

编译原理

算符优先语法分析设计原理与实现

学 院：

专 业：

学生姓名：

学 号：

北京交通大学

2023年10月

**实验要求**

## 1. 理论传授

编译器后端的设计，编译器的设计，理解编译程序的整体概念，目标代码生成的基本概

念和实现方法。

## 2. 目标任务

**[实验项目]** 在专题 6 编译器前端实现的基础上，将所生成的中间代码翻译成某种目标

机的汇编代码，实现编译器后端实现的任务，再进一步实现程序的输入是源程序，输出是汇

编语言目标代码小型实验语言编译器的任务。生成目标代码的方法可手工编程实现，也可以

查找生成工具完成。

**[设计要求]（**1）编译器后端的输入为中间代码序列，输出为汇编代码序列；（2）小型实验语言编译器的输入是字符串形式的源程序，输出是汇编语言目标程序；（3）设计两个测试用例（尽可能完备），并给出程序执行结果。

3. 任务分析

可通过资料查找的方式，研究和了解编译器后端的手工编程实现方式及借助生成工具的实现方式，拓展学习目前各大公司程序设计语言编译程序的实现方式，深入理解编译原理和编译程序总体结构，提高驾驭程序设计语言的能力，提高复杂问题求解能力，提高工程实践能力

## 一、程序功能描述

本程序目的是将所生成的中间代码翻译成某种目标机的汇编代码，实现编译器后端实现的任务，即编译器后端的输入为中间代码序列，输出为汇编代码序列；

为了输出汇编代码，首先需要进行四元式的输出，然后再进行汇编指令的输出。这两步骤成为翻译。

## 二、主要数据结构

**定义四元式结构体**

**struct Four\_exp**{ //四元式结构体

string op;

string r1, r2;

string left;

};

**定义优先级**，并进行数字的赋值代表优先级高低。

**void init\_pri()**{ //优先级定义

pri["("] = 10;

pri["++"] = pri["--"] = 9;

pri["\*"] = pri["/"] = pri["%"] = 8;

pri["+"] = pri["-"] = 7;

pri[">"] = pri["<"] = pri[">="] = pri["<="] = 6;

pri["!="] = pri["=="] = 5;

pri["&&"] = 4;

pri["||"] = 3;

pri["="] = 2;

pri[")"] = pri["@"] = 1;

}

**void get\_token()**;//获取四元式

**bool expect(string Expect);**//匹配函数

如果type和与其相同则进行四元式获取，并返回true

**string itos(int x)**;//该函数是定义一个流并且输出字符string类型

**string Caculate(string stop)**;表达式处理

首先定义一些变量来存放拆解后的式子，并且定义两个栈来存放数字和符号。首先检验表达式是否结束，判断方法为判断当前指针指向内容是否为左右括号。如果是的话则进行左右移动。

其次是表达式处理。当当前指针指向变量或者常量的时候就压入num栈中，并且判断是否为id和number。

当判断为运算符时，取出符号栈中top的值，当当前优先级小于取出的优先级且不为左右括号的时候进行如下判断：（1）如果是结束符，则返回num栈中的顶部，并将opr栈该符号弹出（2）如果为（时，则跳过（3）如果为单目运算符，则num栈中弹出，作为四元式里的数字部分（4）如果为双目运算符，则两个数字依次弹出，并放入四元式的2,3个位置中作为数字部分（数字部分就是四元式中的源操作数）。

**void fuzhi(pro left, string stop)//赋值语句**

这里是进行等号的判断。如果遇到了赋值语句即判断出来有等号，则输出对应四元式。

**bool body()；//该函数存放函数快**

这个函数用来对函数里面的函数块进行处理。比如把id，cin，>>，保留字，cout,;，<<等进行处理并放入四元式。

例如：把id放入变量后，当遇到cin和遇到>>则输入四元式：{"cin", "\_", "\_", last.value})，直到遇到分号作为结尾；

cout同样，四元式为{"cout", "\_", "\_", last.value}

**void four\_exp\_to\_file()；//输出四元式**

本部分的代码就是将上述所有的四元式进行输出。由于上面的four\_exp是向量，因此可以进行向量的输出。

for(int i=0; i<n; i++){

out<<"("<<i<<")"<<" ( "<<four\_exp[i].op<<" , "<<four\_exp[i].r1<<" , "<<four\_exp[i].r2<<" , "<<four\_exp[i].left<<" )"<<endl;

