



中国矿业大学
CHINA UNIVERSITY OF MINING AND TECHNOLOGY

中国矿业大学计算机学院 2017级本科生课程设计报告

实验六 目标代码生成

课程名称	系统软件开发实践
报告时间	2020.05.13
学生姓名	李治远
学 号	07172757
专 业	计算机科学与技术
任课教师	席景科老师

目录

1	实验内容	1
2	实验任务	1
3	编译器原理	1
3.1	编译器	1
3.2	编译器的各阶段	1
4	后端代码设计	2
4.1	相关代码	2
4.2	实验步骤	19
5	计算器后端实验	23
5.1	修改代码	23
5.2	中间代码汇编代码生成	32
5.3	emu8086验证代码	34
6	错误处理	35
7	图形GUI	36

实验六 目标代码生成

1 实验内容

是在词法分析、语法分析、语义分析和中间代码生成程序的基础上，将C子集源代码翻译为MIPS32指令序列（可以包含伪指令），并在SPIM Simulator或翻译为x86指令序列并在x86 Simulator上运行。

2 实验任务

- 确定目标机的体系结构后，选择一款模拟器进行安装学习；
- 学习与具体体系结构相关的指令选择、寄存器选择以及存储器分配管理等技术。
- 编写目标代码的汇编生成程序。该程序的功能是可以将在之前已经得到的中间代码映射为相应的汇编代码。
- 要求你的程序能输出正确的汇编代码。
- “正确”是指该汇编代码在相应模拟器上运行结果正确。

3 编译器原理

3.1 编译器

编译器（compiler）是一种计算机程序，它会将某种编程语言写成的源代码（原始语言）转换成另一种编程语言（目标语言）。

它主要的目的是将便于人编写、阅读、维护的高级计算机语言所写作的源代码程序，翻译为计算机能解读、运行的低阶机器语言的程序，也就是可执行文件。编译器将原始程序（source program）作为输入，翻译产生使用目标语言（target language）的等价程序。源代码一般为高级语言（High-level language），如Pascal、C、C++、C#、Java等，而目标语言则是汇编语言或目标机器的目标代码（Object code），有时也称作机器代码（Machine code）。

一个现代编译器的主要工作流程如下：

源代码（source code）→ 预处理器（preprocessor）→ 编译器（compiler）→ 汇编程序（assembler）→ 目标代码（object code）→ 链接器（linker）→ 可执行文件（executables），最后打包好的文件就可以给电脑去判读运行了。

3.2 编译器的各阶段

第一阶段：词法分析词法分析的任务是：输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个个的单词（亦称单词符号或简称符号），如基

本字 (begin、end、if、for、while)，标识符、常数、运算符和界符 (标点符号、左右括号)。

单词符号是语言的基本组成成分，是人们理解和编写程序的基本要素。识别和理解这些要素无疑也是翻译的基础。如同将英文翻译成中文的情形一样，如果对英语单词不理解，那就谈不上进行正确的翻译。在词法分析阶段的工作中所依循的是语言的词法规则 (或称构词规则)。描述词法规则的有效工具是正规式和有效自动机。

第二阶段：语法分析语法分析的任务是：在词法分析的基础上，根据语言的语法规则，把单词符号串分解成各类语法单位 (语法范畴)，如“短语”、“句子”、“程序段”和“程序”等。通过语法分析，确定整个输入串是否构成语法上正确的“程序”。语法分析所依循的是语言的语法规则。语法规则通常用上下文无关文法描述。词法分析是一种线性分析，而语法分析是一种层次结构分析。例如： $Z = X + 0.618 * Y$;

代表一个“赋值语句”，而其中的 $X + 0.618 * Y$ 代表一个“算术表达式”。因而，语法分析的任务就是识别 $X + 0.618 * Y$ 为算术表达式。

第三阶段：语义分析与中间代码生成语义分析阶段的任务是：依据源语言限定的语义规则对语法分析所识别的语法范畴进行语义检查并分析其含义，初步翻译成与其等价的中间代码。语义分析是整个编译程序完成的最具实质的性的翻译任务。

第四阶段：代码优化代码优化是为了改进目标代码的质量而在编译过程中进行的工作。代码优化可以在中间代码级或目标代码级进行，其实质是在不该比那源程序语义的基础上对其进行加工变换，以期获得更高效的目标代码。而高效一般是对所产生的目标程序在运行时间上的缩短和存储空间上的节省而言的。

第五阶段：目标代码生成目标代码生成作为编译程序的最后阶段，其任务是：根据中间代码及编译过程中产生的各种表格的有关信息，最终生成所期望的目标代码程序，一般为特定机器的机器语言代码或汇编语言代码。这个阶段实现了最后的翻译工作，处理过程较为繁琐，需要充分考虑计算机硬件和软件所提供的资源，以生成较高质量的目标代码。

4 后端代码设计

4.1 相关代码

```
// 头文件
#include<iostream>
#include<fstream>
#include<string>
#include<cctype>
#include<vector>
#include<stack>
#include<stdlib.h>
```

```
// 定义各种单词的key值
#define K_DIGIT      3      //整数
#define K_CHAR       4      //字符
#define K_STRING     5      //字符串
#define K_TYPE       6      //数据类型
#define K_KEYWORDS   7      //关键字
#define K_OPERATOR   8      //运算符
#define K_IDENTIFIER 9      //标识符
#define K_BRACKET    10     //括号

using namespace std;

//存储分词类型
typedef struct IDwords
{
    int      id;      //标志
    string   word;    //单词
}IDwords;

typedef struct Variable
{
    string   var;      //变量
    string   value;    //初始化值
}Variable;

//目标代码元素
typedef struct Target{
    string   dsf;      //结果
    string   op;       //操作
    string   dst;      //目的操作数
    string   dsc;      //源操作数
    string   mark;     //标志
    string   step;     //跳转位置
}Target;

//保存声明变量
vector<Variable>      var_table;
//保存目标代码
vector<Target>        target_code;

char                lab='A'; //记录跳转标志
char                vab='A'; //记录中间变量

//生成的汇编文件名称
string  asmfile(string source){
    if(source.size()==0){
        cout<<"源文件名不能为空"<<endl;
        exit(-1);
    }
}
```

```
    }
    string temp="";
    int i,j;
    j = source.size();
    for(i = j-1;i>=0;i--){
//        if(source[i] == '\\\' || source[i]== '/')
//            break;
        if(source[i] == '.'){
            j = i;
            break;
        }
    }
    temp = source.substr(0,j) + ".asm";
    return temp;
}
```

//运算符优先级

```
int level(string s){
    if(s=="#")
        return 1;
    else if(s=="=")
        return 2;
    else if(s=="+" || s=="-")
        return 3;
    else if(s=="*" || s=="/")
        return 4;
    else
        return -1;
}
```

//保存到目标代码

```
void add_target_code(string dsf,string op,string dst,string dsc,string mark,string step){
    Target tmp;
    tmp.dsf = dsf;
    tmp.op = op;
    tmp.dst = dst;
    tmp.dsc = dsc;
    tmp.mark = mark;
    tmp.step = step;
    target_code.push_back(tmp);
}
```

//字符转字符串

```
string char_to_str(char c){
    char s[2] = " ";
    s[0] = c;
    return string(s);
}
```

//是否为运算操作符

```
int is_operator(char c){
    if(c == '+' || c == '-' || c == '*' || c == '/' || c == ',' || c == '=' || c == '>' || c == '<')
        return 1;
    else
        return 0;
}
```

//是否为大括号、小括号、分号

```
int is_bracket(char c){
    if(c == '{' || c == '}' || c == '(' || c == ')' || c == ';')
        return 1;
    else
        return 0;
}
```

//是否为空白

```
int is_blank(char c){
    if(c == '\n' || c == '\t' || c == ' ' || c == '\r')
        return 1;
    else
        return 0;
}
```

