中国矿业大学计算机学院

2016 级本科生课程设计报告

课程名称 系统软件开发实践

报告时间 2019.4.15

学生姓名 黄雪纯

学　　号 08163215

专　　业 计算机科学与技术

任课教师 张博

成绩考核

编号	课程教学目标	占比	得分
1	**目标 1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用 FLEX 脚本语言描述单词结构。	15%	
2	**目标 2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用 Bison 脚本语言描述语法结构。	15%	
3	**目标 3：**针对计算器需求描述，采用 Flex/Bison 设计实现高级解释器，进行系统设计，形成结构化设计方案。	30%	
4	**目标 4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。	30%	
5	**目标 5：**培养独立解决问题的能力，理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。	10%	
总成绩			
指导教师		评阅日期	

目录

C 编译器后端设计

一 实验目的

在学习和熟悉了编译器前端具体过程并进行相关的实验练习后，继续深入学习编译器后端的设计和实现；通过对后端的了解并且结合前面所学知识能够对编译器有着具体的轮廓和系统的认识；在将来的具体项目中能够依据自己的见解和认识做出来具体成果。

二 实验内容

将前四周得到的编译器前端结果转换为四元式，最后生成 X86 汇编。对得到的汇编代码进行如下处理：

（1）将得到的汇编代码在 Dosbox 环境下生成汇编程序；

（2）将生成的汇编程序在 masm 汇编生成可执行程序

（3）运用 masm 相关工具对可执行程序进行反汇编，查看寄存器内容，单步调式，检查汇编程序的正确性。

三 编译器原理

（1）编译器工作流程

编程语言对人或机器来说是用来描述计算任务的符号和规则。能够在机器上运行的所有软件都是用某种编程语言编写的程序。但是在程序运行前，必须要将其转化为机器能够执行的格式，编译器的功能就是“转化”。

（2）编译器各阶段的分组和介绍

编译器一般分为前端和后端，主要目的是在多种源语言和多种目标语言的开发过程中，可以灵活搭配组合，消除重复开发的工作量，提高编译系统的开发效率。

前端：依赖于语言并很大程度上独立于目标机器。一般包括语法分析、词法分析、符号表的建立、语义分析、中间代码生成以及相关错误处理。

后端：依赖于目标机器的阶段或某些阶段的某些部分。一般来说，后端完成的任务不依赖于源语言而只依赖于中间语言。主要包括代码优化、代码生成以及相关的错误处理和符号表操作。

符号表管理：记录源程序中符号的必要信息，并加以合理组织，从而在编译器的各个阶段能对它们进行快速、准确的查找和操作。符号表中的某些内容甚至要保留到程序的运行阶段。

错误处理：用户编写的源程序中往往会有一些错误，可分为静态错误和动态错误两类。所谓动态错误，是指源程序中的逻辑错误，它们发生在程序运行的时候，也被称作动态语义错误，如变量取值为零时作为除数，数组元素引用时下标出界等。静态错误又可分为语法错误和静态语义错误。语法错误是指有关语言结构上的错误，如单词拼写错、表达式中缺少操作数、begin 和 end 不匹配等。静态语义错误是指分析源程序时可以发现的语言意义上的错误，如加法的两个操作数中一个是整型变量名，而另一个是数组名等。

词法分析：在一个编译器中，词法分析的作用是将输入的字符串流进行分析，并按一定的规则将输入的字符串流进行分割，从而形成所使用的源程序语言所允许的记号(token)，在这些记号序列将送到随后的语法分析过程中，与此同时将不符合规范的记号识别出来，并产生错误提示信息。通常采用确定的有限状态自动机（Deterministic Finite Automaton，DFA）来构造词法分析工具。已有一些专门的、开源的词法分析程序自动生成器可供免费使用，例如

Lex，其 GNU 版本为 Flex。

语法分析：语法分析的过程是分析词法分析产生的记号序列，并按一定的语法规则识别并生成中间表示形式，以及符号表。符号表记录程序中所使用的标识符及其标识符的属性；同样，将不符合语法规则的记号识别出其位置并产生错误提示语句。词法分析与语法分析及其符号表的关系图如下图所示。目前大多数开源编译器使用的语法分析方法有两类，一种是自顶向下（Top-Down Parsing）的分析方法，另一种是自底向上（Bottom-Up Parsing）的分析方法。目前已有一些专门的、开源的语法分析程序自动生成器可供免费使用，例如 yacc，其 GNU 版本为 bison。在编译器开发中通常联合使用词法分析工具 LEX 和语法分析工具 YACC，这样可以大大减少前端设计工作量。

语义分析：也即静态语法检查，静态语义检查的作用是分析语法分析过程中产生的中间表示形式和符号表，以检查源程序的语义是否与源语言的静态语义属性相符合。由于现有的大部分高级编程语言的语言是语法制导翻译的语言，因此目前通常使用的静态语义检查的方法是语法制导翻译的方法。语法制导翻译的方法是通过语法制导的属性文法来进行程序的语义分析。静态语义检查的根本目的是确认程序是否满足源编程语言要求的静态语义属性，可以理解检查为标识符的声明和使用是否相一致。

中间代码生成：是产生中间代码的过程。所谓“中间代码”是一种结构简单、含义明确的记号系统，这种记号系统复杂性介于源程序语言和机器语言之间，容易将它翻译成目标代码。另外，还可以在中间代码一级进行与机器无关的优化。中间代码可以有若干种形式，它们的共同特征是与具体机器无关。最常用的一种中间代码是三地址码，它的一种实现方式是四元式。三地址码的优点是便于阅读、便于优化。

代码优化：所谓代码优化是指对程序代码进行等价（指不改变程序的运行结果）变换。程序代码可以是中间代码（如四元式代码），也可以是目标代码。等价的含义是使得变换后的代码运行结果与变换前代码运行结果相同。优化的含义是最终生成的目标代码短（运行时间更短、占用空间更小），时空效率优化。原则上，优化可以在编译的各个阶段进行，但最主要的一类是对中间代码进行优化，这类优化不依赖于具体的计算机。