}

#### 汇编指令描述

从四元式到汇编进行转换。

首先四元式已经存入了four\_exp向量中。然后对其中的四元式进行依次翻译。

**void to\_assemble()**

（1）四元式中有=；即用MOV指令

Four\_exp{"MOV", four\_exp[i].left, four\_exp[i].r1, "\_"}

这条指令就是把源操作数从第二个移到第一个。

（2）四元式中有cin，使用IN指令

Four\_exp{"IN", four\_exp[i].left, "\_", "\_"}

这条指令是输入操作，把输入内容输入至寄存器中。

（3）四元式有cout，使用OUT指令

Four\_exp{"OUT", four\_exp[i].left, "\_", "\_"}

这条指令是输出操作，把内容输出至相应寄存器中。

（4）四元式有+，使用MOV,ADD,MOV指令

Four\_exp{"MOV", "R" + itos(temp), four\_exp[i].r1, "\_"}

Four\_exp{"ADD", "R" + itos(temp), four\_exp[i].r2, "\_"}

Four\_exp{"MOV", four\_exp[i].left, "R" + itos(temp), "\_"}

这条指令是加操作，需要使用的指令有ADD，是将两个源操作数相加，第一个MOV是将第一个源操作数移动进入R寄存器中，第二个MOV是将结果移入寄存器中。

（5）四元式有-

Four\_exp{"MOV", "R" + itos(temp), four\_exp[i].r1, "\_"}

Four\_exp{"SUB", "R" + itos(temp), four\_exp[i].r2, "\_"}

Four\_exp{"MOV", four\_exp[i].left, "R" + itos(temp), "\_"}

这条指令是减操作，需要使用的指令有SUB，是将两个源操作数相减，第一个MOV是将第一个源操作数移动进入R寄存器中，第二个MOV是将结果移入寄存器中。

（6）四元式有\*

Four\_exp{"MOV", "R" + itos(temp), four\_exp[i].r1, "\_"}

Four\_exp{"MUL", "R" + itos(temp), four\_exp[i].r2, "\_"}

Four\_exp{"MOV", four\_exp[i].left, "R" + itos(temp), "\_"}

这条指令是乘操作，需要使用的指令有MUL，是将两个源操作数相乘，第一个MOV是将第一个源操作数移动进入R寄存器中，第二个MOV是将结果移入寄存器中。

（7）四元式有/

Four\_exp{"MOV", "R" + itos(temp), four\_exp[i].r1, "\_"}

Four\_exp{"DIV", "R" + itos(temp), four\_exp[i].r2, "\_"}

Four\_exp{"MOV", four\_exp[i].left, "R" + itos(temp), "\_"}

这条指令是除操作，需要使用的指令有DIV，是将两个源操作数相除，第一个源操作数除以第二个源操作数。第一个MOV是将第一个源操作数移动进入R寄存器中，第二个MOV是将结果移入寄存器中。

（8）四元式有%

Four\_exp{"MOV", "R" + itos(temp), four\_exp[i].r1, "\_"}

Four\_exp{"MOD", "R" + itos(temp), four\_exp[i].r2, "\_"}

Four\_exp{"MOV", four\_exp[i].left, "R" + itos(temp), "\_"}

这条指令是取余操作，需要使用的指令有MOD，是将两个源操作数进行取余，第一个源操作数除以第二个源操作数。第一个MOV是将第一个源操作数移动进入R寄存器中，第二个MOV是将结果移入寄存器中。

（9）四元式有>

Four\_exp{"CMP", four\_exp[i].r1, four\_exp[i].r2, "\_"}

Four\_exp{"JG", "(" + itos(assemble.size() + 2) + ")", "\_", "\_"}

Four\_exp{"MOV", four\_exp[i].left, "0", "\_"}

Four\_exp{"MOV", four\_exp[i].left, "1", "\_"}

这条指令是CMP和JG指令（即进行判断和跳转操作）CMP是进行两个源操作数的大小判断，本指令是>，则对应的是JG指令，然后JG指令对应的是括号内，将四元式中+2位置转化成string类型。判断是否为>，如果是的话进行1的MOV，不是的话为0的MOV。