//判断单词类型

```
int word_token(string s){
    int size = s.size();
    //字符数据
    if(s[0] == '\\'){
        if(s[size-1] == '\\')
            return K_CHAR;
        else{
            cout<<"错误的字符串数据: "<<s<<endl;
            exit(-1);
        }
    }
    //字符串数据
    else if(s[0] == '\"'){
        if(s[size-1] == '\"')
            return K_STRING;
        else{
            cout<<"错误的字符串数据: "<<s<<endl;
            exit(-1);
        }
    }
    //整数
    else if(isdigit(s[0])){
        for(int i=1;i<size;i++){
```

```

        if(!isdigit(s[i])){
            cout<<"不合法的标识符: "<<s<<endl;
            exit(-1);
        }
    }
    return K_DIGIT;
}else{
    for(int i=0;i<size;i++){
        if(!isalnum(s[i]) && s[i]!='_'){
            cout<<"不合法的标识符: "<<s<<endl;
            exit(-1);
        }
    }
    //数据类型
    if(s=="int" || s=="char")
        return K_TYPE;
    //关键字
    else if(s=="if" || s=="else" || s=="printf" || s=="main")
        return K_KEYWORDS;
    //自定义标识符
    else
        return K_IDENTIFIER;
}
}

//添加分词结果
void add_keywords(vector<IDwords> &v,int id,string word){
    IDwords    temp;
    temp.id = id;
    temp.word = word;
    v.push_back(temp);
}

//词法分析
void lexical_analysis(string source,vector<IDwords> &AnalysisResults){
    char        ch;
    ifstream    rfile(source.c_str());
    if(!rfile.is_open()){
        cout<<"无法打开源文件"<<endl;
        exit(-1);
    }

    rfile>>noskipws;    //不过滤空格
    while(rfile>>ch){
        int        state=0;        //判断状态
        string      temp("");        //字符串缓存
        char        try_ch;        //探测前面的字符

        switch(state){

```



```
case 0:
    if(ch=='/') //可能是注释
    {
        rfile>>try_ch;
        if(try_ch=='/'){
            while(rfile>>try_ch){
                if(try_ch=='\n')
                    break;    //这是一行注释
            }
            break;
        }
        else if(try_ch=='*'){
            while(rfile>>try_ch){
                if(try_ch=='*'){
                    rfile>>try_ch;
                    if(try_ch=='/')
                        break; //这是多行注释
                }
            }
            break;
        }
        }else{
            add_keywords(AnalysisResults,K_OPERATOR,char_to_str(ch));
            ch = try_ch; //继续状态1
        }
    }
case 1:
    if(is_operator(ch)) //判断操作符
    {
        add_keywords(AnalysisResults,K_OPERATOR,char_to_str(ch));
        break;
    }
case 2:
    if(is_bracket(ch)) //大括号、小括号
    {
        add_keywords(AnalysisResults,K_BRACKET,char_to_str(ch));
        break;
    }
case 3:
    if(is_blank(ch)) //空白符
        break;
case 4:
    if(ch=='#') //跳过预处理
    {
        while(rfile>>ch){
            if(is_blank(ch)){
                break;
            }
        }
        break;
    }
```

```

    }
    default://判断单词类型
        temp = temp + char_to_str(ch);
        while(rfile>>try_ch){
            if(try_ch == '\'){
                temp = temp + char_to_str(try_ch);
                if(ch == '\'){
                    add_keywords(AnalysisResults,word_token(temp),temp);
                    break;
                }
            }
            else{
                cout<<"不合法的标识符: "+temp<<endl;
                exit(-1);
            }
        }
        else if(is_blank(try_ch) ){
            if(ch != '\'' && ch != '\\"'){
                add_keywords(AnalysisResults,word_token(temp),temp);
                break;
            }
            else
                temp = temp + char_to_str(try_ch);
        }
        else if(is_operator(try_ch) ){
            if(ch != '\'' && ch != '\\" ){
                add_keywords(AnalysisResults,word_token(temp),temp);
                add_keywords(AnalysisResults,K_OPERATOR,char_to_str(try_ch));
                break;
            }
            else
                temp = temp + char_to_str(try_ch);
        }
        else if(is_bracket(try_ch)){
            add_keywords(AnalysisResults,word_token(temp),temp);
            add_keywords(AnalysisResults,K_BRACKET,char_to_str(try_ch));
            break;
        }
        else
            temp = temp + char_to_str(try_ch);
    }
}

}

rfile.close();
}

//输出词法分析结果
void print_lexical(vector<IDwords> &v){
    vector<IDwords>::iterator it;
    for(it = v.begin();it != v.end();it++)

```

```

        cout<<it->id<<" "<<it->word<<endl;
    }

//获取变量声明
void add_var_table(vector<IDwords>::iterator &it){
    while(it->id == K_TYPE){
        it++;
        while(it->word != ";"){

            if(it->id == K_IDENTIFIER){
                Variable    tmp;
                tmp.var = it->word;
                string    tmp_var = it->word;
                if((it+1)->word=="")    //判断变量有没有初始化
                {
                    it = it+2;
                    tmp.value = it->word;
                    add_target_code(tmp_var,"=",tmp.value," "," "," ");
                }
                var_table.push_back(tmp);
            }
            it++;
        }
        it++;
    }
}

```

```

//表达式分析
void expression(vector<IDwords>::iterator &it){
    string dsf,op,dst,dsc;
    //保存非操作符栈
    stack<string>    word_stack;
    //操作符栈
    stack<string>    oper_stack;
    oper_stack.push("#");
    while(it->word != ";")    //遇到','一条语句结束
    {

        if(it->word == "(")
            oper_stack.push(it->word);
        else if(it->word == ")"){

            while(oper_stack.top() != "("){
                op = oper_stack.top();

                oper_stack.pop();
            }
            //            oper_stack.push(it->word);
            dsc = word_stack.top();
            word_stack.pop();
        }
    }
}

```

```

        dst = word_stack.top();
        word_stack.pop();
        vab = vab+1;
        if(vab == 91)
            vab = '0';
        dsf = "tmp" + char_to_str(vab);

        Variable    tmp;
        tmp.var = dsf;
        var_table.push_back(tmp);

        word_stack.push(dsf);
        add_target_code(dsf,op,dst,dsc," "," ");
    }
    oper_stack.pop();

}

}

else if(it->id != K_OPERATOR)
    word_stack.push(it->word);
else if(oper_stack.top() == "("){
    oper_stack.push(it->word);
}

else if(level(it->word) < level(oper_stack.top())) //优先级低
{
    op = oper_stack.top();
    oper_stack.pop();
    oper_stack.push(it->word);
    dsc = word_stack.top();
    word_stack.pop();
    dst = word_stack.top();
    word_stack.pop();
    vab = vab+1;
    if(vab == 91)
        vab = '0';
    dsf = "tmp" + char_to_str(vab);

    Variable    tmp;
    tmp.var = dsf;
    var_table.push_back(tmp);

    word_stack.push(dsf);
    add_target_code(dsf,op,dst,dsc," "," ");
}

else //优先级高
    oper_stack.push(it->word);
it++;
}

//弹出剩下的
while(oper_stack.top() != "#"){
    op = oper_stack.top();

```

```

oper_stack.pop();
dsc = word_stack.top();
word_stack.pop();
dst = word_stack.top();
word_stack.pop();

if(op=="=")//赋值运算
{
    add_target_code(dst,op,dsc," "," "," ");
}
else{
    vab = vab+1;
    if(vab == 91)
        vab = '0';
    dsf = "tmp" + char_to_str(vab);

    Variable      tmp;
    tmp.var = dsf;
    var_table.push_back(tmp);

    word_stack.push(dsf);
    add_target_code(dsf,op,dst,dsc," "," ");
}
}
}

//分析printf输出
void printf_analysis(vector<IDwords>::iterator &it){
    int j,i=1;
    it = it+2;
    string str = it->word; //获取输出内容
    string strvar;         //获取输出变量

    Variable      tmp;
    //分析输出内容及格式
    for(j=1;j<str.size()-1;){
        if(str[j]=='%'){
            if(i != j){
                vab = vab + 1;
                if(vab == 91)
                    vab = '0';
                add_target_code("\'+str.substr(i,j-i)+"$\'","p"," "," ","tmp"+
                    char_to_str(vab)," ");
                tmp.var = "tmp"+char_to_str(vab);
                tmp.value = "\'+str.substr(i,j-i)+"$\'";
                var_table.push_back(tmp);
            }

            i = j+2;
            it = it+2; //获取对应变量
        }
    }
}

```

```

        strvar = it->word;
        add_target_code(strvar,"p"," "," ",str.substr(j,2)," ");
        j = i;
        continue;
    }
    j++;
}
if(i!=j){
    vab = vab+1;
    if(vab == 91)
        vab = '0';
    add_target_code("\'+str.substr(i,j-i)+"$\'","p"," "," ", "tmp"+
        char_to_str(vab)," ");
    tmp.var = "tmp"+char_to_str(vab);
    tmp.value = "\'+str.substr(i,j-i)+"$\'";
    var_table.push_back(tmp);
}
it = it+2; //略过 “)”
}

```

//分析if语句

```

void if_analysis(vector<IDwords>::iterator &it)
{
    string op,mark,dst,dsc;
    it++;
    if(it->word != "("){
        cout<<"错误的if语句: 缺少'(' "<<endl;
        exit(-1);
    }
    it++;
    dst = it->word;
    it++;
    mark = it->word;
    it++;
    dsc = it->word;
    op = "if";

    add_target_code(" ",op,dst,dsc,mark,"step"+char_to_str(lab+1));
    it++;
    if(it->word != ")"){
        cout<<"错误的if条件语句: 缺少')' "<<endl;
        exit(-1);
    }
    it++; //略过 '{'
    it++;