代码生成：目标代码生成是编译的最后一个阶段。目标代码生成器把语法分析后或优化后的中间代码变换成目标代码。目标代码（object code）指计算机科学中编译器或汇编器处理源代码后所生成的代码，它一般由机器代码或接近于机器语言的代码组成。在生成目标代码时要考虑以下几个问题：计算机的系统结构、指令系统、寄存器的分配以及内存的组织等。编译器生成的目标程序代码可以有多种形式：汇编语言、可重定位二进制代码、内存形式。

四 代码设计

4.1 关键代码

① 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
```

② 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;
vector<Variable>          var_table;     //保存声明变量
vector<Target>            target_code;   //保存中间代码
char    lab='A'; //记录跳转标志
char    vab='A'; //记录中间变量
```

③ 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] == '.')
                {
                        j = i;
                        break; } }
        temp = source.substr(0,j) + ".asm";
        return temp;
}
```

④ 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);
}
```

⑤ 判断操作符、界符、空白以及单词类型

```
//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<<"error: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<<"error: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<<"error: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<<"error: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);
}
```

⑥ 词法分析

```
void cifa_analysis(string source,vector<IDwords> &AnalysisResults)
{
    char            ch;
    ifstream    rfile(source.c_str());
    if(!rfile.is_open())
```

```
{
        cout<<"error: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 I
                        }
```

```
        }
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<<"error: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();
}
```

⑦ 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();
                        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," "," ");
        }
    }
}
```

⑧ 语法分析

```
void yufa_analysis(vector<IDwords> &AnalysisResults)
{
    vector<IDwords>::iterator   it=AnalysisResults.begin();
   if(it->word != "main")
    {
        cout<<"error:缺少 main"<<endl;
        exit(-1);
    }
    it = it+3; //跳过“（）”
    if(it->word != "{")
    {
        cout<<"error: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;
        else
        {
            expression(it); //表达式分析
        }
```

```
            it++;
        }
}
```

⑨ 汇编代码生成

```
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);
        }
    }
```

4.2 汇编代码生成

待分析代码，如下所示：

```
1  main()
2  {
3      int a,b,c;
4      int d,e;
5
6      a = 0;
7      b = a+1;
8      c = 10-b;
9      d = b*c;
10     e = c/3;
11 }
```

执行编译器程序，输入源文件路径 C:\Users\simona\Desktop\hxc\test.txt 并运行，得到相关变量声明与初始化信息以及四元式。

```
请输入源文件的路径:
C:\Users\simona\Desktop\hxc\test.txt
变量声明及初始化
a
b
c
d
e
tmpB
tmpC
tmpD
tmpE
中间代码
a  =  0
tmpB  +  a  1
b  =  tmpB
tmpC  -  10  b
c  =  tmpC
tmpD  *  b  c
d  =  tmpD
tmpE  /  c  3
e  =  tmpE

使用结束
请按任意键继续. . .
```

在相应文件夹下发现新生成的 test.asm 汇编代码文件。打开 test.asm 文件，具体汇编代码如下所示：


```
ASSUME CS:codesg,DS:datasg
datasg segment
        a DB '?'
        b DB '?'
        c DB '?'
        d DB '?'
        e DB '?'
        tmpB DB '?'
        tmpC DB '?'
        tmpD DB '?'
        tmpE DB '?'
datasg ends
codesg segment
    start:
        mov AX,datasg
        mov DS,AX
        mov BL,0
        mov a,BL
        mov BL,a
        add BL,I
        mov tmpB,BL
        mov BL,tmpB
        mov b,BL
        mov BL,I0
        sub BL,b
        mov tmpC,BL
        mov BL,tmpC
```

