编译原理与设计实验报告

姓名：卜梦煜 学号：1120192419 班级：07111905

# 1. 实验名称

目标代码生成实验

# 2. 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/RISC-V/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。

# 3. 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。

# 4. 实验环境

Intellij IDEA Community 2021.3.2、masm32

# 5. 实验过程与步骤

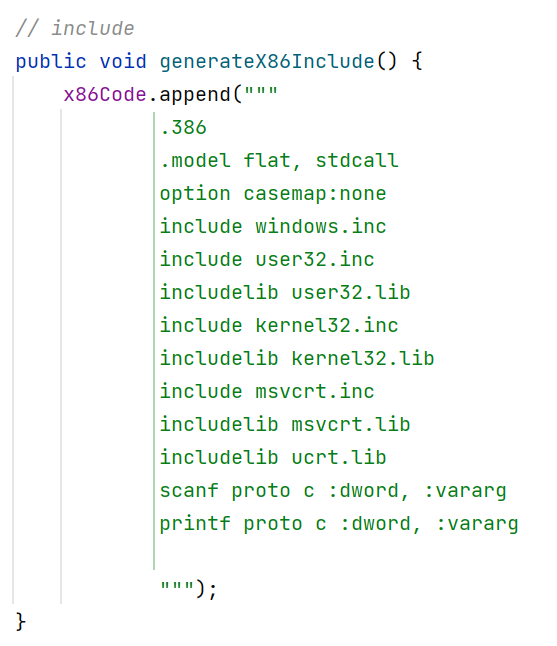
本实验的目标是基于中间代码生成模块生成的四元式，编写编译器目标代码生成模块，生成的目标代码格式选择x86汇编语言。

本实验主要步骤如下：首先编写x86代码框架，生成引用头文件部分、数据部分、代码部分；之后对中间代码四元式按类进行翻译，编写对应的目标代码；最后修改配置文件，将测试用例生成的x86汇编代码在masm32汇编器上编译成.exe文件并运行。

## 5.1 编写x86代码框架

x86汇编语言代码框架主要由引用头文件部分、数据部分、代码部分组成，编写的四元式大多对应代码部分，因此在进行四元式翻译前，需要先生成x86代码框架。

引用头文件部分包括.386框架伪指令、头文件引用、C函数调用。框架伪指令、头文件引用对一般的汇编代码是固定的，直接加入生成的汇编代码即可。本次实验提供的测试用例中输入输出函数较为特殊，包括输入函数Mars\_PrintStr、Mars\_PrintInt和输出函数Mars\_GetInt，分别对应于C语言的printf、scanf函数，声明需要写在C函数调用部分。引用部分代码如图。



数据部分包括全局变量，需要定义为全局变量的包括scanf数字输入格式和printf数字输出格式、以及Mars\_PrintStr函数中字符串。Mars\_PrintStr函数中字符串需要定义为全局变量的原因是，如果不定义为全局变量则需要声明全局的临时字符串空间，遇到字符串常量时需要进行字符串拷贝，导致代码较复杂，因此直接将字符串声明成全局变量会更方便。



代码部分包括定义的各个函数。函数包括函数声明语句、参数声明语句、局部变量声明语句、函数体代码、函数定义结束语句，由四元式形式的中间代码得到。对于函数体中定义的临时变量，为避免寄存器分配带来的困难，本实验选择将寄存器分配工作交给汇编器，将所有临时变量声明成局部变量，这样保证了在访问四元式时，遇到的所有变量都已经预先声明为局部变量或函数参数。

## 5.2 编写四元式翻译为x86汇编代码的方法

四元式翻译为x86汇编代码是本次实验的核心，基本方法是根据四元式类型和内容，编写对应的汇编代码生成格式。四元式类型由四元式中op字段确定。本实验需要对四元式组按照函数进行翻译，每个函数做两次遍历，第一次遍历定义局部变量，第二次遍历翻译其他四元式的代码。

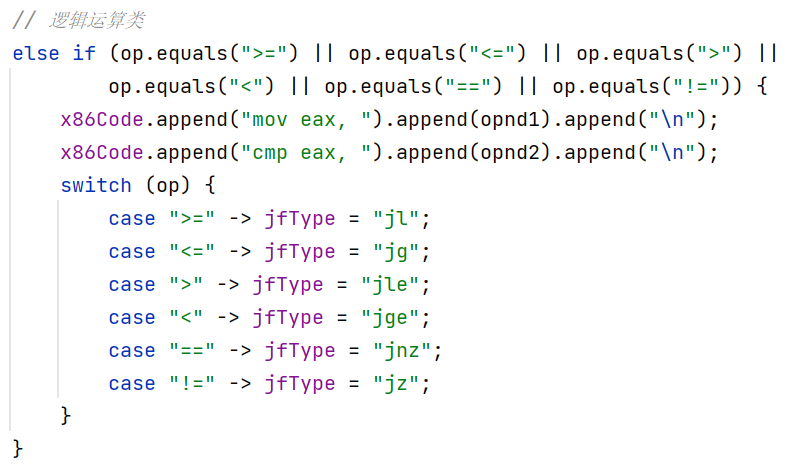
函数范围界定相关的四元式op包括param、func、funcEnd。每个函数以func为开始，funcEnd为结束，对应翻译为汇编代码“函数名 proc”、“函数名 endp”，main函数结尾还需要语句“end main”。函数形参由param指出，对应翻译为汇编代码“参数名 :参数类型。

局部变量定义需要紧跟在函数声明语句后，范围包括：以@开头、在当前函数内未定义的变量，四元式op为var、arr的变量，四元式op为arg且非Mars\_PrintStr的变量。对应翻译为汇编代码“local 局部变量名 :变量类型”。

跳转相关的四元式op包括label、jmp、jf、ret，用于控制程序的执行顺序。Label、jmp、jf的标签名存放在四元式的result中。jf四元式需要根据条件判断的类型，生成条件为假时的跳转指令。ret四元式需要根据是否有返回值编写，有返回值时将函数返回值保存在eax寄存器中。生成汇编指令的代码如下。



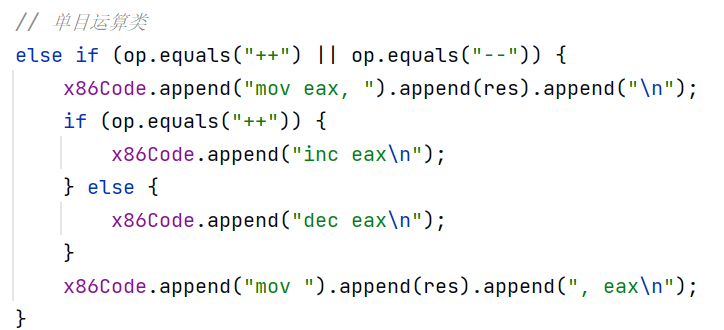
逻辑运算相关的四元式op包括>=、<=、>、<、==、!=。由于逻辑运算往往用于控制程序跳转，而程序跳转指令根据标志寄存器判断，因此逻辑运算需要使用cmp指令完成，对于每个op，保存条件为假时对应的跳转语句供跳转语句使用。生成汇编指令代码如下。