```

//分析else

```

vector<IDwords>::iterator it2 = it;
while(it2->word != "}") {

```

```

        it2++;
    }
    it2++;
    //判断有没有else
    if(it2->word == "else"){
        it2++; //略过 '{'
        while(it2->word != "}") {
            expression(it2);
            it2++;
        }
    } //else分析完成
    else
        it2--;
    lab = lab + 2;
    add_target_code(" ", "jmp", " ", " ", " ", "step"+char_to_str(lab));

    add_target_code(" ", "pstep", " ", " ", " ", "step"+char_to_str(lab-1));

    while(it->word != "}") {
        expression(it); //表达式分析
        it++;
    }
    add_target_code(" ", "jmp", " ", " ", " ", "step"+char_to_str(lab));
    add_target_code(" ", "pstep", " ", " ", " ", "step"+char_to_str(lab));

    it = it2;
}

//语法分析
void syntax_analysis(vector<IDwords> &AnalysisResults)
{
    vector<IDwords>::iterator it=AnalysisResults.begin();
    if(it->word != "main"){
        cout<<"缺少main"<<endl;
        exit(-1);
    }
    it = it+3; //跳过 "("
    if(it->word != "{"){
        cout<<"main函数缺少{''"<<endl;
        exit(-1);
    }
    it++;
    //获取变量声明
    add_var_table(it);
    //获取代码段的操作
    while(it != AnalysisResults.end()){
        //遇到printf
        if(it->word == "printf"){
            printf_analysis(it);

```

```

    }
    // if 语句
    else if(it->word == "if"){
        if_analysis(it);
    }
    else if(it->word == "}")
        break;
    //表达式分析
    else{
        expression(it); //表达式分析
    }
    it++;
}
}

//输出语法分析结果
void print_syntax(){
    vector<Variable>::iterator it;
    cout<<"变量声明及初始化"<<endl;
    for(it = var_table.begin();it != var_table.end();it++){
        cout<<it->var<<"    "<<it->value<<endl;
    }

    vector<Target>::iterator it2;
    cout<<"中间代码"<<endl;
    for(it2 = target_code.begin();it2 != target_code.end();it2++){
        cout<<it2->dsf<<"    "<<it2->op<<"    "<<it2->dst<<"    "<<it2->dsc<<"    "<<
            it2->mark<<"    "<<it2->step<<endl;
    }
}

//加减法转换
void addsub_asm(ofstream &out,string dsf,string op,string dst,string dsc){
    out<<"    mov BL,"<<dst<<endl;
    if(op == "+")
        out<<"    add BL,"<<dsc<<endl;
    else
        out<<"    sub BL,"<<dsc<<endl;
    out<<"    mov "<<dsf<<","BL"<<endl;
}

//乘法
void mul_asm(ofstream &out,string dsf,string dst,string dsc){
    out<<"    mov AL,"<<dst<<endl;
    out<<"    mov BH,"<<dsc<<endl;
    out<<"    mul BH"<<endl;
    out<<"    mov BL,1"<<endl;
    out<<"    div BL"<<endl;
    out<<"    mov "<<dsf<<","AL"<<endl;
}

```



```
}

```

```
//除法

```

```
void div_asm(ofstream &out,string dsf,string dst,string dsc){
    out<<"    mov AL,"<<dst<<endl;
    out<<"    CBW"<<endl;
    out<<"    mov BL,"<<dsc<<endl;
    out<<"    div BL"<<endl;
    out<<"    mov "<<dsf<<","AL"<<endl;
}

```

```
//赋值运算

```

```
void sign_asm(ofstream &out,string dsf,string dst){
    out<<"    mov BL,"<<dst<<endl;
    out<<"    mov "<<dsf<<","BL"<<endl;
}

```

```
//输出转换

```

```
void print_asm(ofstream &out,string dsf,string mark){
    //以字符格式输出
    if(mark=="%c"){
        out<<"    mov DL,"<<dsf<<endl;
        out<<"    mov AH,02H"<<endl;
        out<<"    int 21H"<<endl;
    }
    //以整数格式输出
    else if(mark=="%d"){
        out<<"    mov AL,"<<dsf<<endl;
        out<<"    CBW"<<endl;
        out<<"    mov BL,10"<<endl;
        out<<"    DIV BL"<<endl;
        out<<"    mov BH,AH"<<endl;
        out<<"    add BH,30H"<<endl;
        out<<"    add AL,30H"<<endl;
        out<<"    CMP AL,48"<<endl;
        //确定十位是否是0
        lab = lab + 2;
        string step2 = "step" + char_to_str(lab);
        out<<"    JE "<<step2<<endl;
        string step1 = "step" + char_to_str(lab-1);
        out<<"    "<<step1<<":"<<endl;
        out<<"    mov DL,AL"<<endl;
        out<<"    mov AH,2"<<endl;
        out<<"    int 21H"<<endl;

        //输出个位
        out<<"    "<<step2<<":"<<endl;
        out<<"    mov DL,BH"<<endl;
        out<<"    mov AH,2"<<endl;
    }
}

```

```

        out<<"    int 21H"<<endl;
    }
    //字符串输出
    else{
        out<<"    LEA DX,"<<mark<<endl;
        out<<"    mov AH,09"<<endl;
        out<<"    int 21H"<<endl;
    }
}

//if语句转换
void if_asm(ofstream &out,string dst,string dsc,string mark,string step){
    out<<"    mov AL,"<<dst<<endl;
    out<<"    CMP AL,"<<dsc<<endl;
    if(mark == ">")
        out<<"    JG "<<step<<endl;
    else if(mark == "<")
        out<<"    JL "<<step<<endl;
    else{
        cout<<"暂不支持其他条件判断"<<endl;
        exit(-1);
    }
}

//生成汇编文件
void create_asm(string file){
    //变量声明
    ofstream wfile(file.c_str());
    if(!wfile.is_open())
        cout<<"无法创建汇编文件"<<endl;

    vector<Variable>::iterator it_var;

    wfile<<"ASSUME CS:codesg,DS:datasg"<<endl;
    //数据段
    wfile<<"datasg segment"<<endl;
    for(it_var=var_table.begin();it_var!=var_table.end();it_var++){
        wfile<<"    "<<it_var->var<<" DB ";
        if(it_var->value != "")
            wfile<<it_var->value<<endl;
        else
            wfile<<"\'?\'"<<endl;
    }
    wfile<<"datasg ends"<<endl;
    //代码段
    wfile<<"codesg segment"<<endl;
    wfile<<"    start:"<<endl;
    wfile<<"    mov AX,datasg"<<endl;
    wfile<<"    mov DS,AX"<<endl;
}

```

```

vector<Target>::iterator it;
Target tmp;
for(it = target_code.begin();it != target_code.end();it++){
    //加减法转化
    if(it->op == "+" || it->op=="-")
        addsub_asm(wfile,it->dsf,it->op,it->dst,it->dsc);
    //乘法转换
    else if(it->op == "*")
        mul_asm(wfile,it->dsf,it->dst,it->dsc);
    //除法转换
    else if(it->op == "/")
        div_asm(wfile,it->dsf,it->dst,it->dsc);
    //赋值运算
    else if(it->op == "=")
        sign_asm(wfile,it->dsf,it->dst);
    //输出操作
    else if(it->op == "p")
        print_asm(wfile,it->dsf,it->mark);
    //if语法分析
    else if(it->op == "if"){
        if_asm(wfile,it->dst,it->dsc,it->mark,it->step);
    }
    else if(it->op == "else"){
        cout<<"else 没有找到匹配的 if"<<endl;
        exit(-1);
    }
    //跳转语句
    else if(it->op == "jmp"){
        wfile<<"    JMP " <<it->step<<endl;
    }
    //跳转语句段标识
    else if(it->op == "pstep"){
        wfile<<"    "<<it->step<<":"<<endl;
    }
    //其他
    else{
        cout<<"编译器暂不支持该语法操作"<<endl;
        exit(-1);
    }
}

//代码段结束
wfile<<"    mov ax,4C00H"<<endl;
wfile<<"    int 21H"<<endl;
wfile<<"codesg ends"<<endl;
wfile<<" end start"<<endl;

wfile.close();