（10）四元式有<

Four\_exp{"CMP", four\_exp[i].r1, four\_exp[i].r2, "\_"});

Four\_exp{"JNG", "(" + itos(assemble.size() + 2) + ")", "\_", "\_"});

Four\_exp{"MOV", four\_exp[i].left, "0", "\_"});

Four\_exp{"MOV", four\_exp[i].left, "1", "\_"});

这条指令是CMP和JNG指令（即进行判断和跳转操作）CMP是进行两个源操作数的大小判断，本指令是<，则对应的是JNG指令，然后JNG指令对应的是括号内，将四元式中+2位置转化成string类型。判断是否为<，如果是的话进行1的MOV，不是的话为0的MOV。

（11）四元式有J

Four\_exp{"JMP", "(" + itos(foo) + ")", "\_", "\_"}

这部分为跳转指令JMP

（12）四元式有J=

Four\_exp{"CMP", four\_exp[i].r1, four\_exp[i].r2, "\_"});

Four\_exp{"JZ", "(" + itos(foo) + ")", "\_", "\_"});

本条指令为CMP和JZ

**汇编指令输出**

fstream是定义输出流，然后打开文件进行输出。

fstream out;

out.open("assemble.txt", ios::out);

for(int i=0; i<len; i++){

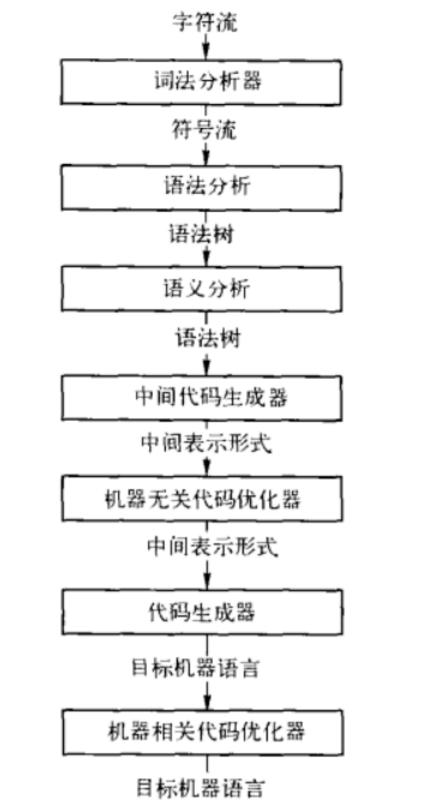
if(assemble[i].op[0] == 'J' || assemble[i].op == "IN" || assemble[i].op == "OUT") out<<"("<<i<<")"<<" "<<assemble[i].op<<" "<<assemble[i].r1<<endl;

else out<<"("<<i<<")"<<" "<<assemble[i].op<<" "<<assemble[i].r1<<" , "<<assemble[i].r2<<endl;