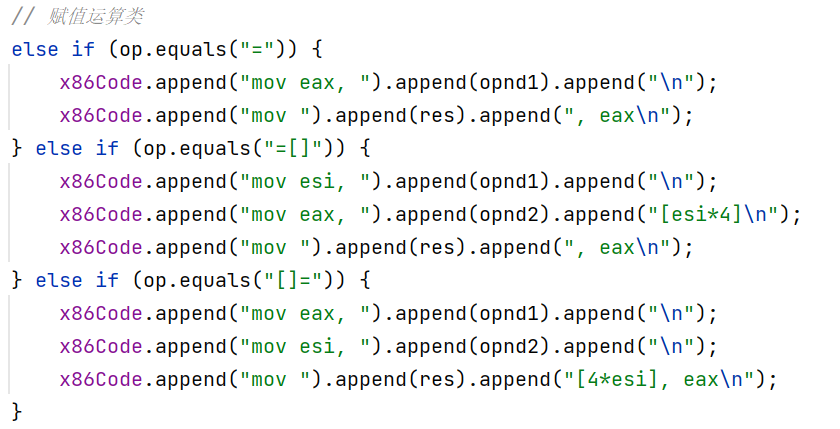
算术运算类相关的四元式op包括+、-、\*、/、%、+=、-=、\*=、/=、%=。对于加、减、乘，生成汇编指令的结构相似，只是运算指令不同。对于除、取模指令，由于汇编指令特性，32位寄存器的源操作数为“edx:eax”，商保存在eax中，余数保存在edx中，据此编写汇编指令即可。生成汇编指令如下。



单目运算类相关的四元式op包括++、--。对应生成自增、自减指令即可。生成汇编指令如下。



赋值运算类相关的四元式op包括=、[]=、=[]。普通赋值=只需将源操作数移入寄存器、寄存器移入目的操作数即可。[]=含义是取数组值，换算后的地址保存在opnd1中，取出的值保存在opnd2中。=[]含义是数组赋值，赋值保存在opnd1中，换算的地址保存在opnd2中。生成汇编指令如下。

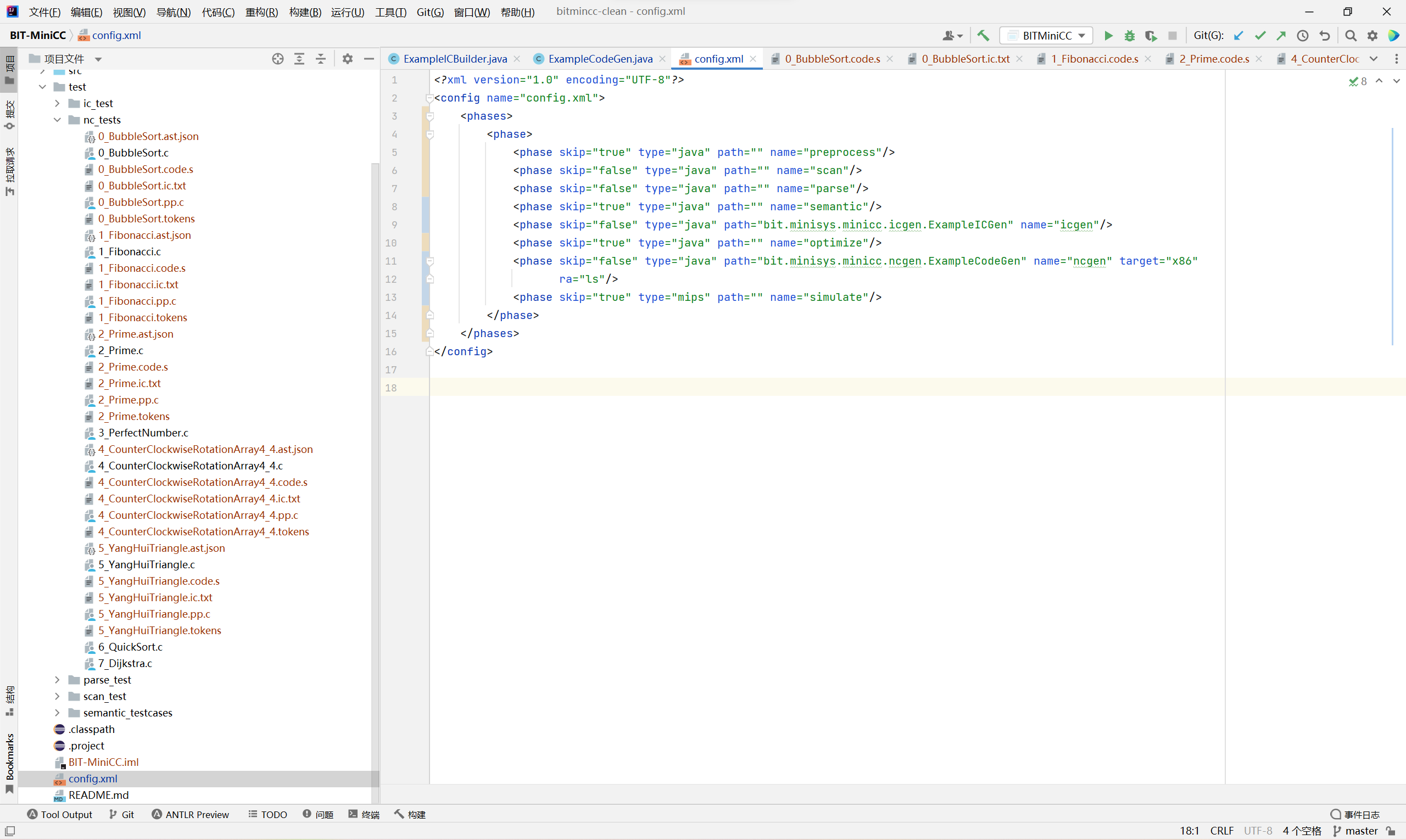


函数调用类相关的四元式op包括arg、call。对函数参数arg，按从右向左顺序压入一个argStack栈中。对函数调用语句call首先特判生成Mars\_PrintStr、Mars\_PrintInt、Mars\_GetInt对应的汇编指令。对其他函数调用，用invoke伪指令生成，存在参数时argStack逆序出栈至栈空，依次拼接到invoke伪指令后即可。对有返回值的函数，还需要将返回值从eax中取出，保存在result中。编写代码如下。