```
        mov c,BL
        mov AL,b
        mov BH,c
        mul BH
        mov BL,I
        div BL
        mov tmpD,AL
        mov BL,tmpD
        mov d,BL
        mov AL,c
        CBW
        mov BL,3
        div BL
        mov tmpE,AL
        mov BL,tmpE
        mov e,BL
        mov ax,4C00H
        int 21H
codesg ends
    end start
```

4.3 将汇编代码在 Dosbox 环境下生成汇编程序

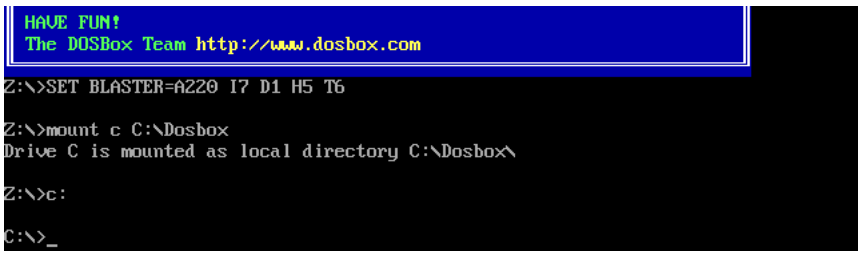
（1）首先在系统里建立一个用于保存汇编工具和汇编文件的目录，在此我直接在 C 盘根目录下建立文件夹 Dosbox，即 C:\Dosbox，然后将 MASM 工具拷贝到该文件夹中。

名称	修改日期	类型	大小
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB

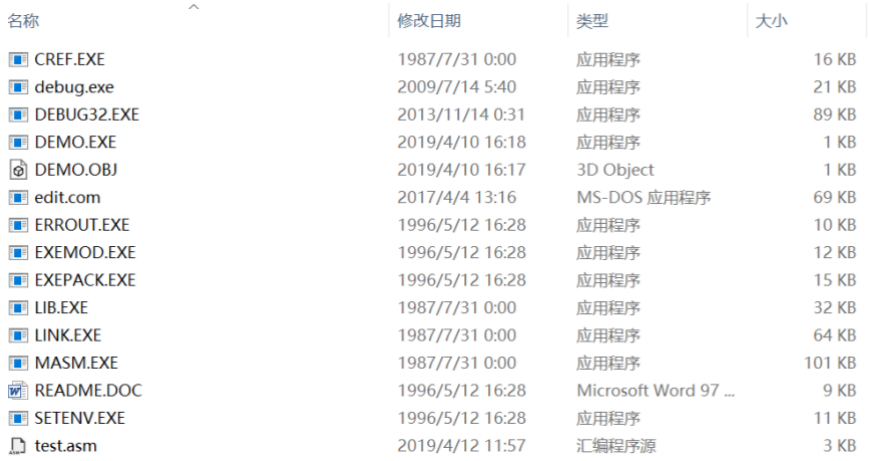
（2）挂载

Z:\> mount c C:\dosbox。　　//将本机工作目录挂载到 dosbox 虚拟 C 盘下

Z:\>c:　　//工作空间转移

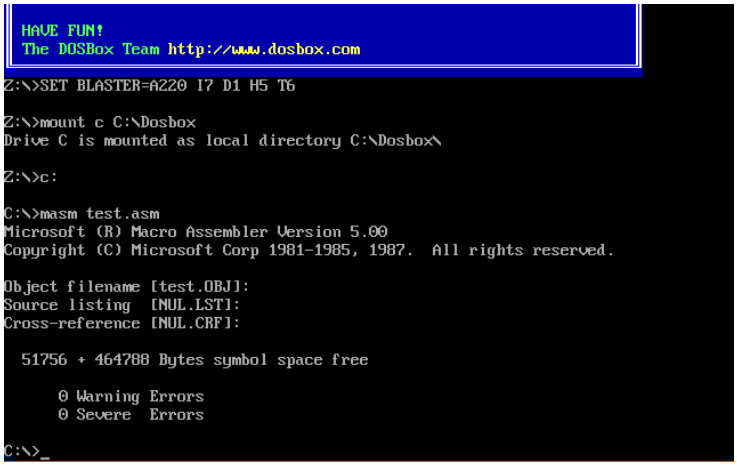


（3）将得到的汇编文件 test.asm 放入 C:\Dosbox 文件夹下。

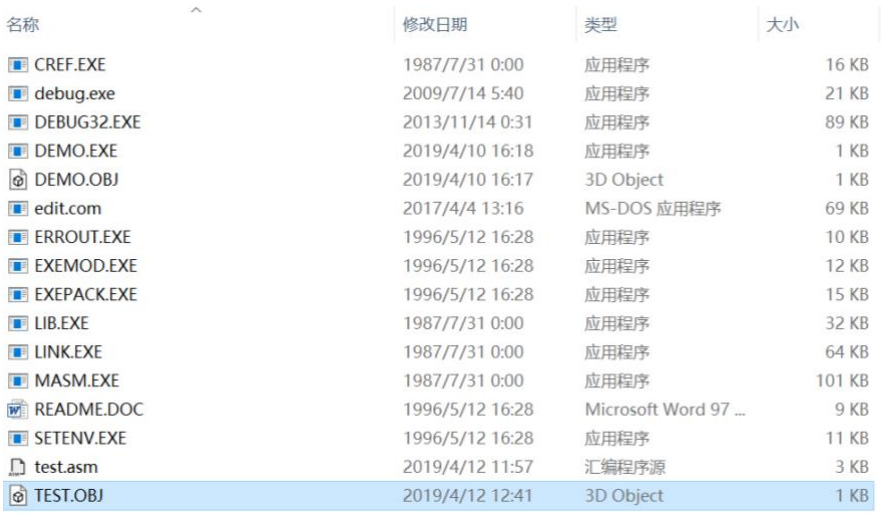


名称	修改日期	类型	大小
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB
test.asm	2019/4/12 11:57	汇编程序源	3 KB

（4）输入指令 masm test.asm，对源程序 test.asm 执行汇编，得到目标程序 TEXT.OBJ。

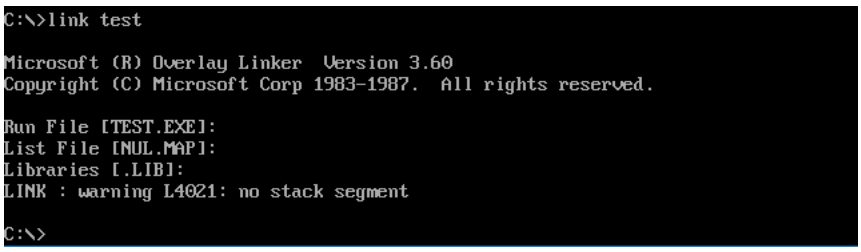


```
HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c C:\Dosbox
Drive C is mounted as local directory C:\Dosbox\

Z:\>c:

C:\>masm test.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [test.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51756 + 464788 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\>_
```



名称	修改日期	类型	大小
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB
test.asm	2019/4/12 11:57	汇编程序源	3 KB
TEST.OBJ	2019/4/12 12:41	3D Object	1 KB

（5）输入指令 link test，链接目标程序，得到可执行程序 test.exe。



```
C:\>link test

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [TEST.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

C:\>
```

名称	修改日期	类型	大小
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB
test.asm	2019/4/12 11:57	汇编程序源	3 KB
TEST.EXE	2019/4/12 12:51	应用程序	1 KB
TEST.OBJ	2019/4/12 12:49	3D Object	1 KB

4.4 对可执行程序进行反汇编

输入指令 dubug test.exe，出现命令符'-'后开始操作。

```
C:\>debug test.exe
-
```

输入指令 –r ，查看程序运行前的寄存器组初始值；

```
C:\>debug test.exe
-r
AX=FFFF  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076B  IP=0000   NV UP EI PL NZ NA PO NC
076B:0000 B86A07        MOV     AX,076A
-_
```

输入指令 –u ，查看程序反汇编代码。从反汇编代码中可看出，变量会被汇编为直接寻址方式，使用变量在数据段内的有效地址表示。