```

```
}

int main(int argc, char* argv[]){
    vector<IDwords> AnalysisResults;

    string source;

    //缺省时给以提示
    if(argc == 1){
        cout<<"*****"<<endl;
        cout<<"\n在源文件目录下生成 .asm 汇编文件\n"<<endl;
        cout<<"*****"<<endl;
        cout<<"\n请输入源文件: ";
        cin>>source;
        //词法分析
        lexical_analysis(source, AnalysisResults);
        //语法分析
        syntax_analysis(AnalysisResults);
        //生成汇编文件
        create_asm(asmfile(source));
        cout<<"\n使用结束"<<endl;
    }else if(argc == 2){
        //默认生成汇编文件: 源文件名.asm
        //词法分析
        lexical_analysis(argv[1], AnalysisResults);
        //语法分析
        syntax_analysis(AnalysisResults);
        //生成汇编文件
        create_asm(asmfile(argv[1]));
    }else if(argc == 3){
        //词法分析
        lexical_analysis(argv[1], AnalysisResults);
        //语法分析
        syntax_analysis(AnalysisResults);
        //生成汇编文件
        create_asm(asmfile(argv[2]));
    }else{
        cout<<"正确使用格式: compile.exe [源文件] [汇编文件]"<<endl;
    }

    //输出分析
    print_lexical(AnalysisResults);
    //输出语法分析结果
    print_syntax();
    return 0;
}
```

4.2 实验步骤

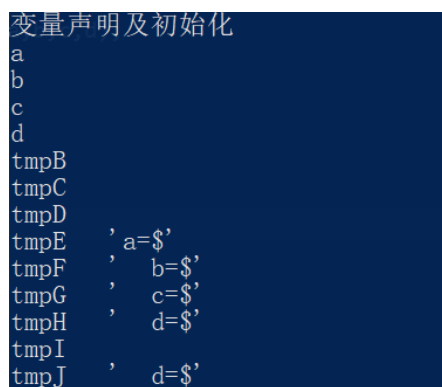
设计一个简单的例子

```
main(){
    int a,b,c;
    int d;

    a = 0;
    b = a+1;
    c = 10-b;
    d = b*c;
    printf("a=%d b=%d c=%d d=%d",a,b,c,d);

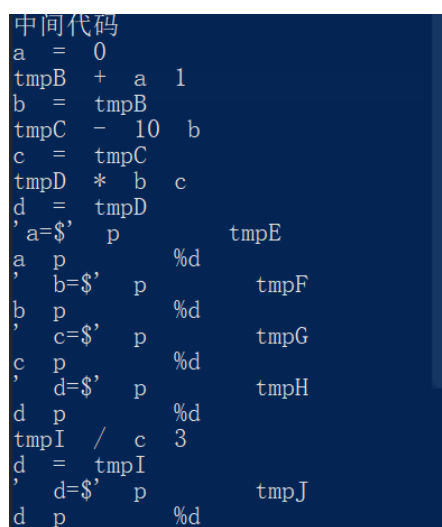
    d = c/3;
    printf(" d=%d",d);
}
```

执行程序进行实验，输出结果如图4.1-4.2所示，并且在当前目录下生成了一个asm文件。



```
变量声明及初始化
a
b
c
d
tmpB
tmpC
tmpD
tmpE    , a=$'
tmpF    , b=$'
tmpG    , c=$'
tmpH    , d=$'
tmpI
tmpJ    , d=$'
```

图 4.1: 变量声明及初始化



```
中间代码
a = 0
tmpB + a 1
b = tmpB
tmpC - 10 b
c = tmpC
tmpD * b c
d = tmpD
'a=$' p tmpE
a p %d
'b=$' p tmpF
b p %d
'c=$' p tmpG
c p %d
'd=$' p tmpH
d p %d
tmpI / c 3
d = tmpI
'd=$' p tmpJ
d p %d
```

图 4.2: 中间代码

打开emu8086，加载此asm文件，如图4.3所示。

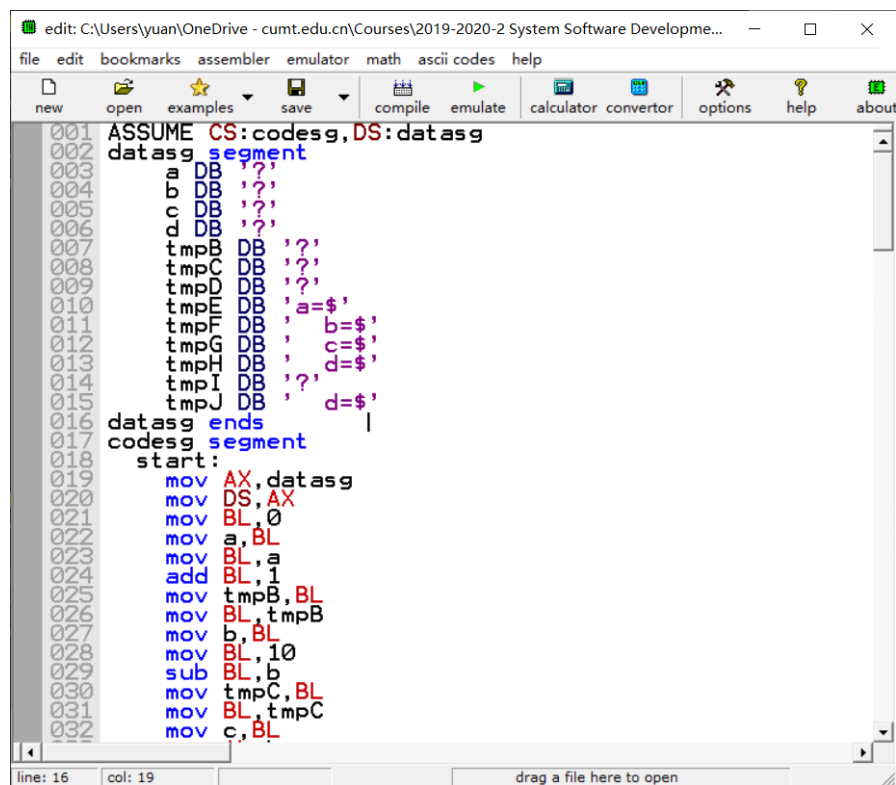


图 4.3: emu8086加载asm

进行编译，得到如下文件，内容是关于符号表地址、程序执行的地址访问次序和可执行文件exe。

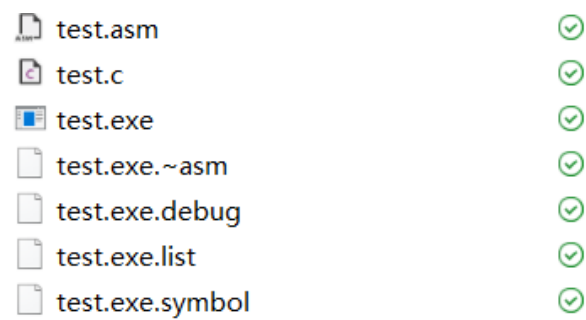


图 4.4: 编译结果

下图展示了所有符号所在的偏移地址，这对于代码的分析很有帮助。

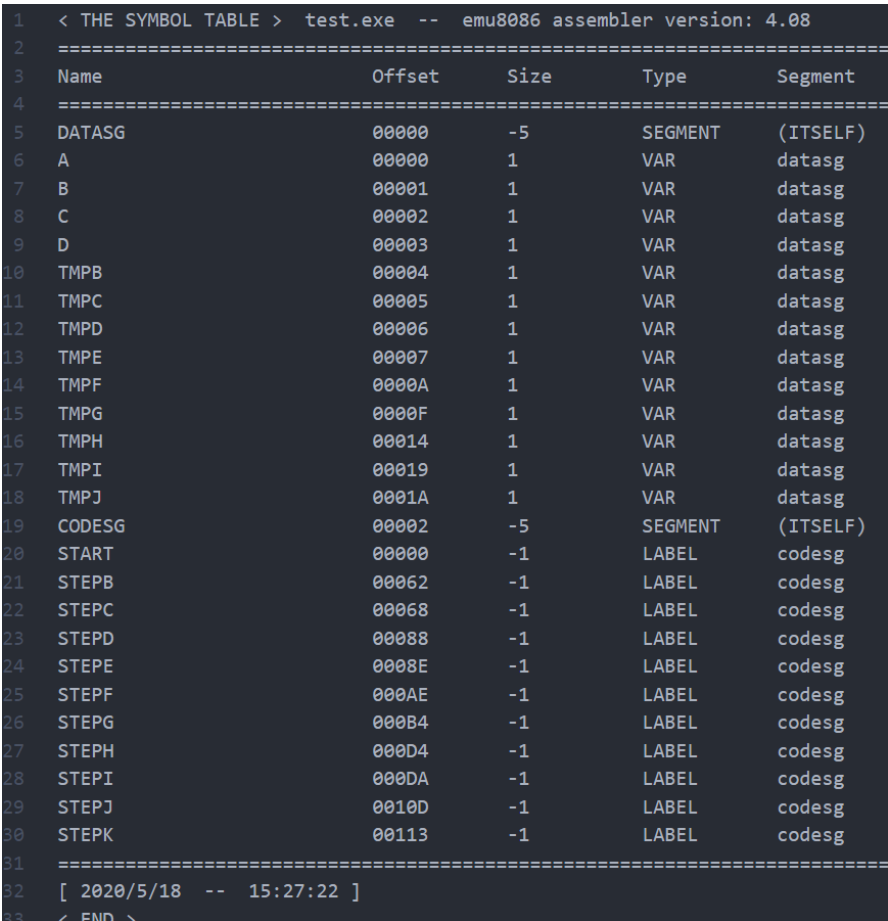


图 4.5: 符号偏移地址

点击emulate进行调试，会发现新出现了三个窗口，如图4.6所示。
original source code展示汇编源代码，emulator: test.exe展示寄存器地址的变化，emulator screen展示程序的输出结果。