## 5.3 修改配置文件，生成汇编代码

修改配置文件中ncgen的skip=false、path为编写的目标代码生成文件、target=x86，配置文件如下图。

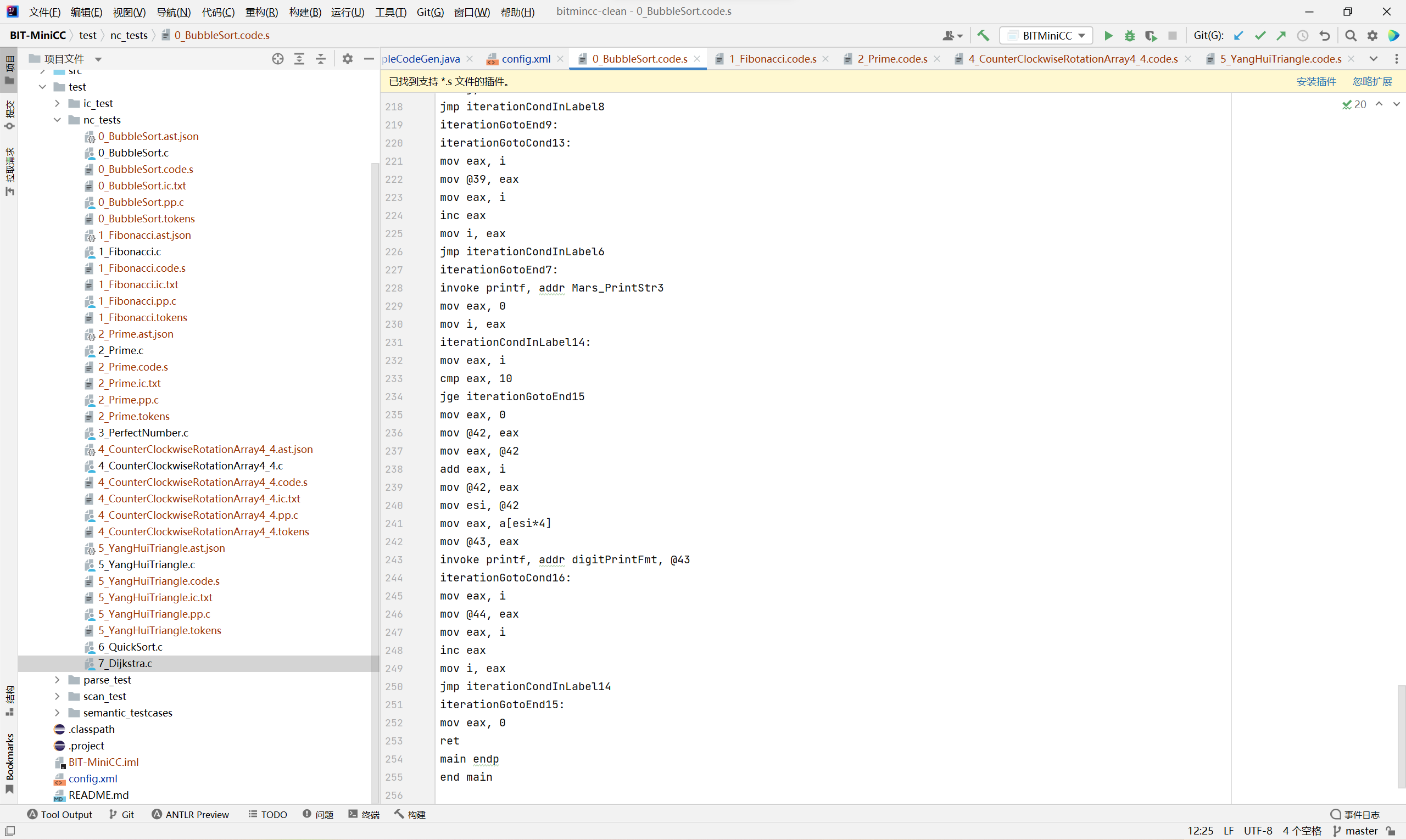


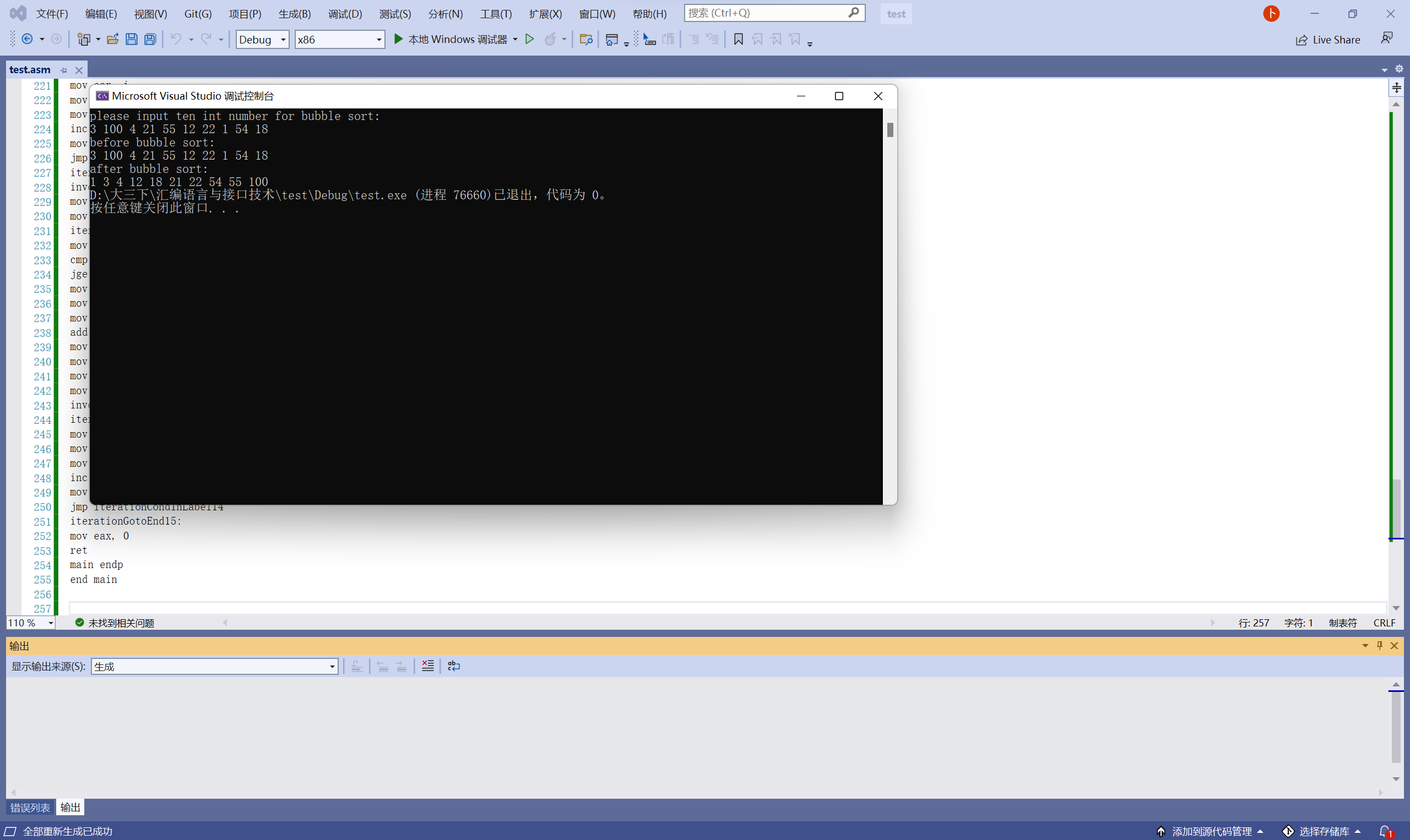
对需要测试的.c文件，将文件路径写入编译配置中即可。