```
-u
076B:0000 B86A07        MOV     AX,076A
076B:0003 8ED8          MOV     DS,AX
076B:0005 B300          MOV     BL,00
076B:0007 881E0000      MOV     [0000],BL
076B:000B 8A1E0000      MOV     BL,[0000]
076B:000F 80C301        ADD     BL,01
076B:0012 881E0500      MOV     [0005],BL
076B:0016 8A1E0500      MOV     BL,[0005]
076B:001A 881E0100      MOV     [0001],BL
076B:001E B30A          MOV     BL,0A
-_
```

输入指令 –t ,单步调试

数据段赋初值

```
-t

AX=076A  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076B  IP=0003   NV UP EI PL NZ NA PO NC
076B:0003 8ED8          MOV     DS,AX
-
```

BL=0H 存入数据栈 DS：0000 --- a 值

```
-t

AX=076A  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0005   NV UP EI PL NZ NA PO NC
076B:0005 B300          MOV     BL,00
-_
```

```
-t

AX=076A  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0007   NV UP EI PL NZ NA PO NC
076B:0007 881E0000      MOV     [0000],BL                       DS:0000=3F
-_
```

取出 a 值，BL=0H

```
-t

AX=076A  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000B   NV UP EI PL NZ NA PO NC
076B:000B 8A1E0000      MOV     BL,[0000]                       DS:0000=00
-_
```

执行 ADD 运算，BL=BL+01H=01H，运算结果存入数据段 DS：0005 --- tmpB 值

```
-t

AX=076A  BX=0000  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000F   NV UP EI PL NZ NA PO NC
076B:000F 80C301        ADD     BL,01
-
```

```
-t

AX=076A  BX=0001  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0012   NV UP EI PL NZ NA PO NC
076B:0012 881E0500      MOV     [0005],BL                       DS:0005=3F
-_
```

取出 tmpB 值，BL=01H，存入数据段 DS:0001 --- b 值

```
-t

AX=076A  BX=0001  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0016   NV UP EI PL NZ NA PO NC
076B:0016 8A1E0500      MOV     BL,[0005]                       DS:0005=01
-_
```

```
-t

AX=076A  BX=0001  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=001A   NV UP EI PL NZ NA PO NC
076B:001A 881E0100      MOV     [0001],BL                       DS:0001=3F
-_
```

BL=0AH

```
-t

AX=076A  BX=0001  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=001E   NV UP EI PL NZ NA PO NC
076B:001E B30A          MOV     BL,0A
-
```

执行 SUB 运算，BL=BL-01H=0AH-01H=09H，存入数据段 DS:0006 --- tmpC 值

```
-t

AX=076A  BX=000A  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0020   NV UP EI PL NZ NA PO NC
076B:0020 2A1E0100      SUB     BL,[0001]                       DS:0001=01
-_
```

```
-t

AX=076A  BX=0009  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0024   NV UP EI PL NZ NA PE NC
076B:0024 881E0600      MOV     [0006],BL                       DS:0006=3F
-_
```

取出 tmpC 值，BL=09H，存入数据段 DS:0002 --- c 值

```
-t

AX=076A  BX=0009  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0028   NV UP EI PL NZ NA PE NC
076B:0028 8A1E0600      MOV     BL,[0006]                       DS:0006=09
-
```

```
-t

AX=076A  BX=0009  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=002C   NV UP EI PL NZ NA PE NC
076B:002C 881E0200      MOV     [0002],BL                       DS:0002=3F
-_
```

取出 b 值，AL=01H

```
-t

AX=076A  BX=0009  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0030   NV UP EI PL NZ NA PE NC
076B:0030 A00100        MOV     AL,[0001]                       DS:0001=01
-_
```

取出 c 值，BH=09H

```
-t

AX=0701  BX=0009  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0033   NV UP EI PL NZ NA PE NC
076B:0033 8A3E0200      MOV     BH,[0002]                       DS:0002=09
-
```

执行 MUL 运算，AX=BH*AL=0009H

```
-t

AX=0701  BX=0909  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0037   NV UP EI PL NZ NA PE NC
076B:0037 F6E7          MUL     BH
-
```

BL=01H

```
-t

AX=0009  BX=0909  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0039   NV UP EI PL NZ NA PE NC
076B:0039 B301          MOV     BL,01
-
```

执行 DIV 运算，AL=AX\BL=09H（商），存入数据段 DS:0007 --- tmpD 值

```
-t

AX=0009  BX=0901  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=003B   NV UP EI PL NZ NA PE NC
076B:003B F6F3          DIV     BL
-_
```

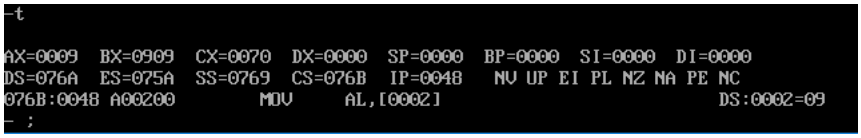
```
-t

AX=0009  BX=0901  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=003D   NV UP EI PL NZ NA PE NC
076B:003D A20700        MOV     [0007],AL                       DS:0007=3F
-_
```

取出 tmpD 值，AL=09H，存入数据段 DS:0003 --- d 值

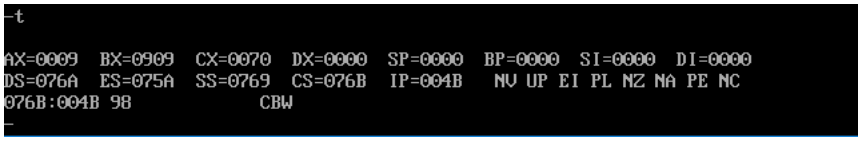
```
-t

AX=0009  BX=0901  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0040   NV UP EI PL NZ NA PE NC
076B:0040 8A1E0700      MOV     BL,[0007]                       DS:0007=09
-_
```