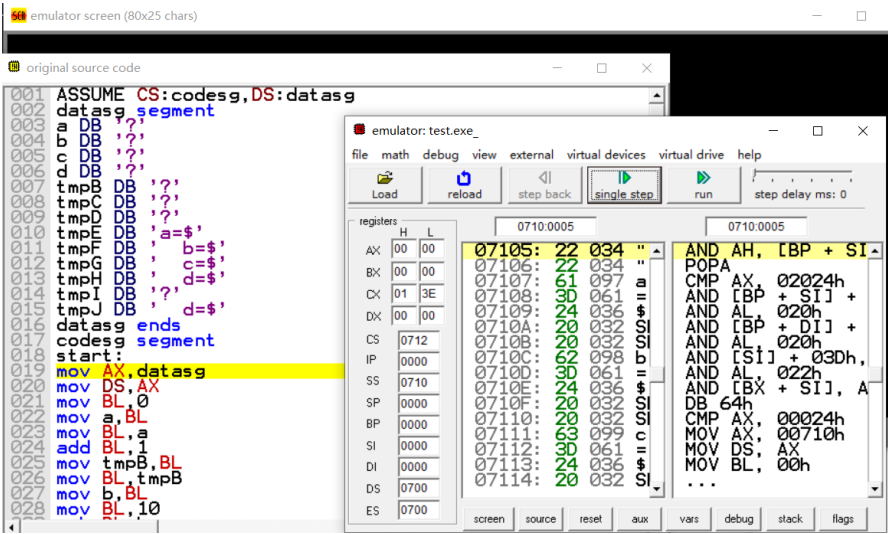


图 4.6: emulate新窗口

单步调试，下图展示了a=0的过程，首先将值00710h赋值给AX，然后将AX的值赋值给DS，然后给BL（即变量a）初始值00h。

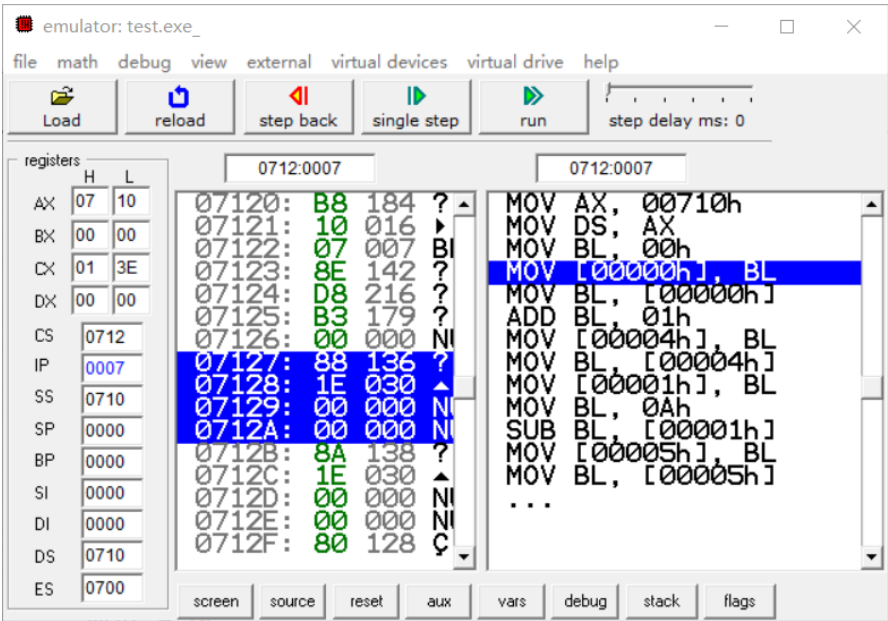


图 4.7: a=0

下图展示了 b=a+1 的过程，首先将地址 00000h（即a的值）赋值给 BL，然后对 BL 的值进行加 01h操作，最后将 BL 的值赋值给地址 00004h（即b的地址）所指向的值。

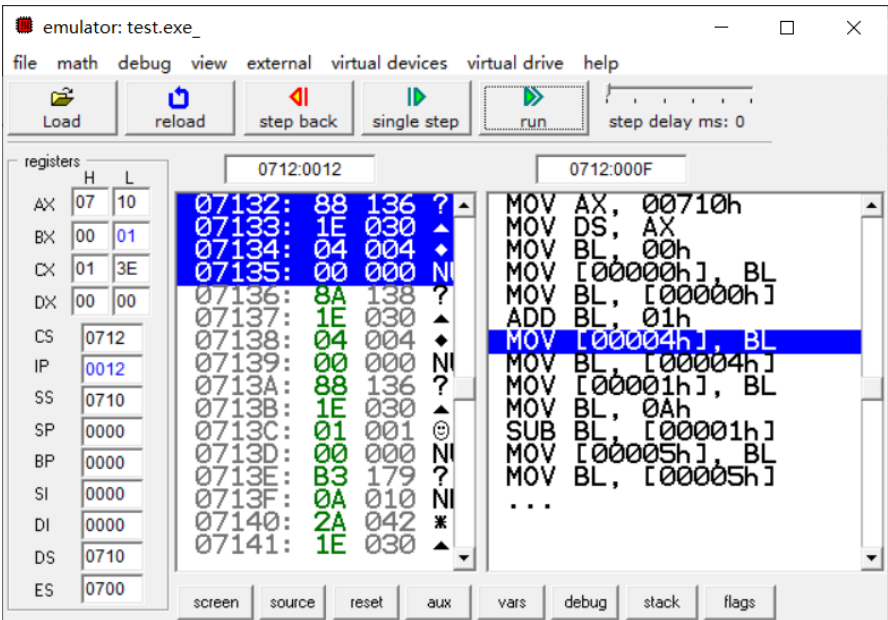


图 4.8: b=a+1

c = 10-b 与 d = b*c 类似，继续调试，最后输出 a=0 b=1 c=9 d=9 d=3，如图4.9所示，与预期结果一致。

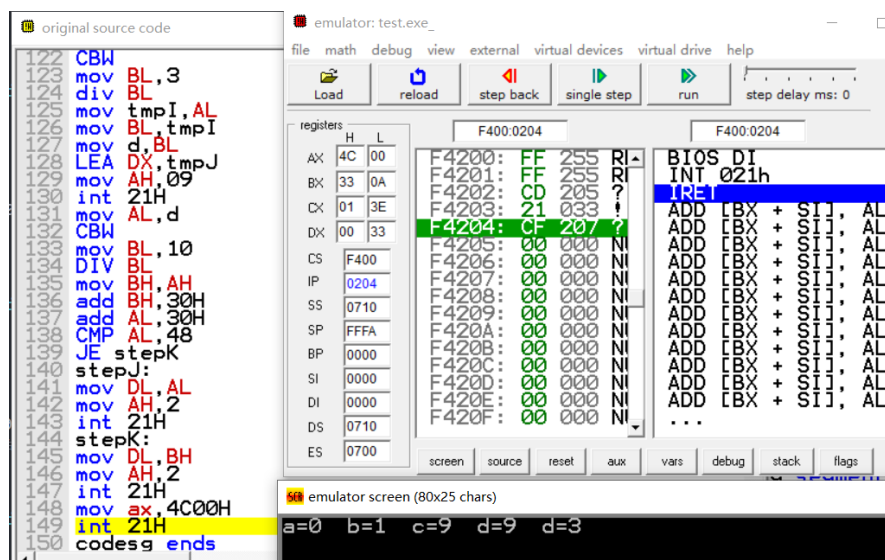


图 4.9: 输出结果

5 计算器后端实验

5.1 修改代码

将原来的实验代码序号 3 均改为 4，如 fb3-1.1 改为 fb4-1.1。

修改 fb4-1.1 文件，增加如下符号，删除不需要的内置函数，只保留最基本的 print() 函数。

```
"int" {return INTT;}
"main" {return MAIN;}
```

