中国矿业大学计算机学院

系统软件开发实践报告

课程名称 系统软件开发实践

报告时间 2020 年 5 月 25 日

学生姓名 陆玺文

学　　号 03170908

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

1、目标代码生成

1.1 实验目的

本实验是系统软件课程设计最后一次实验，其任务是在词法分析、语法分析、语义分析和中间代码生成的基础上，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。

1.2 实验内容

1. 确定目标机的体系结构，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技术。
2. 编写目标代码的汇编生成程序可以将在此之前已经得到的中间代码映射为相应的汇编代码。
3. 实现错误处理程序，包括词法错误、语法错误、编译警告和运行异常。
4. 生成可执行文件，最终获得一个 C 自己的编译器，包含基本的前端和后端功能。
5. 实现链接器，生成可运行程序。

1.3 任务 1：X86 体系学习

X86 架构（The X86 architecture）是微处理器执行的计算机语言指令集，指 intel 通用计算机系列的标准编号缩写，也标识一套通用的计算机指令集合。

1.3.1 Emu8086 模拟器使用

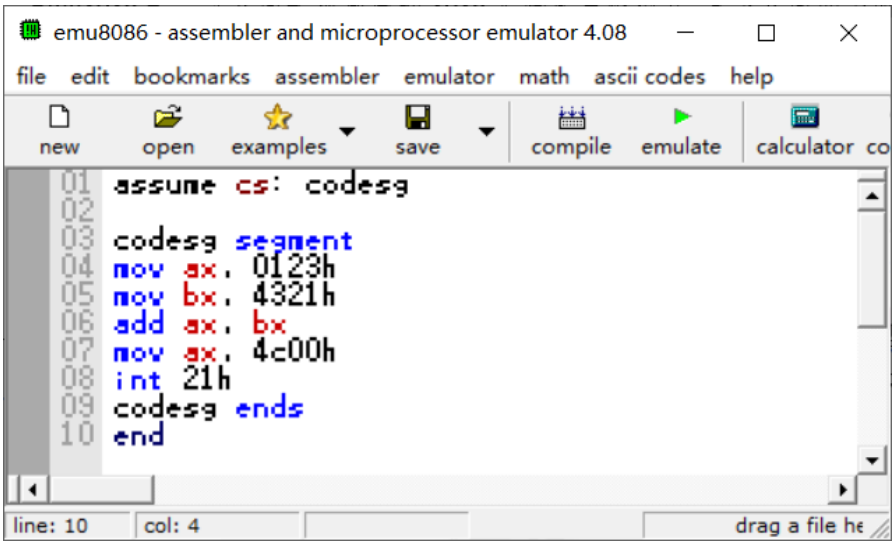

图 1-1 Emu8086 使用界面

Emu8086 是一款界面美观易用的 8086 汇编语言模拟器，其主界面如图 1-1 所示。

在编辑界面输入代码之后，可以通过 compile 进行编译，单击 emulate 进行仿真。仿真界面如图 1-2 所示。其中，single step 可以进行单步执行，run 可以快速执行完整个程序。在 registers 部分，可以观察寄存器状态值的变化，方便追踪程序的运行。