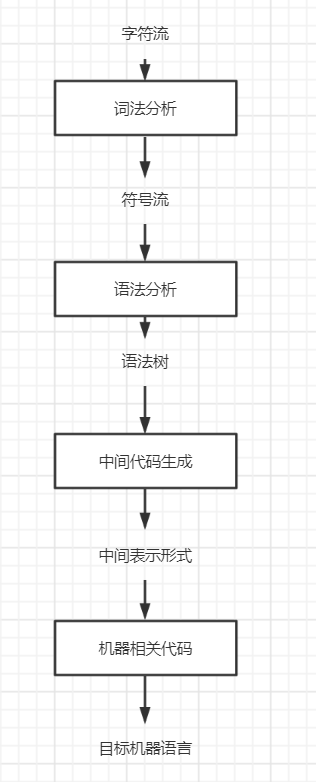
}

### 三、程序结构描述

### 实验性语言编译器总体结构；



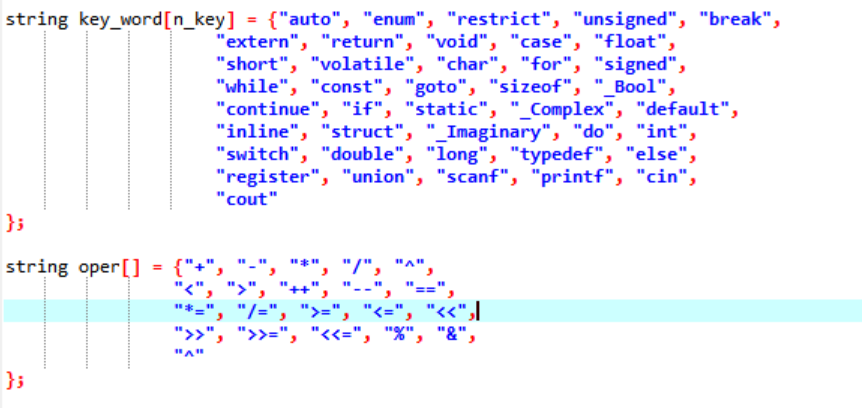
一个编译器的步骤如上。本实验构建的编译器依据此图进行设计和分析。由于优化包含在分析过程当中，因此本实验的流程和结构如下所示：



本实验分为具体以下几个部分：

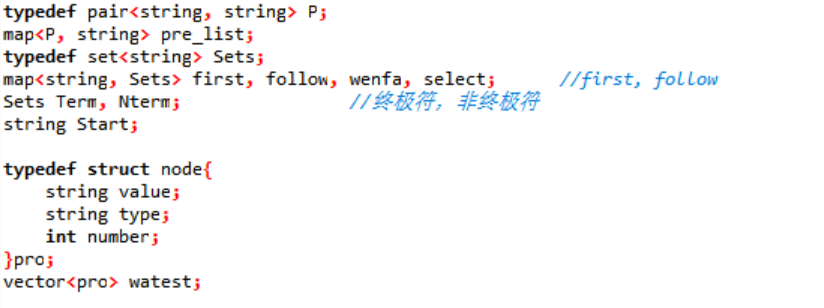
预处理阶段

定义保留字和字符，构建符号表



#### 词法分析

定义如下：进行first，fellow集的定义（使用map），终结符非终符以及产生式的结构体定义等。



词法分析的流程如下：

（1）对正常内容的判断。当判断为相应的保留字或者内容时，则输出相对应的内容。以括号内显示的方法表现。呈现出状态转换图的样子。

（2）错误判断

这里采用判断输入内容来判断错误。如果不是在上述规则内未找到对应的输出，则判断为出错。因此判断出错的规则为未定义则判断错误。

（3）注释判断

在读函数的阶段采用ch来获取文件中每个字符。如果读取到/则为单行注释，如果读取到\*则判断为多行注释。并进行继续读入处理，直到读到换行符或者另一个

（4）保留字和标识符处理

用C语言自带的isalpha和isdigit来判断是标识符还是保留字。然后进行不同处理，再使用OUT进行输出。

#### 语法分析

本过程需要进行first和fellow集以及select集的算法设计。本部分设计参考实验2,3,4，重点参考3构建LL（1）分析器。

构建方式如下：

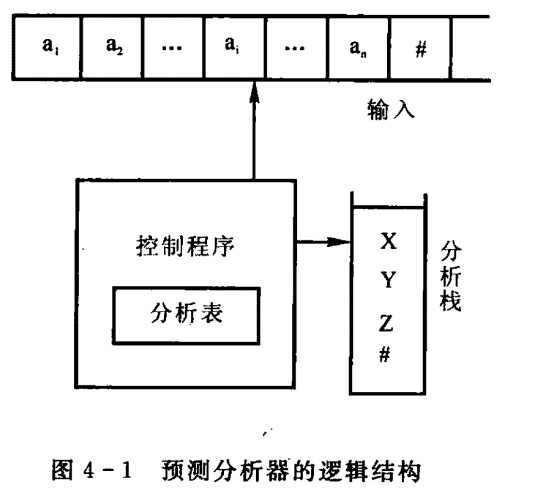
**FIRST/FELLOW集输出**

此处产生式直接使用实验所给产生式，首先存放每个非终结符号的FIRST集合，然后存放，遍历单个产生式的左边，listCell为“|”分割出来，产生式左边用“ | ”分割的单个式子的First(弃用)，标记是否有定义为空,如果有则检查下一个字符。开始符号不能推出空，如果为空，可以查询下一个字符非空的终结符号加入对应的FIRST集合，不是终结符号，入栈。