# 6. 实验结果与分析

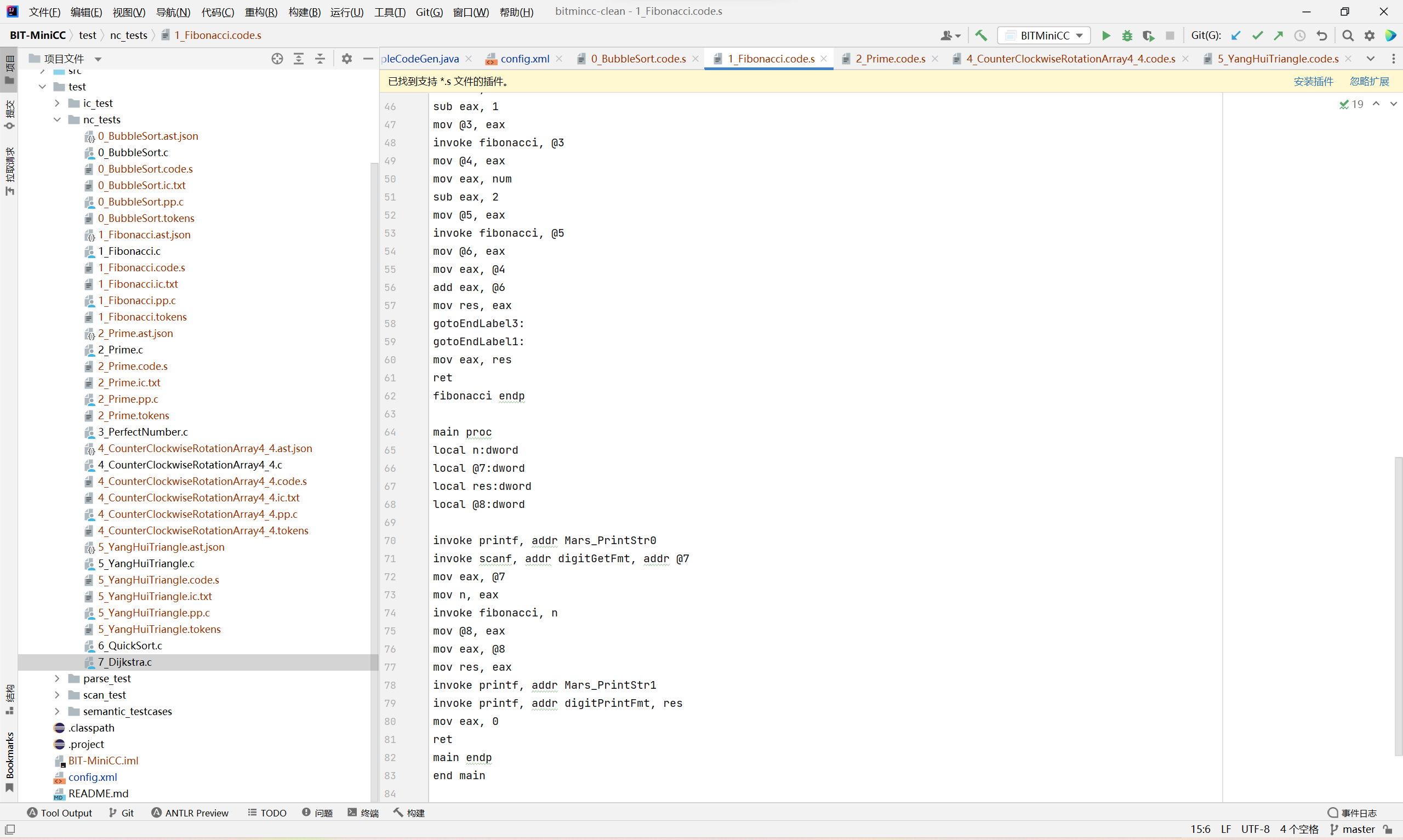
本实验已通过的测试用例有5个，分别是0\_BubbleSort.c、1\_Fibonacci.c、2\_Prime.c、4\_CounterClockwiseRotationArray4\_4.c、5\_YangHuiTriangle.c。对未通过的测试用例，3\_PerfectNumber.c因为存在局部变量时汇编关键字的问题，需要额外编写函数名转换的方法。