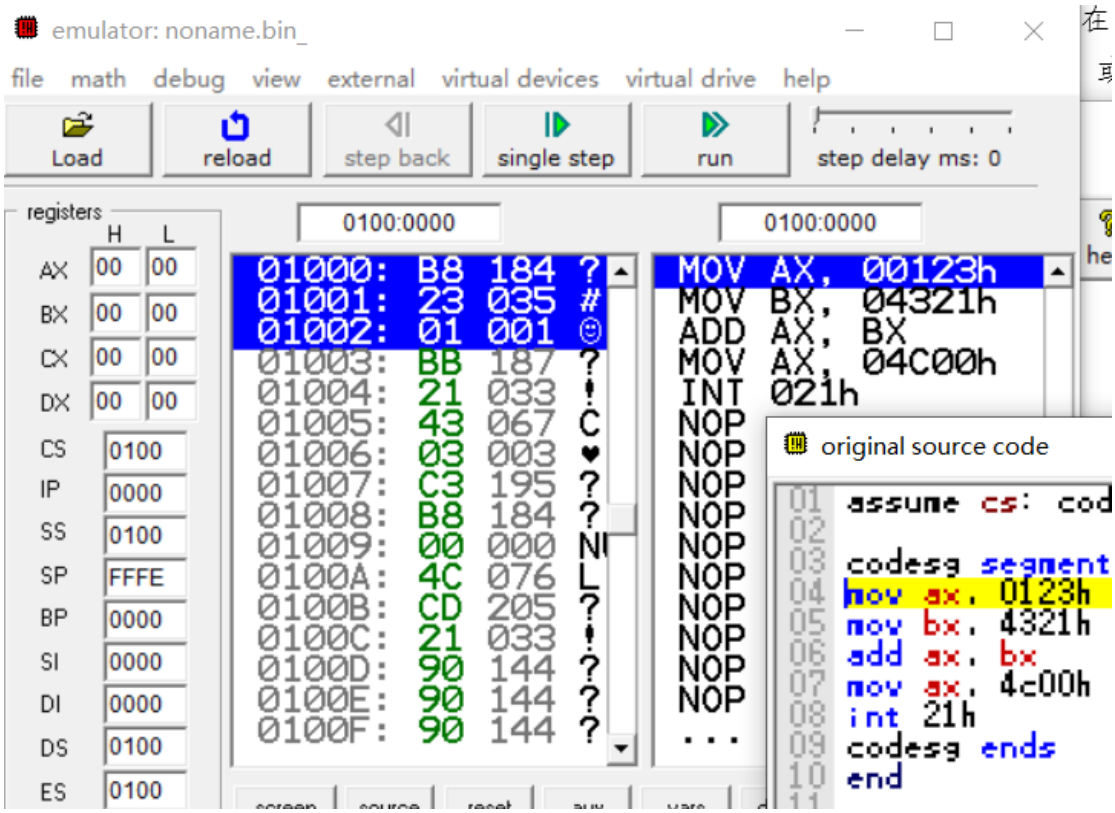

图 1-2　识别文法 G[S1]特定项目集 S

1.3.2 体系结构相关知识

有关 8086 体系的指令选择、寄存器选择以及存储器分配管理技术在《微机原理与接口》课程中有过详细学习。

1.4 任务 2：汇编程序生成

本部分实验内容基础部分使用了开源 Git 项目 Compiler，地址：https://github.com/xiangxianzhang/Compiler 。

1.4.1 四元式生成

四元式作为连接编译器前后端的桥梁，有着十分重要的作用。在语法分析的过程中，得到了遍历之后的抽象语法树 AST，四元式可以在此基础上遍历树

节点生成。

核心部分代码如代码 1-1 所示。

代码 1-1 四元式生成函数（actionSignOP()）

```
1.  private void actionSignOP(){
2.      if(top.name.equals("@ADD_SUB")){
3.          if(OP!=null&&(OP.equals("+")||OP.equals("-"))){
4.              ARG2=semanticStack.pop();
5.              ARG1=semanticStack.pop();
6.              RES=newTemp();
7.              //fourElemCount++;
8.              FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,ARG2,RES);
9.              fourElemList.add(fourElem);
10.             L.value=RES;
11.             semanticStack.push(L.value);
12.             OP=null;
13.         }
14.         analyseStack.remove(0);
15.
16.     }else if(top.name.equals("@ADD")){
17.         OP="+";
18.         analyseStack.remove(0);
19.     }else if(top.name.equals("@SUB")){
20.         OP="-";
21.         analyseStack.remove(0);
22.     }else if(top.name.equals("@DIV_MUL")){
23.         if(OP!=null&&(OP.equals("*")||OP.equals("/"))){
24.             ARG2=semanticStack.pop();
25.             ARG1=semanticStack.pop();
26.             RES=newTemp();
27.             //fourElemCount++;
28.             FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,ARG2,RES);
29.             fourElemList.add(fourElem);
30.             T.value=RES;
31.             semanticStack.push(T.value);
32.             OP=null;
33.         }
34.         analyseStack.remove(0);
35.     }
36.     else if(top.name.equals("@DIV")){
37.         OP="/";
38.         analyseStack.remove(0);
```

```
39.         }
40.     else if(top.name.equals("@MUL")){
41.         OP="*";
42.         analyseStack.remove(0);
43.     }else if(top.name.equals("@TRAN_LF")){
44.         F.value=L.value;
45.         //semanticStack.push(F.value);
46.         analyseStack.remove(0);
47.     }else if(top.name.equals("@ASS_F")){
48.         F.value=firstWord.value;
49.         semanticStack.push(F.value);
50.         analyseStack.remove(0);
51.     }else if(top.name.equals("@ASS_R")){
52.         R.value=firstWord.value;
53.         semanticStack.push(R.value);
54.         analyseStack.remove(0);
55.     }else if(top.name.equals("@ASS_Q")){
56.         Q.value=firstWord.value;
57.         semanticStack.push(Q.value);
58.         analyseStack.remove(0);
59.     }
60.     else if(top.name.equals("@ASS_U")){
61.         U.value=firstWord.value;
62.         semanticStack.push(U.value);
63.         analyseStack.remove(0);
64.     }else if(top.name.equals("@SINGLE")){
65.         if(for_op.peek()!=null){
66.             ARG1=semanticStack.pop();
67.             RES=ARG1;
68.             //fourElemCount++;
69.             FourElement fourElem=new
    FourElement(++fourElemCount,for_op.pop(),ARG1,"/",RES);
70.             fourElemList.add(fourElem);
71.         }
72.         analyseStack.remove(0);
73.     }else if(top.name.equals("@SINGLE_OP")){
74.         for_op.push(firstWord.value);
75.         analyseStack.remove(0);
76.     }else if(top.name.equals("@EQ")){
77.         OP="=";
78.         ARG1=semanticStack.pop();
79.         RES=semanticStack.pop();;
80.         //fourElemCount++;
81.         FourElement fourElem=new
```

```
    FourElement(++fourElemCount,OP,ARG1,"/",RES);
82.         fourElemList.add(fourElem);
83.         OP=null;