**LL（1）分析表**

按照大小关系进行分析表的输入，本程序中，使用数字进行分析表的书写。

**LL（1）分析器**



根据栈顶符号Xm和正在扫视的输入符号ai，执行以下三个动作之一：

（1）若Xm∈Vt∪{#}，且Xm=ai，则表名栈顶符号已经与当前正扫视的输入符号（包括句尾标志符号#在内）相匹配，此时应将Xm从栈中退出，并将输入串指示器向前推进一个位置，否则进行语法错误处理。

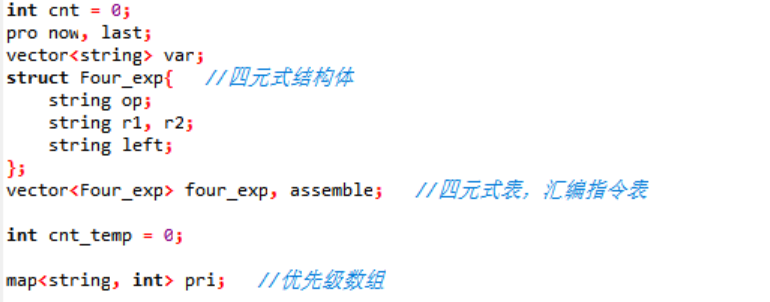
（2）若Xm属于Vn，则以符号对（Xm，ai）查分析表，设表元素A[Xm,ai]为产生式Xm->Y1Y2......Yk，则将Xm从栈中退出，并将Y1Y2.....Yk按反序推入栈中（即用该产生式推导一步），从而得到如下格局：