在最下方增加如下代码。

```
void print_val(){
    FILE *fwriteM = fopen("ans.asm","a+");
    fprintf(fwriteM, ";example for display register content\n");
    fprintf(fwriteM, "print    proc\npush    ax\npush    dx\npush    cx\nmov    dx,
-1\npush    dx\nmov    cx, 10\nl_div10:\nxor    dx,dx\ndiv    cx\npush    dx\ntest    ax,ax\n
jne    l_div10\nmov    cx,-1\nmov    ah,2\nl_disp:\npop    dx\ncmp    dx,cx\nje    l_ret\nadd    dl,
'0'\nint    21h\njmp    l_disp\nl_ret:\npop    cx\npop    dx\npop    ax\nret\nprint    endp\n");
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

int main(){
    FILE *fwriteCs = fopen("temp.txt","wt");
    fclose(fwriteCs);
    fwriteCs = fopen("ans.asm","wt");
    fclose(fwriteCs);
    fwriteCs = fopen("out.txt", "wt");
```

```

fclose(fwriteCs);

FILE *fwrites = fopen("temp.txt","a+");
fprintf(fwrites, "start");

fclose(fwrites);

FILE *fwriteM = fopen("ans.asm","a+");
fprintf(fwriteM, "data segment\n");
fprintf(fwriteM, "T DW 1000 DUP(0)\n");
fprintf(fwriteM, "H DW 1000 DUP(0)\n");
fprintf(fwriteM, "data ends\n\n");
fprintf(fwriteM, "stacks segment stack\ndb 1000 dup (128)\nstacks ends\ncode
segment\nassume cs:code,ds:data,ss:stacks,es:data\n\n");
fprintf(fwriteM, "start:\nMOV AX,CODE\nMOV DS,AX\nMOV DX,00H\n");
fprintf(fwriteM, "\n");
fclose(fwriteM);

yyin = fopen("in.txt", "r");
if(!(yyin)){
    perror("in.txt");
    return 0;
}
yyparse();

FILE *fwritesE = fopen("temp.txt","a+");
fprintf(fwritesE, "end");
fclose(fwritesE);
FILE *fwritesE1 = fopen("ans.asm","a+");
fprintf(fwritesE1, "MOV AH,4CH\nINT 21H\n\n");
fclose(fwritesE1);
print_val();
FILE *fwriteE = fopen("ans.asm","a+");
fprintf(fwriteE, "code ends\n");
fprintf(fwriteE, "end start");
fclose(fwriteE);

fclose(yyin);
return 0;
}

```

修改 fb4-1.y 文件，修改 stmt 规则，如下。

```

stmt: /* nothing */ { $$ = NULL; }
| list '}' { $$ = $1; }
;

```

修改 calclist 规则，如下。

```

calclist: /* nothing */{ }

```

```

| calclist INTT MAIN '(' ')' '{' stmt {
    initt();
    if(debug) dumpast($7, 0);
    else {
        eval($7);
        treefree($7);
        FILE *fwritez = fopen("temp.txt","a+");
        fprintf(fwritez, "\n");
        fclose(fwritez);
    }
}
| calclist error { yyerrok; fprintf(yyout, " "); }
;

```

修改 fb4-1.h 文件，更改 symbol 的结构，再增加 adress 与 four 结构体。

```

struct symbol {          /* a variable name */
    char *name;
    double value;
    int pos;
    struct ast *func;     /* stmt for the function */
};

struct adress {
    char type;
    int pos;
    double v;
};

struct four{
    char t1;
    int p1;
    char t2;
    int p2;
    char t3;
    int p3;
};

```

修改 fb4-1funcs.c 文件，增加汇编代码生成。

```

void MOV(struct four *p){
    if (p->t1 == 'N'){
        FILE *fwriteM = fopen("ans.asm", "a+");
        p->p3 = p->p3 * 2 + 1;
        fprintf(fwriteM, "MOV BX,%d\n", p->p3);
        fprintf(fwriteM, "MOV %c[BX],%d\n", p->t3, p->p1);
        fprintf(fwriteM, "\n");
        fclose(fwriteM);
    }
    else{

```

```

        FILE *fwriteM = fopen("ans.asm", "a+");
        p->p1 = p->p1 * 2 + 1;
        p->p3 = p->p3 * 2 + 1;
        fprintf(fwriteM, "MOV BX,%d\n", p->p1);
        fprintf(fwriteM, "MOV AX,%c[BX]\n", p->t1);
        fprintf(fwriteM, "MOV BX,%d\n", p->p3);
        fprintf(fwriteM, "MOV %c[BX],AX\n", p->t3);
        fprintf(fwriteM, "\n");
        fclose(fwriteM);
    }
}

void ADD(struct four *p){
    p->p1 = 2 * p->p1 + 1;
    p->p2 = 2 * p->p2 + 1;
    p->p3 = 2 * p->p3 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p1);
    fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t1);
    fprintf(fwriteM, "MOV BX,%d\n", p->p2);
    fprintf(fwriteM, "MOV DX,%C[BX]\n", p->t2);
    fprintf(fwriteM, "MOV BX,DX\n");
    fprintf(fwriteM, "ADD AX,BX\n");
    fprintf(fwriteM, "MOV BX,%d\n", p->p3);
    fprintf(fwriteM, "MOV %c[BX],AX\n", p->t3);
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

void SUB(struct four *p){
    p->p1 = 2 * p->p1 + 1;
    p->p2 = 2 * p->p2 + 1;
    p->p3 = 2 * p->p3 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p1);
    fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t1);
    fprintf(fwriteM, "MOV BX,%d\n", p->p2);
    fprintf(fwriteM, "MOV DX,%C[BX]\n", p->t2);
    fprintf(fwriteM, "MOV BX,DX\n");
    fprintf(fwriteM, "SUB AX,BX\n");
    fprintf(fwriteM, "MOV BX,%d\n", p->p3);
    fprintf(fwriteM, "MOV %c[BX],AX\n", p->t3);
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

void MUL(struct four *p){
    p->p1 = 2 * p->p1 + 1;
    p->p2 = 2 * p->p2 + 1;
    p->p3 = 2 * p->p3 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p1);

```

```

        fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t1);
        fprintf(fwriteM, "MOV BX,%d\n", p->p2);
        fprintf(fwriteM, "MOV DX,%C[BX]\n", p->t2);
        fprintf(fwriteM, "MOV BX,DX\n");
        fprintf(fwriteM, "MUL BX\n");
        fprintf(fwriteM, "MOV BX,%d\n", p->p3);
        fprintf(fwriteM, "MOV %c[BX],AX\n", p->t3);
        fprintf(fwriteM, "\n");
        fclose(fwriteM);
    }

```

```

void DIV(struct four *p){
    p->p1 = 2 * p->p1 + 1;
    p->p2 = 2 * p->p2 + 1;
    p->p3 = 2 * p->p3 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p1);
    fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t1);
    fprintf(fwriteM, "MOV DX,0\n");
    fprintf(fwriteM, "MOV BX,%d\n", p->p2);
    fprintf(fwriteM, "MOV CX,%C[BX]\n", p->t2);
    fprintf(fwriteM, "MOV BX,CX\n");
    fprintf(fwriteM, "DIV BX\n");
    fprintf(fwriteM, "MOV BX,%d\n", p->p3);
    fprintf(fwriteM, "MOV %c[BX],AX\n", p->t3);
    //fprintf(fwriteM, "MOV,%c[BX],DX\n",p->t3-1);
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

```

```

void NEG(struct four *p){
    p->p3 = 2 * p->p3 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p3);
    fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t3);
    fprintf(fwriteM, "NEG AX\n");
    fprintf(fwriteM, "MOV %C[BX],AX\n", p->t3);
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

```

```

void PRT(struct four *p){
    p->p1 = 2 * p->p1 + 1;
    FILE *fwriteM = fopen("ans.asm", "a+");
    fprintf(fwriteM, "MOV BX,%d\n", p->p1);
    fprintf(fwriteM, "MOV AX,%C[BX]\n", p->t1);
    fprintf(fwriteM, "call print\n");
    fprintf(fwriteM, "\n");
    fclose(fwriteM);
}