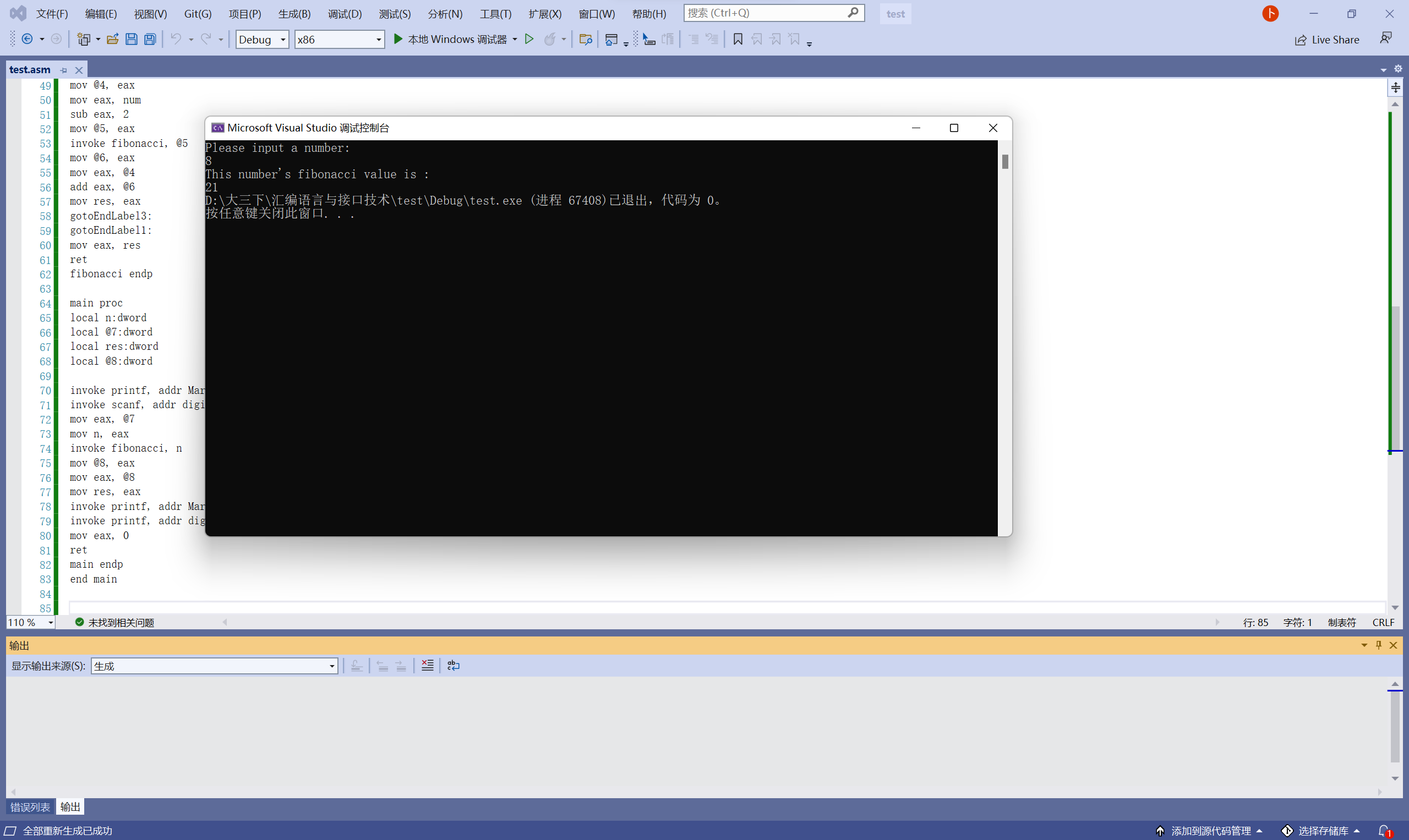
0\_BubbleSort.c生成的汇编代码部分、在masm32上的运行结果如下。