```
-t

AX=0009  BX=0909  CX=0070  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0044   NV UP EI PL NZ NA PE NC
076B:0044 881E0300      MOV     [0003],BL                       DS:0003=3F
-_
```

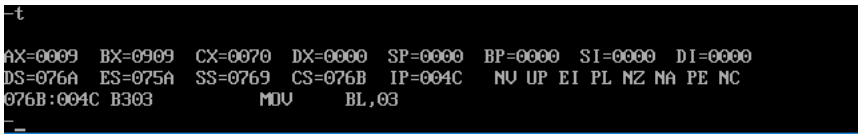
取出 c 值,AL=09H



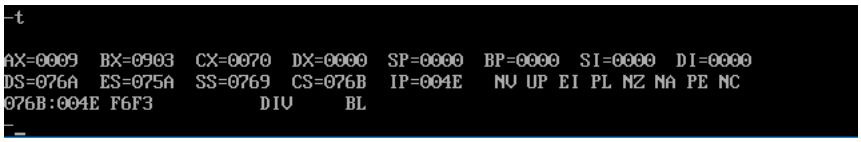
执行 CBW 指令，AX=0009H



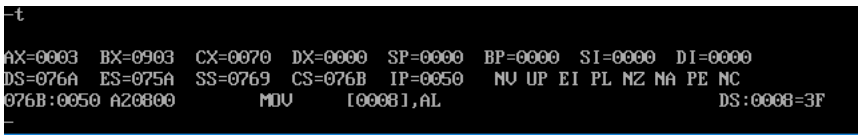
BL=03H



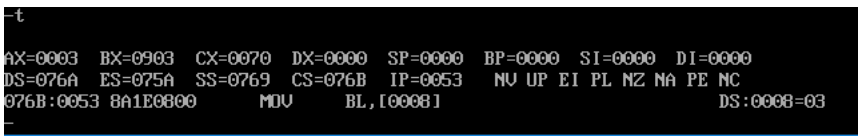
执行 MOV 运算，AL=AX/BL(商);AH=AX/BL(余数)

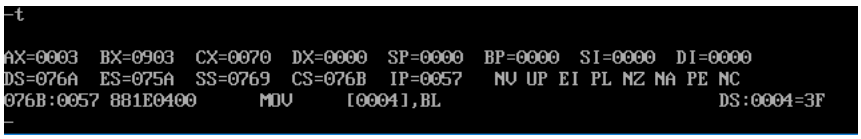


商存入 AL,AL=03H，存入数据段 DS:0008 --- tmpE 值

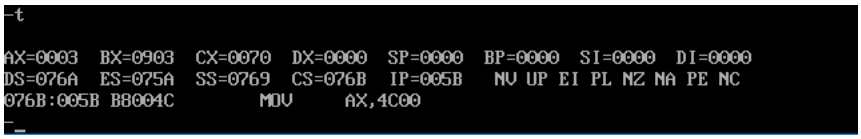


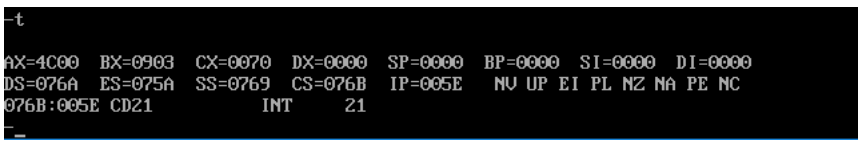
取出 tmpE 值，BL=03H，存入数据段 DS:0004 --- e 值





中断，退出程序





最终计算结果为：

a=0；b=1；c=9；d=9；e=3；该结果与预期结果相符。

五、基于之前实验的编译器后端设计

5.1 代码修改

（1）修改. l 文件

◆ 在用户代码段重新编写主函数 main()，待分析文件为 in.txt，分析结果输出至文件 out.txt，中间代码存于文件 temp.txt，生成的汇编源程序为 ans.asm。

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

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

    fclose(fwrites);

    FILE *fwriteM = fopen("ans.asm","a+");
    fprintf(fwriteM, ";* 计算器实验 *;\n\n");
    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 (?)\nstacks ends
    fprintf(fwriteM, "start:\nMOV AX,CODE\nMOV DS,AX\nMOV DX,00H\n");
    fprintf(fwriteM, "\n");
    fclose(fwriteM);

    yyin = fopen("in.txt", "r");
    yyout = fopen("out.txt", "wt");
    if(!(yyin)||!(yyout))
    {
        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);
    expendfunc();

    FILE *fwriteE = fopen("ans.asm","a+");
    fprintf(fwriteE, "code ends\n");
    fprintf(fwriteE, "end start");
    fclose(fwriteE);

    fclose(yyin);
    fclose(yyout);
    return 0;
}
```

◆ 加入关键字 int 和 main，使两者能被词法分析程序所识别；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;}

 /* built in functions */
"print" { yylval.fn = B_print; return FUNC; }
```

（2）修改.y 文件

◆ 只保留简单的四则运算

```
exp: exp '+' exp          { printf("5 ");$$ = newast('+', $1,$3); }
   | exp '-' exp          { printf("6 ");$$ = newast('-', $1,$3);}
   | exp '*' exp          { printf("7 ");$$ = newast('*', $1,$3); }
   | exp '/' exp          { printf("8 ");$$ = newast('/', $1,$3); }
   | '|' exp              { printf("9 ");$$ = newast('|', $2, NULL); }
   | '(' exp ')'          { printf("10 ");$$ = $2; }
   | '-' exp %prec UMINUS { printf("11 ");$$ = newast('M', $2, NULL); }
   | NUMBER               { printf("12 ");$$ = newnum($1); }
   | FUNC '(' explist ')' { printf("13 ");$$ = newfunc($1, $3); }
   | NAME                 { printf("14 ");$$ = newref($1); }
   | NAME '=' exp         { printf("15 ");$$ = newasgn($1, $3); }
;
```

◆ 顶部规约规则改为 calclist → calclist INTT MAIN ( ) { stmt }