```

```
}

```

更改计算过程，调用函数输出汇编代码与四元式。

```
struct address *eval(struct ast *a){
    double v;
    struct address *re = malloc(sizeof(struct address));
    struct address *l = malloc(sizeof(struct address));
    struct address *r = malloc(sizeof(struct address));
    struct four *f = malloc(sizeof(struct four));
    f->t1 = 'N';
    f->t2 = 'N';
    f->t3 = 'N';
    if (!a){
        yyerror("internal error, null eval");
        re->v = 0.0;
        re->type = 'N'; //-1为空
        return re;
    }

    switch (a->nodetype){
        /* constant常量 */
        case 'K':
            v = ((struct numval *)a)->number;
            re->v = v;
            re->type = 'T';
            re->pos = ti++;
            int pv = v;
            FILE *fwrite = fopen("temp.txt", "a+");
            fprintf(fwrite, "MOV,%d,,%c%d\n", pv, re->type, re->pos);
            fclose(fwrite);
            f->t3 = 'T';
            f->p1 = pv;
            f->p3 = re->pos;
            MOV(f);
            break;

            /* name reference名字引用 */
        case 'N':
            v = ((struct symref *)a)->s->value;
            re->v = v;
            re->type = 'H';
            if (((struct symref *)a)->s->pos == -1){
                FILE *fwrite1 = fopen("temp.txt", "a+");
                pv = v;
                ((struct symref *)a)->s->pos = re->pos = hi++;
                fprintf(fwrite1, "MOV,%d,,%c%d\n", pv, re->type, re->pos);
                fclose(fwrite1);
                f->t3 = 'H';
                f->p1 = pv;
            }
    }
}
```

```

        f->p3 = re->pos;
        MOV(f);
    }
    else
        re->pos = ((struct symref *)a)->s->pos;
    break;

    /* assignment赋值 */
case '=':
    re = eval(((struct symasn *)a)->v);
    ti = 0;
    v = re->v;
    int Hpos, Tpos, Ttyp;
    if (((struct symasn *)a)->s->pos == -1)
        Hpos = ((struct symasn *)a)->s->pos = hi++;
    else
        Hpos = ((struct symasn *)a)->s->pos;
    Tpos = re->pos;
    Ttyp = re->type;
    FILE *fwrite2 = fopen("temp.txt", "a+");
    fprintf(fwrite2, "MOV,%c%d,,H%d\n", Ttyp, Tpos, Hpos);
    fclose(fwrite2);
    f->t1 = Ttyp;
    f->p1 = Tpos;
    f->t3 = 'H';
    f->p3 = Hpos;
    MOV(f);
    ((struct symasn *)a)->s->value = re->v;
    break;

    /* expressions 表达式*/
case '+':
    l = eval(a->l);
    r = eval(a->r);
    v = l->v + r->v;
    int lt, rt, ls, rs;
    lt = l->type;
    rt = r->type;
    ls = l->pos;
    rs = r->pos;
    re->pos = ti++;
    re->type = 'T';
    re->v = v;
    FILE *fwrite3 = fopen("temp.txt", "a+");
    fprintf(fwrite3, "ADD,%c%d,%c%d,T%d\n", lt, ls, rt, rs, re->pos);
    fclose(fwrite3);
    f->t1 = lt;
    f->p1 = ls;
    f->t2 = rt;

```

```

        f->p2 = rs;
        f->t3 = 'T';
        f->p3 = re->pos;
        ADD(f);
        break;
case '-':
    l = eval(a->l);
    r = eval(a->r);
    v = l->v - r->v;
    /*int lt,rt,ls,rs;*/
    lt = l->type;
    rt = r->type;
    ls = l->pos;
    rs = r->pos;
    re->pos = ti++;
    re->type = 'T';
    re->v = v;
    FILE *fwrite4 = fopen("temp.txt", "a+");
    fprintf(fwrite4, "SUB,%c%d,%c%d,T%d\n", lt, ls, rt, rs, re->pos);
    fclose(fwrite4);
    f->t1 = lt;
    f->p1 = ls;
    f->t2 = rt;
    f->p2 = rs;
    f->t3 = 'T';
    f->p3 = re->pos;
    SUB(f);
    break;
case '*':
    l = eval(a->l);
    r = eval(a->r);
    v = l->v * r->v;
    /*int lt,rt,ls,rs;*/
    lt = l->type;
    rt = r->type;
    ls = l->pos;
    rs = r->pos;
    re->pos = ti++;
    re->type = 'T';
    re->v = v;
    FILE *fwrite5 = fopen("temp.txt", "a+");
    fprintf(fwrite5, "MUL,%c%d,%c%d,T%d\n", lt, ls, rt, rs, re->pos);
    fclose(fwrite5);
    f->t1 = lt;
    f->p1 = ls;
    f->t2 = rt;
    f->p2 = rs;
    f->t3 = 'T';
    f->p3 = re->pos;

```



```

        MUL(f);
        break;
    case '/':
        l = eval(a->l);
        r = eval(a->r);
        v = l->v / r->v;
        /*int lt,rt,ls,rs;*/
        lt = l->type;
        rt = r->type;
        ls = l->pos;
        rs = r->pos;
        re->pos = ti++;
        re->type = 'T';
        re->v = v;
        FILE *fwrite6 = fopen("temp.txt", "a+");
        fprintf(fwrite6, "DIV,%c%d,%c%d,T%d\n", lt, ls, rt, rs, re->pos);
        fclose(fwrite6);
        f->t1 = lt;
        f->p1 = ls;
        f->t2 = rt;
        f->p2 = rs;
        f->t3 = 'T';
        f->p3 = re->pos;
        DIV(f);
        break;
    case 'M':
        l = eval(a->l);
        v = -l->v;
        /*int lt,ls;*/
        lt = l->type;
        ls = l->pos;
        re->pos = ls;
        re->type = lt;
        re->v = v;
        FILE *fwrite7 = fopen("temp.txt", "a+");
        fprintf(fwrite7, "NEG,,,%c%d\n", lt, ls);
        fclose(fwrite7);
        f->t3 = lt;
        f->p3 = ls;
        NEG(f);
        break;
    case 'L':
        eval(a->l);
        re = eval(a->r);
        v = re->v;
        break;

    case 'F':
        re = callbuiltin((struct fncall *)a);

```

```
        v = re->v;
        break;

default:
    printf("internal error: bad node %c\n", a->nodetype);
}
return re;
}
```

5.2 中间代码汇编代码生成

打开终端，进入实验目录，依次输入如下命令，进行代码的编译。

```
bison -d fb4-1.y
flex fb4-1.l
gcc lex.yy.c fb4-1.tab.c fb4-1funcs.c -lm
```

然后修改 in.txt，内容如下，实验结果应该输出6和3。

```
int main () {
    a = 2 * (2 + 1);
    b = a / 2;
    print(a);
    print(b);
}
```

输入 ./a.out，结果如图5.1-5.2所示。

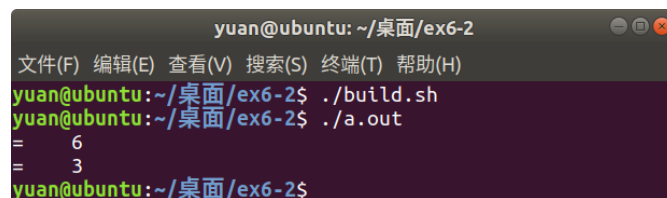


图 5.1: 终端结果

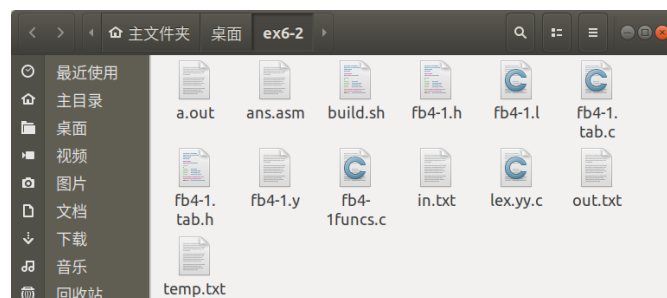


图 5.2: 生成文件

中间代码内容如下所示。

```

start
MOV,2,,T0
MOV,2,,T1
MOV,1,,T2
ADD,T1,T2,T3
MUL,T0,T3,T4
MOV,T4,,H0
MOV,2,,T0
DIV,H0,T0,T1
MOV,T1,,H1
print,H0, ,
print,H1, ,

end