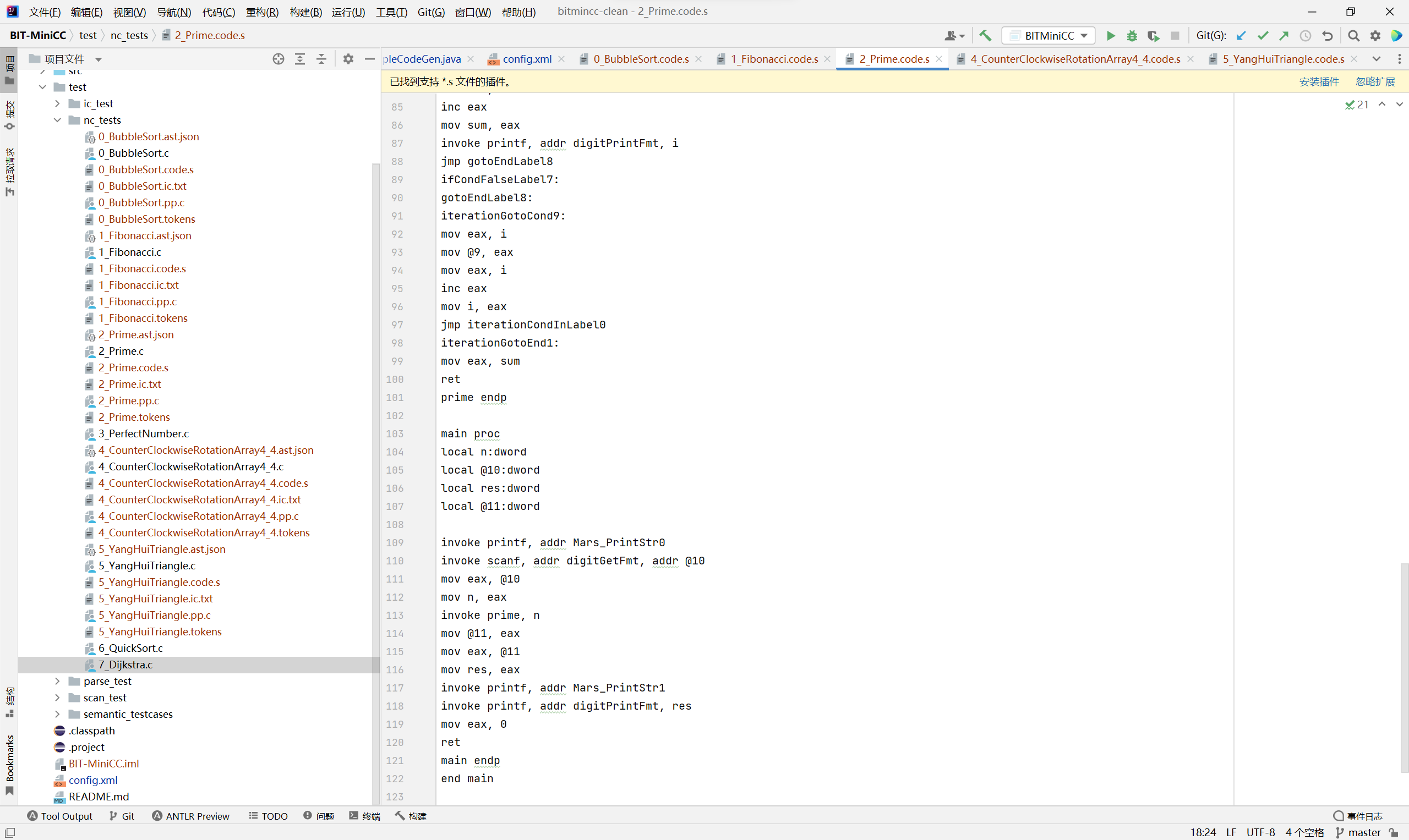


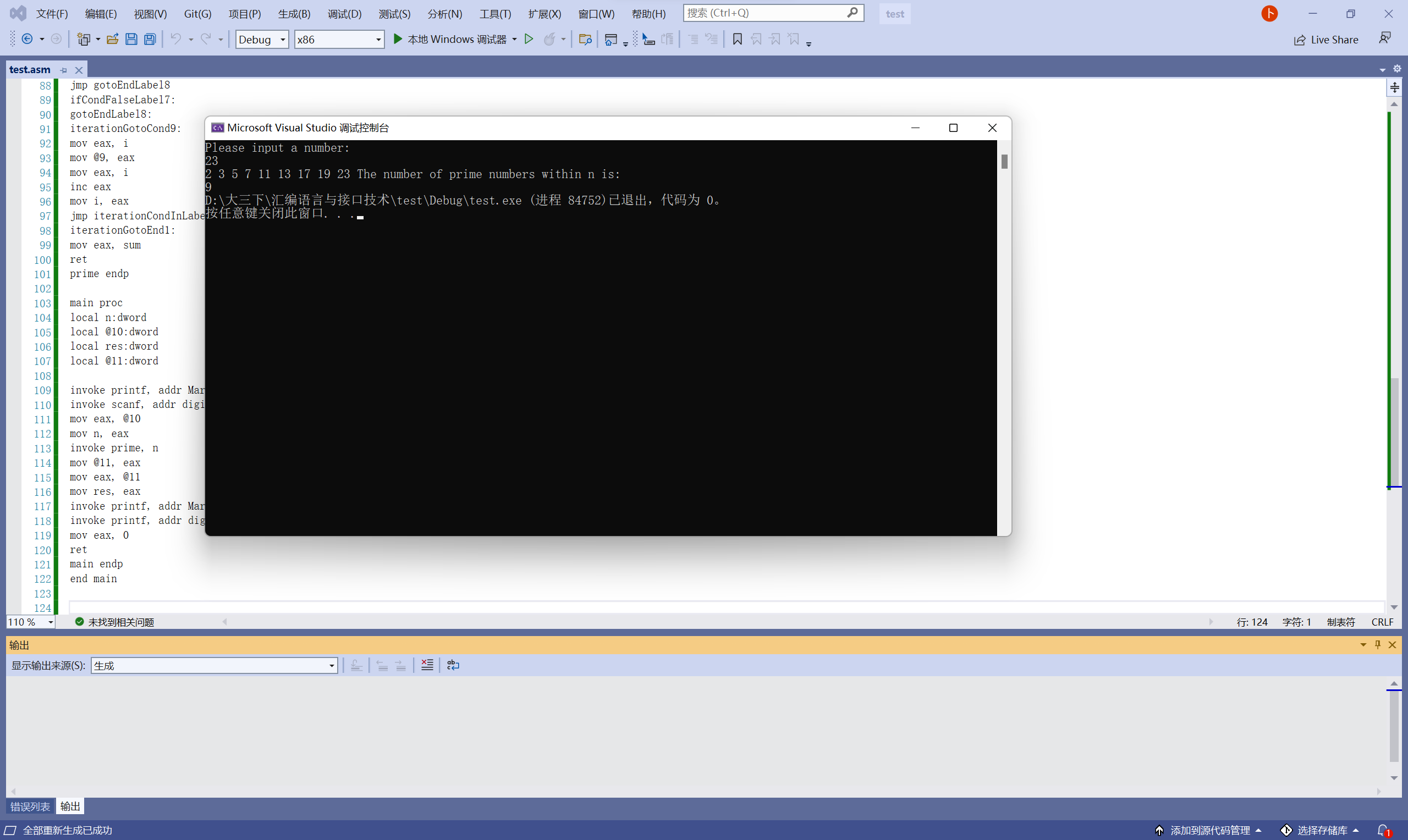
1\_Fibonacci.c生成的汇编代码部分、在masm32上的运行结果如下。



