，再使用call来进行函数调用，所以，通过维护一个ParamStack，将遇到arg四元式时，将内容压入，即要传入的参数，当遇到call四元式时，将ParamStack中的变量均弹出，即可进行函数调用。所以，对于Mars\_PrintInt，Mars\_PrintStr，Mars\_GetInt这三个特殊函数，对应的输出字符串首先定义在x86代码的.data段，以便后面调用。

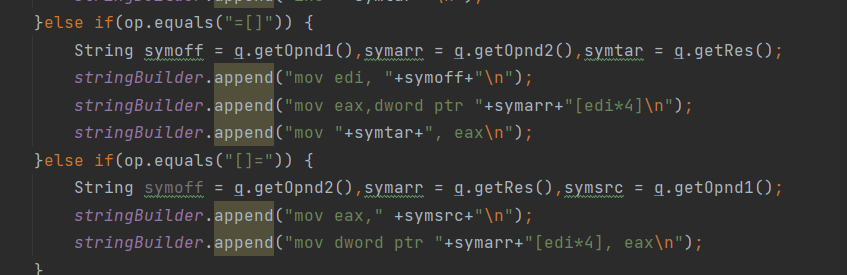


对于四元式中频繁出现的@1，@2之类的临时变量，本次实验选择将其都定义为局部变量，不进行寄存器分配。我们通过维护符号表，将TemporaryValue统统加入对应函数的符号表中，这样在访问到func四元式时，即可访问对应的函数的符号表，将其中的变量通过local关键字进行定义。