```
calclist: /* nothing */{printf("17 ");}
  | calclist INTT MAIN '(' ')' '{' stmt  '}'
    initt();
    printf("18 ");
    if(debug) dumpast($7, 0);
    else {
    char ch[100];
    gcvt(eval($7)->v,10,ch);
    fprintf(yyout,"%s\n",ch);
    treefree($7);
    FILE *fwritez = fopen("temp.txt","a+");
    fprintf(fwritez, "\n");
    fclose(fwritez);
     }
    }
  | calclist error { printf("19 ");yyerrok; fprintf(yyout,""); }
```

（3）修改.h 文件

◆ 添加结构体 adress，用于定义计算结果。

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

type: 类型(变量&中间变量)

pos: 汇编表中的位置

v: 结果数值

◆ 添加结构体 four，用于定义四元式。（操作符、操作数、类型、位置）

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

◆ 最后，在符号表结构体 symbol 中新增 pos 属性。

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

（4）修改 funcs.c 文件

funcs.c 文件的关键在于求值函数 eval()，对于该函数，我们既要执行计算功能，返回 struct address* 类型值，又要在计算的同时生成对应的四元式和汇编代码。

在此举一个简单的示例，若匹配到的节点类型值为'K'，首先计算节点值并填充域，接着向文件 temp.txt 中写入对应的中间代码，最后调用 MOV（）函数生成汇编代码。

```
switch(a->nodetype) {
  /* constant常量 */
case 'K': /*计算值并填充域*/
          v = ((struct numval *)a)->number;
          re->v=v;
          re->type='T';
          re->pos=ti++;
          /*往temp.txt文件中写入相应的中间代码*/
          int pv = v;
          FILE *fwrite = fopen("temp.txt","a+");
          fprintf(fwrite, "MOV,%d,,%c%d\n", pv,re->type,re->pos);
          fclose(fwrite);
          /*调用MOV()函数，生成汇编代码*/
          f->t3 = 'T';
          f->p1 = pv;
          f->p3 = re->pos;
          MOV(f);
          break;
```

用于生成汇编代码的函数如下：

· MOV

```
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);
  }
}
```

· ADD

```
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);
}
```

· SUB

```
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);
}
```

· MUL

```
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 AX,BX\n");
  fprintf(fwriteM, "MOV BX,%d\n",p->p3);
  fprintf(fwriteM, "MOV %c[BX],AX\n",p->t3);
  fprintf(fwriteM, "MOV BX,%d\n", p->p3-1);
  fprintf(fwriteM, "MOV %c[BX],DX\n",p->t3);
  fprintf(fwriteM, "\n");
  fclose(fwriteM);
}
```

· DIV

```
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 BX,%d\n", p->p1 - 1);
  fprintf(fwriteM, "MOV DX,%C[BX]\n",p->t1);
  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 AX,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);
}
```

· NEG

```
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);
}
```

· PRT

```
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);
}
```

4.2 生成汇编文件

首先，为了操作方便，编写脚本文件 build.sh，具体内容如下：

```
flex -ofb4-1.lex.c fb4-1.l
bison -d fb4-1.y
gcc -o x86 fb4-1.tab.c fb4-1.lex.c expendfunc.c fb4-1funcs.c -lm
```

事先在文件 in.txt 中输入待分析的文件内容，具体如下：

```
int main ()
{ a = 1;
  b=a+1;
  c=10-a;
  d=c/3;

  print(a);
  print(b);
  print(c);
  print(d);
}
```

终端输入指令 sh com.sh，生成可执行文件 parse，进而调用可执行文件 parse，对输入文件 in.txt 进行处理，计算结果如下所示：

```
root@simona-virtual-machine:/home/simona# sh com.sh
fb4-1.tab.c: In function 'yyparse':
fb4-1.tab.c:1166:16: warning: implicit declaration of function 'yylex' [-Wimplic
it-function-declaration]
      yychar = yylex ();
               ^
root@simona-virtual-machine:/home/simona# ./parse <in.txt
=    1
=    2
=    9
=    3
root@simona-virtual-machine:/home/simona#
```

生成的中间代码文件：

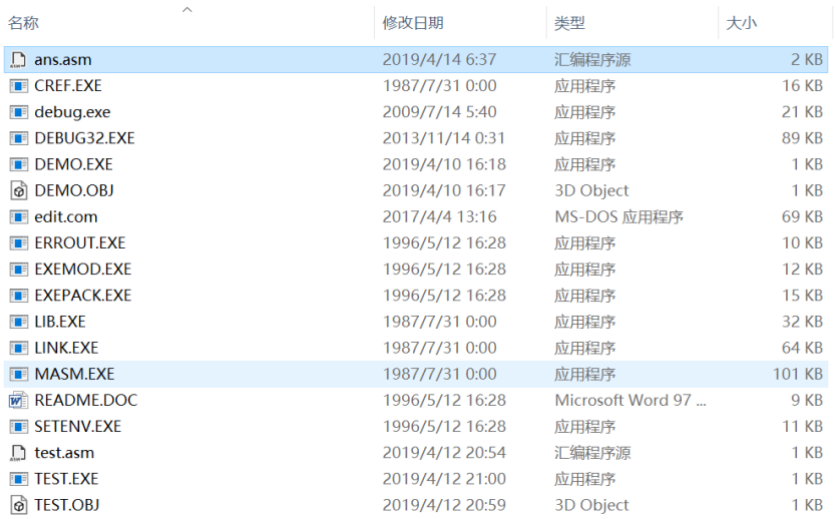
```
startMOV,1,,T0
MOV,T0,,H0
MOV,1,,T0
ADD,H0,T0,T1
MOV,T1,,H1
MOV,10,,T0
SUB,T0,H0,T1
MOV,T1,,H2
MOV,3,,T0
DIV,H2,T0,T1
MOV,T1,,H3
print,H0, ,
print,H1, ,
print,H2, ,
print,H3, ,

end
```

生成的汇编代码如下：

```
data segment
T DW 1000 DUP(0)
H DW 1000 DUP(0)
data ends

stacks segment stack
db 1000 dup (?)
stacks ends
code segment
assume cs:code,ds:data,ss:stacks,es:data

start:

MOV BX,1
MOV T[BX],1

MOV BX,1
MOV AX,T[BX]
MOV BX,1
MOV H[BX],AX

MOV BX,1
MOV T[BX],1

MOV BX,1
MOV AX,H[BX]
MOV BX,1
MOV DX,T[BX]
MOV BX,DX
ADD AX,BX
MOV BX,3
MOV T[BX],AX

MOV BX,3
MOV AX,T[BX]
MOV BX,3
MOV H[BX],AX

MOV BX,1
MOV T[BX],10

MOV BX,1
MOV AX,T[BX]
MOV BX,1
MOV DX,H[BX]
MOV BX,DX
SUB AX,BX
MOV BX,3
MOV T[BX],AX

MOV BX,3
MOV AX,T[BX]
MOV BX,5
MOV H[BX],AX

MOV BX,1
MOV T[BX],3

MOV BX,5
MOV AX,H[BX]
MOV BX,4
MOV DX,H[BX]
MOV BX,1
MOV CX,T[BX]
MOV BX,CX
DIV AX,BX
MOV BX,3
MOV T[BX],AX

MOV BX,3
MOV AX,T[BX]
MOV BX,7
MOV H[BX],AX

MOV BX,1
MOV AX,H[BX]
call wrhax

MOV BX,3
MOV AX,H[BX]
call wrhax

MOV BX,5
MOV AX,H[BX]
call wrhax

MOV BX,7
MOV AX,H[BX]
call wrhax
```

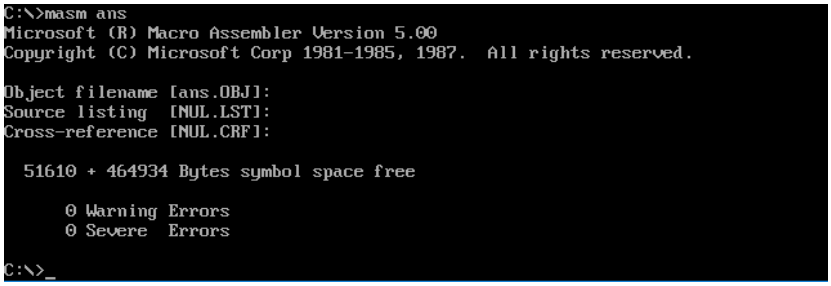
4.3 在 Dosbox 中进行验证

（1）将得到的汇编文件 res.asm 放入 C:\Dosbox 文件夹下。



名称	修改日期	类型	大小
ans.asm	2019/4/14 6:37	汇编程序源	2 KB
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB
test.asm	2019/4/12 20:54	汇编程序源	1 KB
TEST.EXE	2019/4/12 21:00	应用程序	1 KB
TEST.OBJ	2019/4/12 20:59	3D Object	1 KB

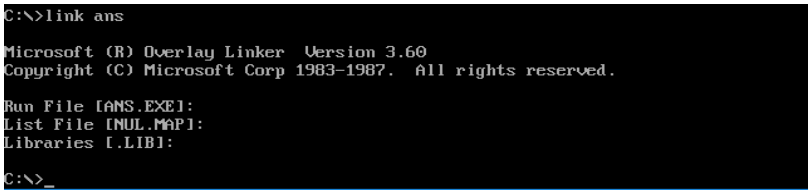
（2）输入指令 masm ans.asm，对源程序 ans.asm 执行汇编，得到目标程序 ANS.OBJ。



```
C:\>masm ans
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [ans.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51610 + 464934 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\>_
```

名称	修改日期	类型	大小
ans.asm	2019/4/14 6:37	汇编程序源	2 KB
ANS.OBJ	2019/4/14 6:41	3D Object	1 KB
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB
edit.com	2017/4/4 13:16	MS-DOS 应用程序	69 KB
ERROUT.EXE	1996/5/12 16:28	应用程序	10 KB
EXEMOD.EXE	1996/5/12 16:28	应用程序	12 KB
EXEPACK.EXE	1996/5/12 16:28	应用程序	15 KB
LIB.EXE	1987/7/31 0:00	应用程序	32 KB
LINK.EXE	1987/7/31 0:00	应用程序	64 KB
MASM.EXE	1987/7/31 0:00	应用程序	101 KB
README.DOC	1996/5/12 16:28	Microsoft Word 97 ...	9 KB
SETENV.EXE	1996/5/12 16:28	应用程序	11 KB
test.asm	2019/4/12 20:54	汇编程序源	1 KB
TEST.EXE	2019/4/12 21:00	应用程序	1 KB
TEST.OBJ	2019/4/12 20:59	3D Object	1 KB

（3）输入指令 link ans，链接目标程序，得到可执行程序 ans.exe。



```
C:\>link ans

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [ANS.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\>_
```

名称	修改日期	类型	大小
ans.asm	2019/4/14 6:37	汇编程序源	2 KB
ANS.EXE	2019/4/14 6:42	应用程序	6 KB
ANS.OBJ	2019/4/14 6:41	3D Object	1 KB
CREF.EXE	1987/7/31 0:00	应用程序	16 KB
debug.exe	2009/7/14 5:40	应用程序	21 KB
DEBUG32.EXE	2013/11/14 0:31	应用程序	89 KB
DEMO.EXE	2019/4/10 16:18	应用程序	1 KB
DEMO.OBJ	2019/4/10 16:17	3D Object	1 KB

（4）执行反汇编，输入指令 dubug res.exe，出现命令符'-'后开始操作。

输入指令 –r ，查看程序运行前的寄存器组初始值:

```
C:\>debug ans.exe
-r
AX=FFFF  BX=0000  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0000   NV UP EI PL NZ NA PO NC
08A3:0000 BB0100        MOV     BX,0001
-
```

输入指令 –u ，查看程序反汇编代码:

```
-u
08A3:0000 BB0100        MOV     BX,0001
08A3:0003 C787000001000 MOV     WORD PTR [BX+0000],0001
08A3:0009 BB0100        MOV     BX,0001
08A3:000C 8B870000      MOV     AX,[BX+0000]
08A3:0010 BB0100        MOV     BX,0001
08A3:0013 8987D007      MOV     [BX+07D0],AX
08A3:0017 BB0100        MOV     BX,0001
08A3:001A C787000001000 MOV     WORD PTR [BX+0000],0001
-
```

输入指令 –t，单步调试：(仅以部分代码为例)

a 的数值=1，存于数据段寄存器 DS:0001

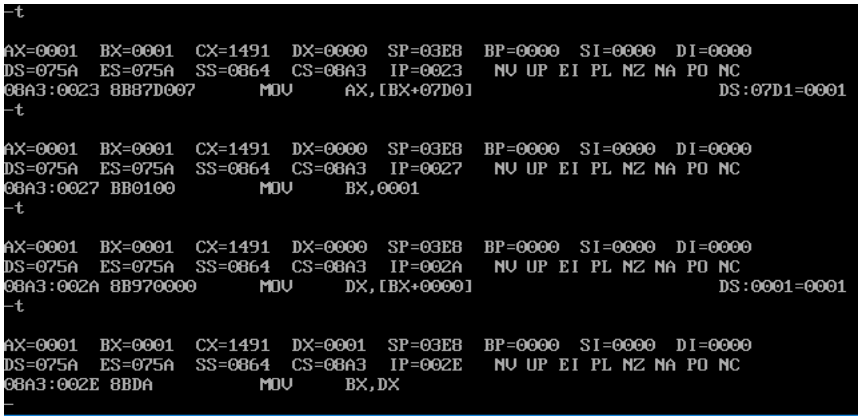
取出数值于 BX，BX=1,T[BX]=1　　AX=T[BX]=1

```
-t

AX=FFFF  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0003   NV UP EI PL NZ NA PO NC
08A3:0003 C787000001000 MOV     WORD PTR [BX+0000],0001          DS:0001=FF20
-t

AX=FFFF  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0009   NV UP EI PL NZ NA PO NC
08A3:0009 BB0100        MOV     BX,0001
-t

AX=FFFF  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=000C   NV UP EI PL NZ NA PO NC
08A3:000C 8B870000      MOV     AX,[BX+0000]                     DS:0001=0001
-t

AX=0001  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0010   NV UP EI PL NZ NA PO NC
08A3:0010 BB0100        MOV     BX,0001
-
```

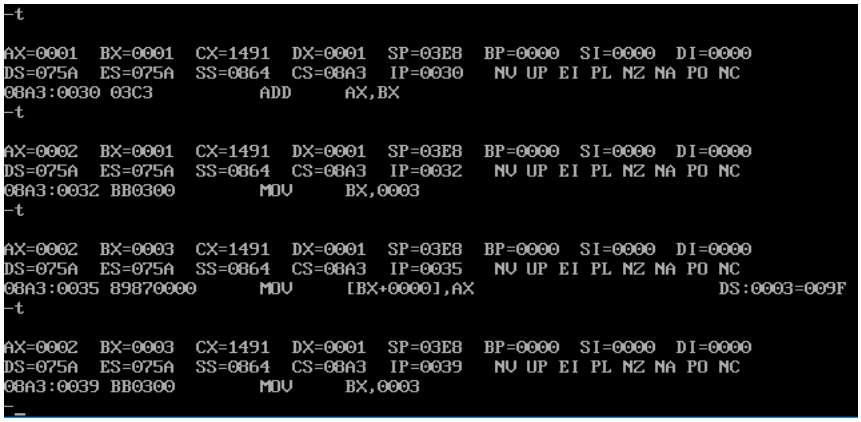
AX=T[BX]=1　　H[BX]=AX=1

```
-t

AX=0001  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0013   NV UP EI PL NZ NA PO NC
08A3:0013 8987D007      MOV     [BX+07D0],AX                     DS:07D1=0000
-t

AX=0001  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0017   NV UP EI PL NZ NA PO NC
08A3:0017 BB0100        MOV     BX,0001
-t

AX=0001  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=001A   NV UP EI PL NZ NA PO NC
08A3:001A C787000001000 MOV     WORD PTR [BX+0000],0001          DS:0001=0001
-t

AX=0001  BX=0001  CX=1491  DX=0000  SP=03E8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0864  CS=08A3  IP=0020   NV UP EI PL NZ NA PO NC
08A3:0020 BB0100        MOV     BX,0001
-
```

AX=H[BX]=I　DX=T[BX]=I　BX=DX=I



执行 ADD 运算，AX=AX+BX=1+1=2 --- b 数值
存于数据段寄存器 DS:0003 位置



五、实验体会

这次实验让我了解了 c 编译器后端的基本原理以及相应的操作流程。编译器后端将前期所得抽象语法树转换成四元式，进而生成汇编代码。得到的汇编代码文件在 Dosbox 环境下编译、链接进而生成可执行程序，至此便可以直接运行。

其实这次实验过程中请教了一些同学，自己也上网搜集了很多的资料，重点学习了抽象语法树、四元式、汇编代码之间的转换思路，虽然过程比较艰难，但还是有了不少的收获。此外，我也是第一次接触 Dosbox 这个软件，我尝试学习和使用了 masm 汇编工具，掌握了一些它的基本的操作指令，并在后续的单步调试过程中验证了计算的正确性。总体感觉，Dosbox 软件的使用还是非常的方便和快捷的。

这次实验带给我的收获不小，我会在今后的学习中不断钻研和深入，争取有更透彻的理解。