但若A[Xm,ai]为“出错”，则进行语法错误处理；

（3）若Xm=ai=#（即分析栈将被拆空）则表明输入串已完全得到匹配，此时可宣告分析成功。结束

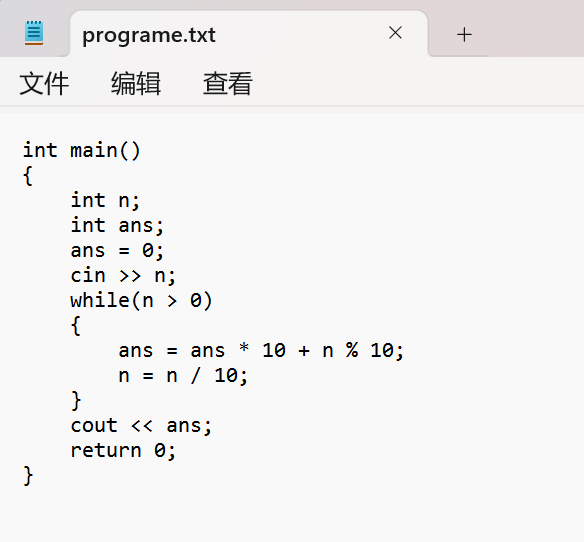
#### 中间代码生成及机器相关代码生成



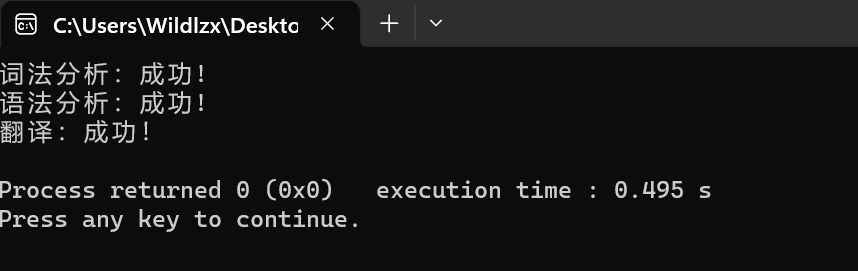
详细分析见5.1中四元式的产生以及汇编代码的生成。

## 四、实验结果与测试

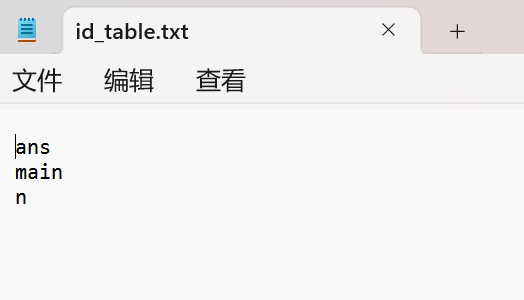
运行main函数，实现词法分析，语法分析和翻译。需要分析的代码如下所示：

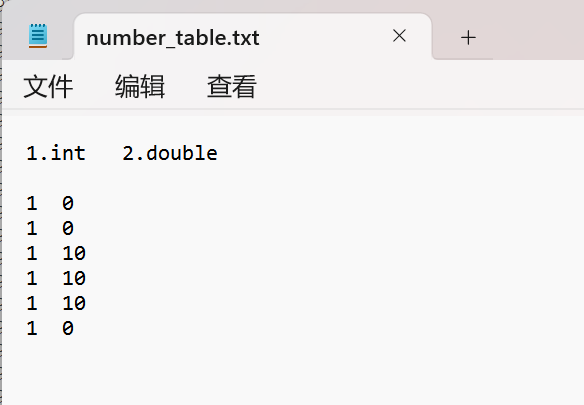


分析是否成功，结果输出在控制台上：

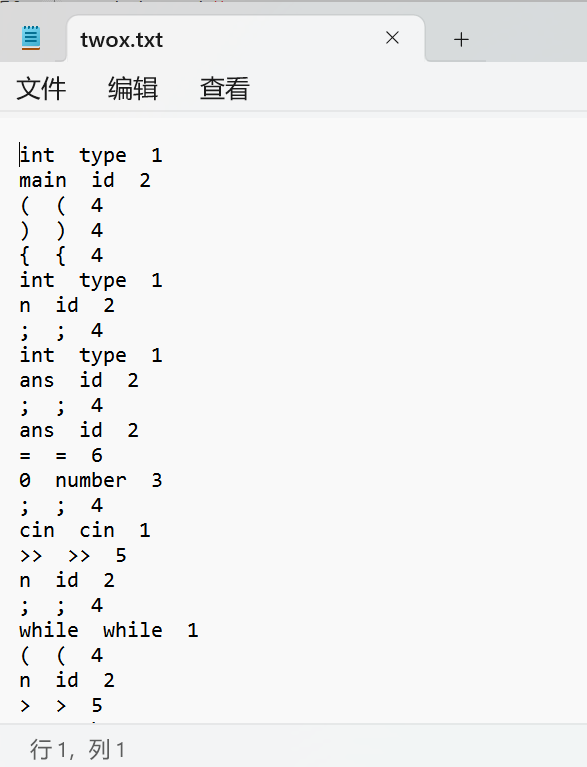


保留字，变量和常量的分离：

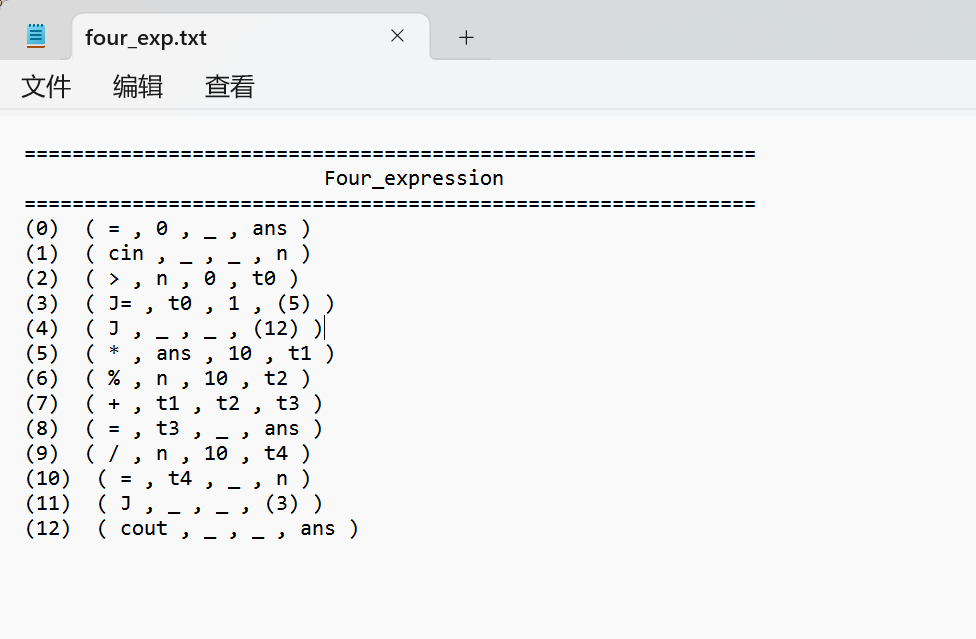