同时，对于数组的处理与临时变量类似，不过需要将多维数组展开成一维数组进行定义。



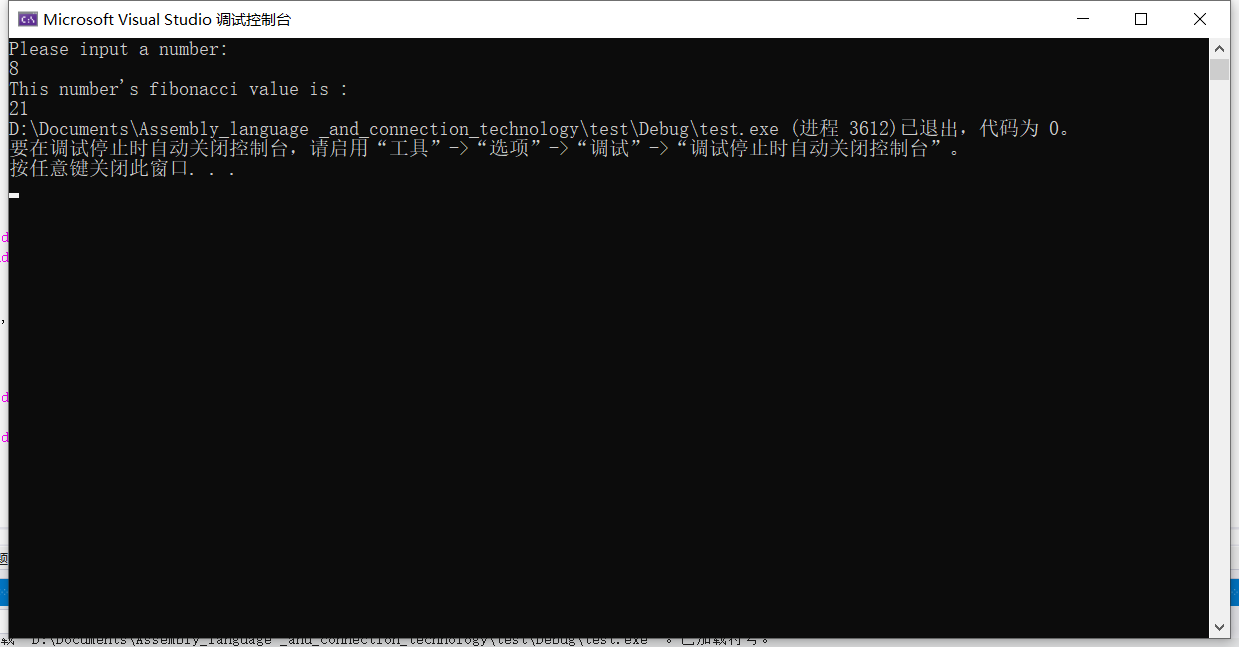
后续调用时，“[ ]=”用来实现对数组的访问，“=[ ]”用来实现对数组的赋值



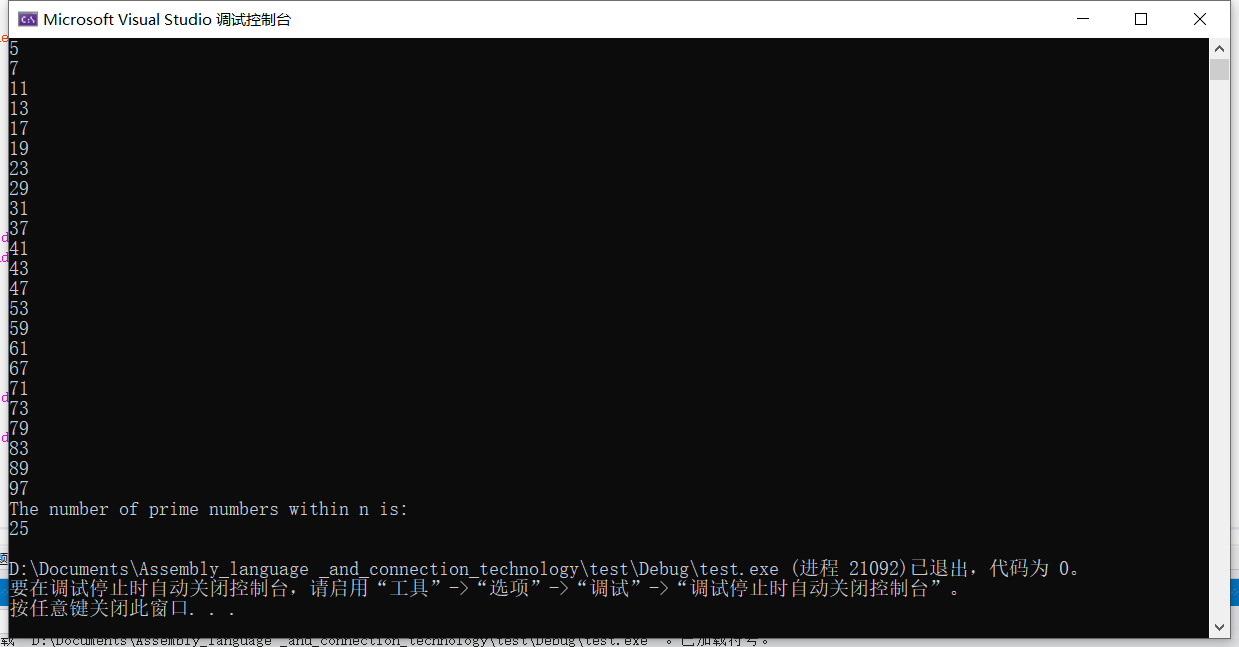
## 实验结果

本次实验能够编译通过的样例为1\_Fibonacci.c，2\_Prime.c，3\_PerfectNumber.c

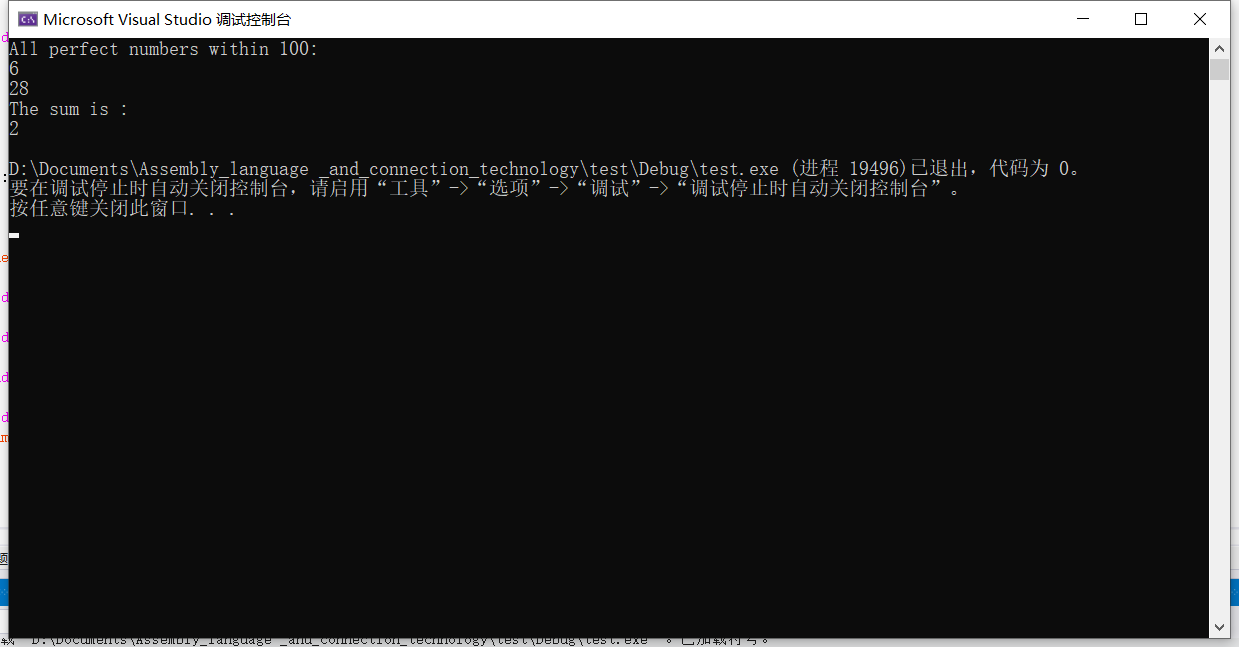
1\_Fibonacci.c



2\_Prime.c



3\_PerfectNumber.c



## 实验总结

在词法分析，语法分析，语义分析，中间代码一系列实验的基础之上，勉强完成了一整套编译器的搭建。从lab5开始的无比痛苦到后面思路的逐渐明朗，我对编译原理的一整套流程也有了渐渐清晰的认识。每次在尝试新实验的时候都会因为各种bug而去反思先前的实验是否有缺陷，进而理解了每一步在这一阶段的真实用意。

最后一段时间真的很紧，考试周安排很紧张，考好了之后开始备战夏令营，还需要积极完成整套架构。这个过程真的很累，但是在看到自己生成的代码真的能像C语言编译器一样运行时，还是有了极大的满足感。

最后感谢计老师的悉心指导，也希望之后自己能够抽出时间完善自己的这一套架构。