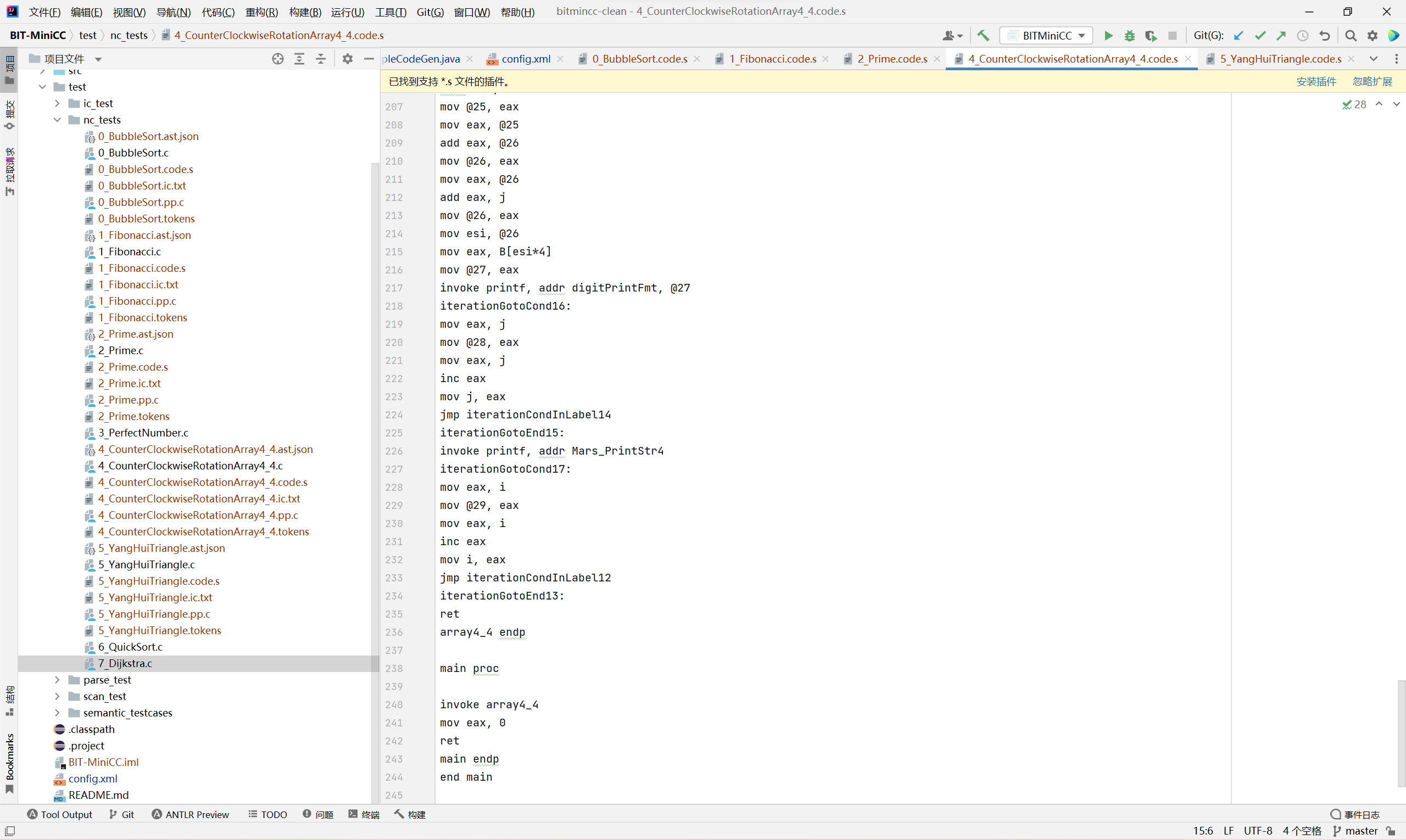


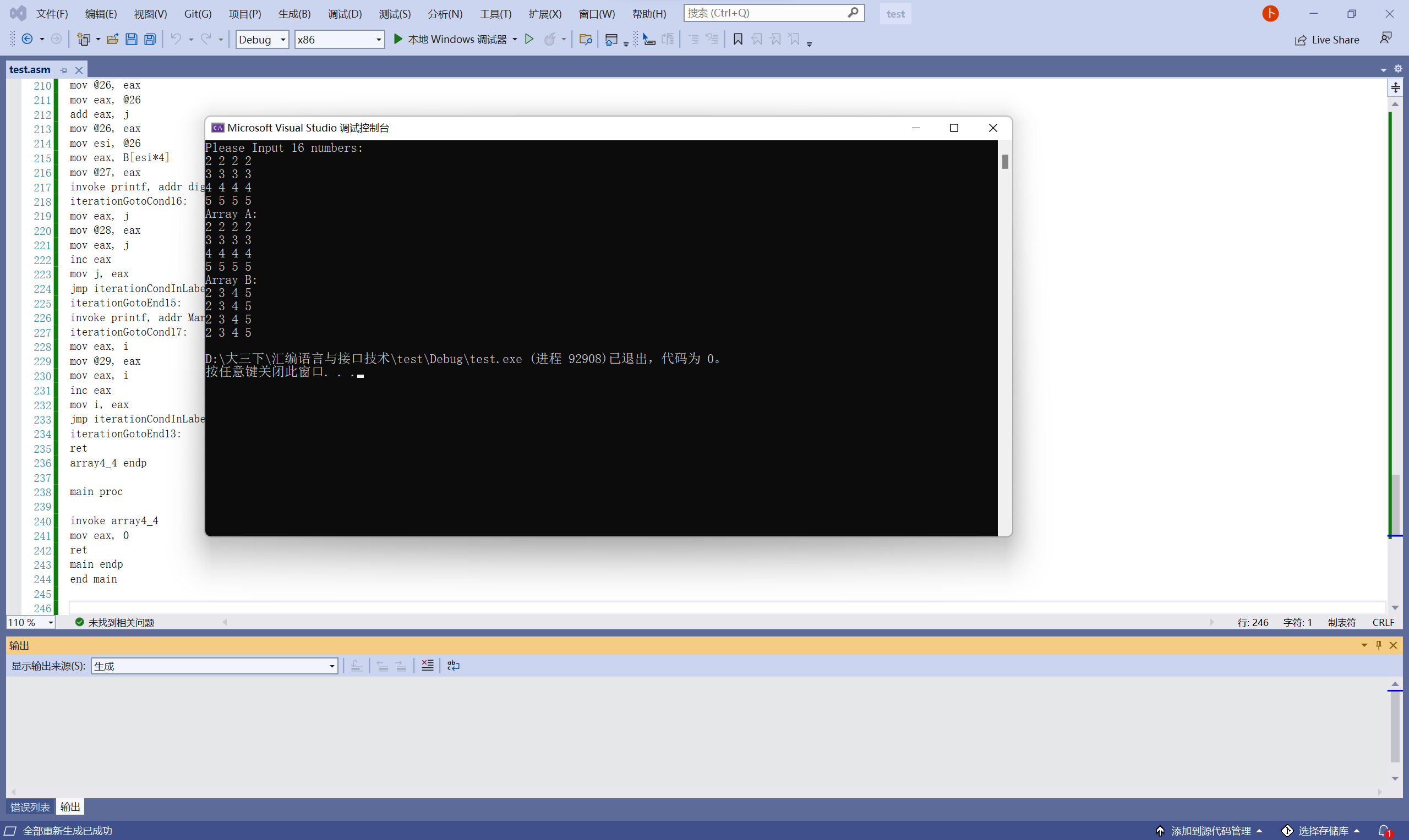
2\_Prime.c生成的汇编代码部分、在masm32上的运行结果如下。



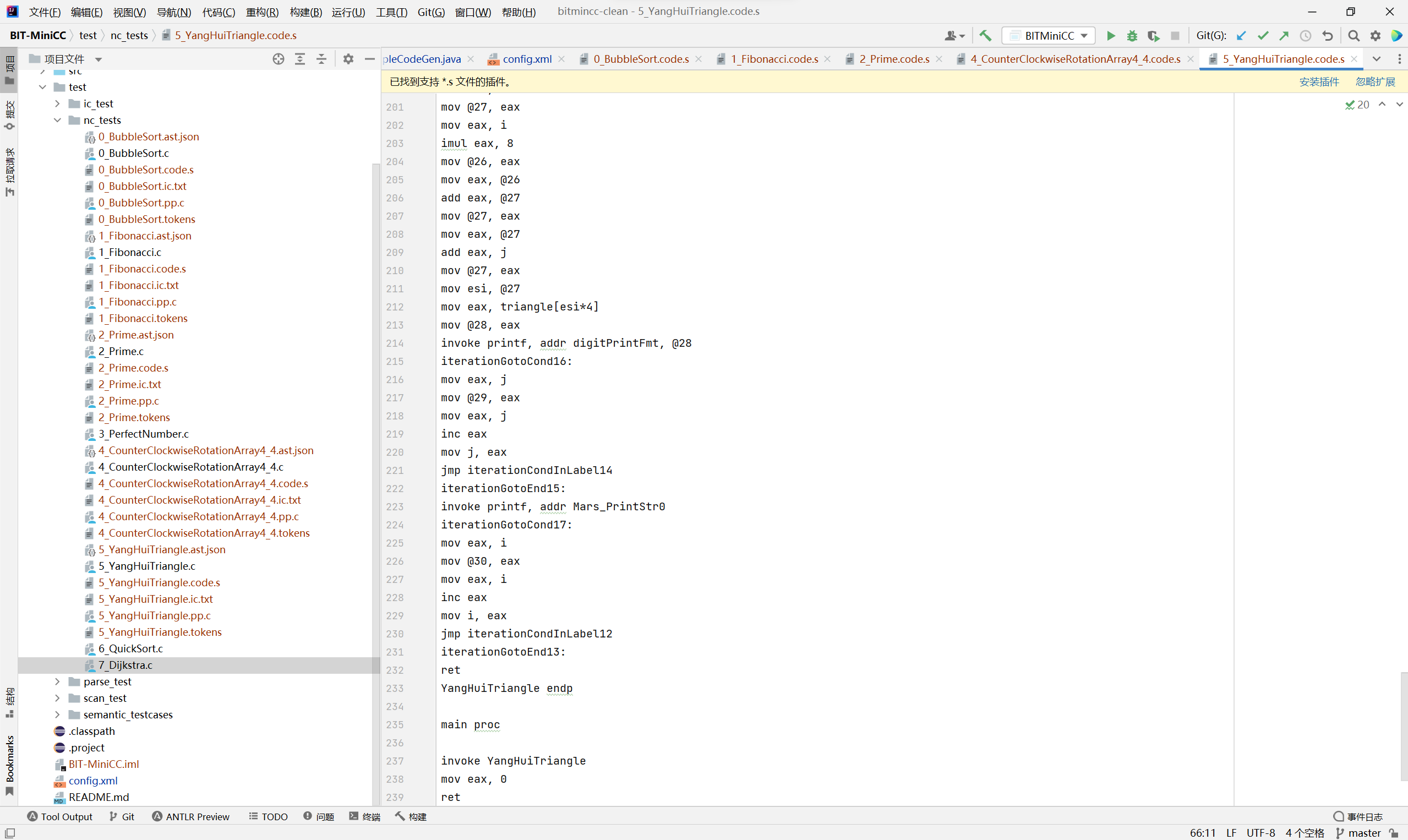


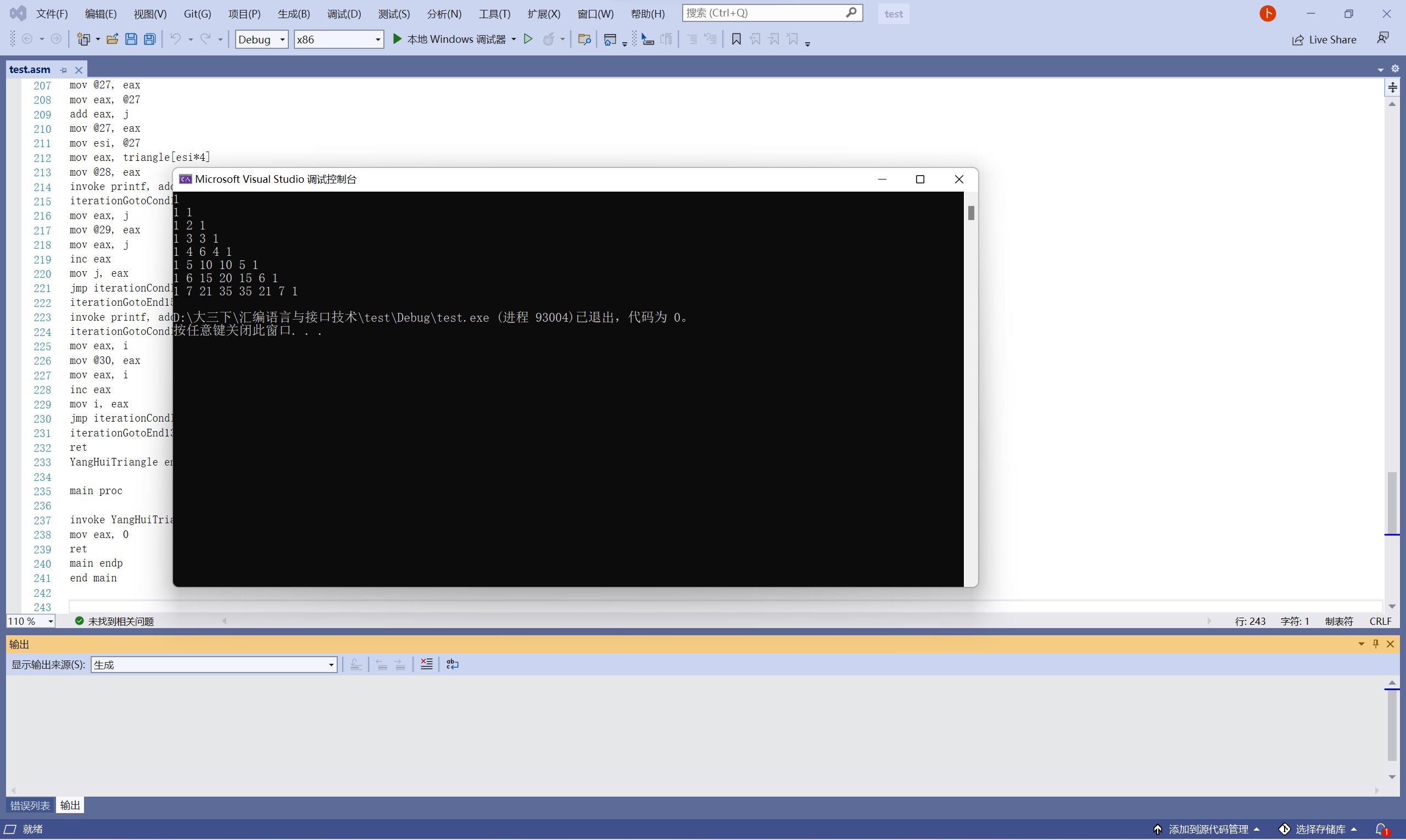
4\_CounterClockwiseRotationArray4\_4.c生成的汇编代码部分、在masm32上的运行结果如下。





5\_YangHuiTriangle.c生成的汇编代码部分、在masm32上的运行结果如下。





通过以上图片可知，生成的汇编代码能够运行并得到正确结果。

# 7. 个人心得体会

本次实验是编译原理的最后一个实验，相比前几个实验，难度上基本持平。本实验难点在于编写四元式转换成汇编语言的代码，这需要结合汇编语言所学的知识。同时，中间代码的质量也决定了本次实验的任务量，在编写目标代码生成时，我也多次反工修改中间代码生成的部分内容，以使的目标代码生成更合理、简单。本次实验我的收获主要有以下几点。

（1）掌握了目标代码生成的方法，熟悉了多种寄存器分配算法，能够编写目标代码生成器，并将之嵌入到BITMiniCC中，在四元式的基础上生成能正确执行的x86汇编代码。

（2）复习了汇编语言相关知识，能够熟练掌握四元式与汇编语言之间的转换。