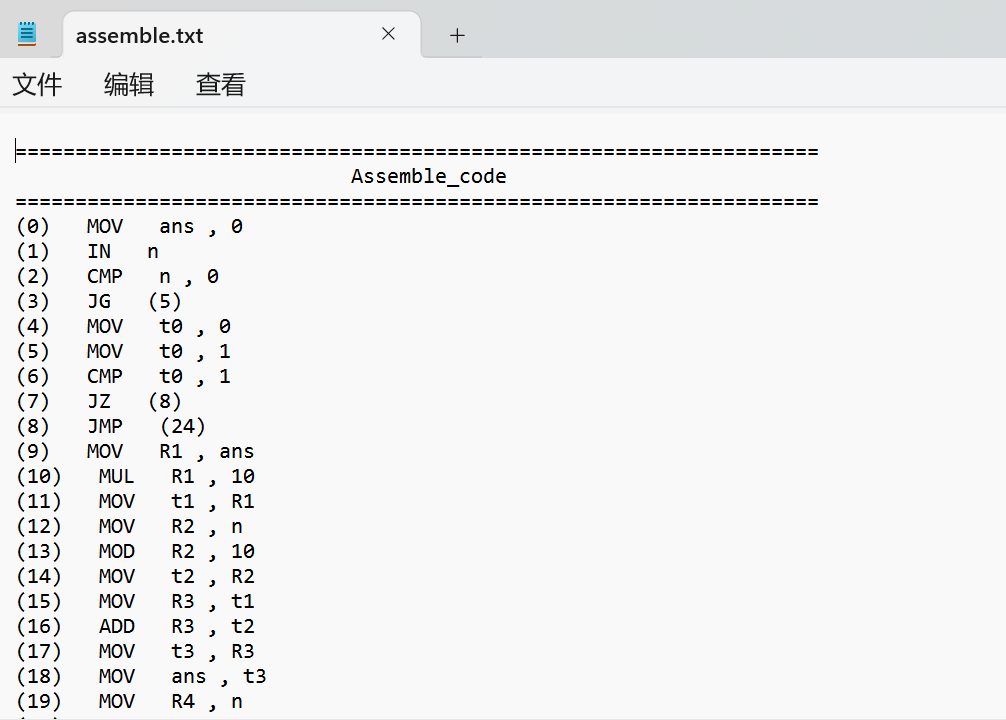
保留字和非终结符号的分类和转换

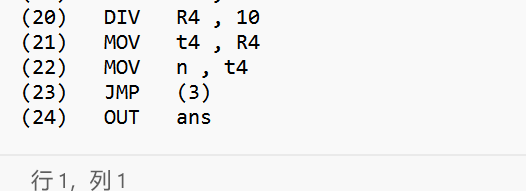


四元式的输出：



汇编指令输出：





**五、研究性教学实验感受**

七个实验做完之后，我对编译原理这门课有了更深刻的体会。  
 编译程序的基本任务就是将源语言程序翻译成等价的目标语言程序。是一个对目标代码进行词法分析，语法分析，生成中间代码，中间代码再转化成机器代码。编译程序是计算机的基本组成部分之一；从功能上看，一个编译程序就是一个语言翻译程序。经过整个七个实验的设计分析和实现，更加深刻清晰地明确了整个编译的过程和流程。在实验过程中遇到了很多问题，都结合着流程图以及课上所学知识进行了理解和体会。

在七次实验过程中对整个编译过程有了更深刻的理解。实验七将前面所有内容相结合，构建了一个小型的编译器。这几次实验极大地提高了代码工程能力。