```

生成的汇编代码如下。

data segment	MOV BX,7	MOV BX,3
T DW 1000 DUP(0)	MOV T[BX],AX	MOV H[BX],AX
H DW 1000 DUP(0)		
data ends	MOV BX,1	MOV BX,1
	MOV AX,T[BX]	MOV AX,H[BX]
stacks segment stack	MOV BX,7	call print
db 1000 dup (128)	MOV DX,T[BX]	
stacks ends	MOV BX,DX	MOV BX,3
code segment	MUL BX	MOV AX,H[BX]
assume cs:code,ds:data,	MOV BX,9	call print
ss:stacks,es:data	MOV T[BX],AX	
		MOV AH,4CH
start:	MOV BX,9	INT 21H
MOV AX,CODE	MOV AX,T[BX]	
MOV DS,AX	MOV BX,1	print proc
MOV DX,00H	MOV H[BX],AX	push ax
		push dx
MOV BX,1	MOV BX,1	push cx
MOV T[BX],2	MOV T[BX],2	mov dx,-1
		push dx
MOV BX,3	MOV BX,1	mov cx, 10
MOV T[BX],2	MOV AX,H[BX]	l_div10:
	MOV DX,0	xor dx,dx
MOV BX,5	MOV BX,1	div cx
MOV T[BX],1	MOV CX,T[BX]	push dx
	MOV BX,CX	test ax,ax
MOV BX,3	DIV BX	jne l_div10
MOV AX,T[BX]	MOV BX,3	mov cx,-1
MOV BX,5	MOV T[BX],AX	mov ah,2
MOV DX,T[BX]		l_disp:
MOV BX,DX	MOV BX,3	pop dx
ADD AX,BX	MOV AX,T[BX]	cmp dx,cx

```
je l_ret      pop cx
add dl,'0'    pop dx      code ends
int 21h       pop ax      end start
jmp l_disp   ret
l_ret:        print      endp
```

5.3 emu8086验证代码

用 emu8086 打开 asm 汇编文件，点击 compile 进行编译，结果如图5.3-5.4所示。

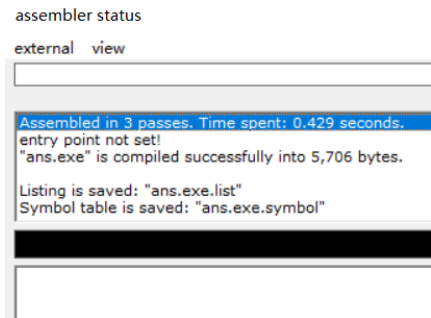


图 5.3: 编译结果1

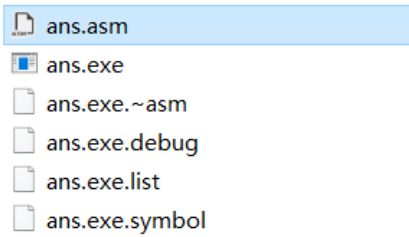


图 5.4: 编译结果2

符号表内容如下所示。

```
< THE SYMBOL TABLE >  ans.exe  --  emu8086 assembler version: 4.08
=====
Name                Offset      Size      Type      Segment
=====
DATA                00000      -5      SEGMENT   (ITSELF)
T                   00000        2      VAR       data
H                   007D0        2      VAR       data
STACKS              000FA      -5      SEGMENT   (ITSELF)
CODE                 00139      -5      SEGMENT   (ITSELF)
START               00000       -1      LABEL     code
PRINT               00092       -1      NEAR      code
L_DIV10             0009C       -1      LABEL     code
L_DISP              000AA       -1      LABEL     code
L_RET               000B6       -1      LABEL     code
=====
[ 2020/5/21  --  10:29:53 ]
< END >
```

进行运行调试，最后结果为6与3，与预期结果一致，如图5.5所示。

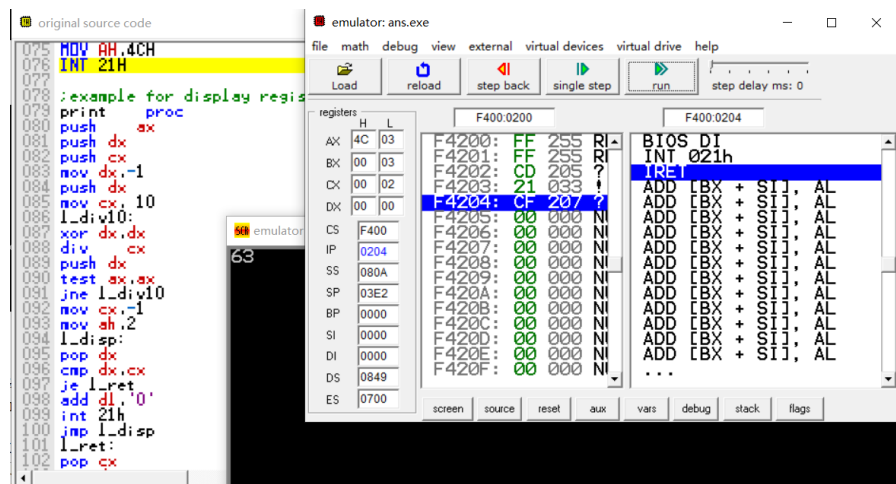


图 5.5: 计算器运行结果

6 错误处理

新建一个 test2.c 文件，修改内容为如下代码，会发现检测出不合法的标识符 1c，如图6.1所示。

```
main(){
    int 1c;
    int a,b;
    return 0;
}
```

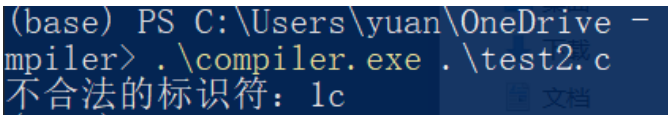


图 6.1: 不合法的标识符

继续测试其他错误，修改内容为如下代码，会发现检测出错误的字符串，如图6.2所示。

```
main(){
    char s = "sssss';
    int a,b;
    return 0;
}
```

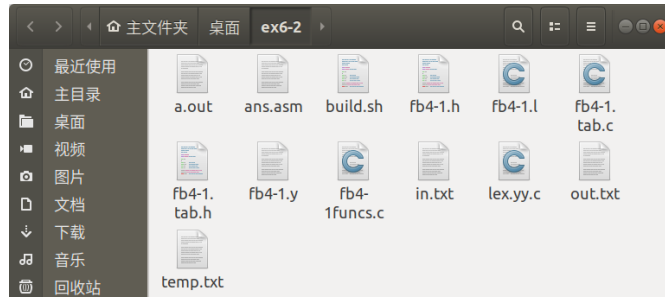


图 6.2: 错误的字符串

继续测试其他错误，修改内容为如下代码，会发现检测出错误的if语句，如图6.3所示。

```
main(){  
    int a,b;  
    if a;  
    else b;  
    return 0;  
}
```

```
mpiler> .\compiler.exe .\test2.c  
错误的if语句: 缺少' ('  
(base) PS C:\Users\yuan\OneDrive
```

图 6.3: 错误的if语句

7 图形GUI

将之前实验集成 GUI 进行可视化实验，如图7.1所示。通过打开实验文件，代码区会显示文件内容，可以进行适当的修改，然后进行保存，可以看得终端文件操作的日志。

保存成功后，可以进行编译，若编译成功文件夹目录内会多出 a.out 文件，然后点击命令里面的运行，就可以输出汇编代码、中间代码，也会有输出结果的日志。

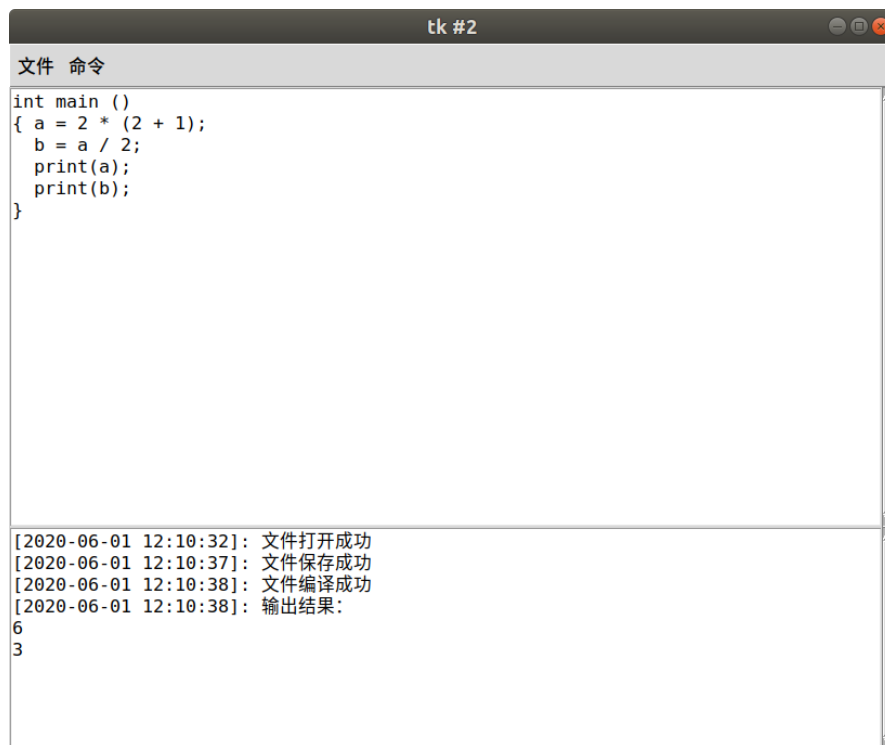


图 7.1: GUI编译器