84.         analyseStack.remove(0);
85.     }
86.     else if(top.name.equals("@EQ_U'")){
87.         OP="=";
88.         ARG1=semanticStack.pop();
89.         RES=semanticStack.pop();;
90.         //fourElemCount++;
91.         FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,"/",RES);
92.         fourElemList.add(fourElem);
93.         OP=null;
94.         analyseStack.remove(0);
95.     }else if(top.name.equals("@COMPARE")){
96.         ARG2=semanticStack.pop();
97.         OP=semanticStack.pop();
98.         ARG1=semanticStack.pop();
99.         RES=newTemp();
100         //fourElemCount++;
101         FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,ARG2,RES);
102         fourElemList.add(fourElem);
103         G.value=RES;
104         semanticStack.push(G.value);
105         OP=null;
106         analyseStack.remove(0);
107     }else if(top.name.equals("@COMPARE_OP")){
108         D.value=firstWord.value;
109         semanticStack.push(D.value);
110         analyseStack.remove(0);
111     }else if(top.name.equals("@IF_FJ")){
112         OP="FJ";
113         ARG1=semanticStack.pop();
114         FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,"/",RES);
115         if_fj.push(fourElemCount);
116         fourElemList.add(fourElem);
117         OP=null;
118         analyseStack.remove(0);
119     }else if(top.name.equals("@IF_BACKPATCH_FJ")){
120         backpatch(if_fj.pop(), fourElemCount+2);
121         analyseStack.remove(0);
```

```
122     }else if(top.name.equals("@IF_RJ")){
123         OP="RJ";
124         FourElement fourElem=new
    FourElement(++fourElemCount,OP,"/","/","/");
125         if_rj.push(fourElemCount);
126         fourElemList.add(fourElem);
127         OP=null;
128         analyseStack.remove(0);
129     }else if(top.name.equals("@IF_BACKPATCH_RJ")){
130         backpatch(if_rj.pop(), fourElemCount+1);
131         analyseStack.remove(0);
132     }else if(top.name.equals("@WHILE_FJ")){
133         OP="FJ";
134         ARG1=semanticStack.pop();
135         FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,"/","/");
136         while_fj.push(fourElemCount);
137         fourElemList.add(fourElem);
138         OP=null;
139         analyseStack.remove(0);
140     }else if(top.name.equals("@WHILE_RJ")){
141         OP="RJ";
142         RES=(while_fj.peek()-1)+"";
143         FourElement fourElem=new
    FourElement(++fourElemCount,OP,"/","/",RES);
144         for_rj.push(fourElemCount);
145         fourElemList.add(fourElem);
146         OP=null;
147         analyseStack.remove(0);
148     }else if(top.name.equals("@WHILE_BACKPATCH_FJ")){
149         backpatch(while_fj.pop(), fourElemCount+1);
150         analyseStack.remove(0);
151     }else if(top.name.equals("@FOR_FJ")){
152         OP="FJ";
153         ARG1=semanticStack.pop();
154         FourElement fourElem=new
    FourElement(++fourElemCount,OP,ARG1,"/","/");
155         for_fj.push(fourElemCount);
156         fourElemList.add(fourElem);
157         OP=null;
158         analyseStack.remove(0);
159     }else if(top.name.equals("@FOR_RJ")){
160         OP="RJ";
161         RES=(for_fj.peek()-1)+"";
```

```
162         FourElement fourElem=new
    FourElement(++fourElemCount,OP,"/","/",RES);
163         for_rj.push(fourElemCount);
164         fourElemList.add(fourElem);
165         OP=null;
166         analyseStack.remove(0);
167     }else if(top.name.equals("@FOR_BACKPATCH_FJ")){
168         backpatch(for_fj.pop(), fourElemCount+1);
169         analyseStack.remove(0);
170     }
171 }
```

对于代码 1-2，其工作之后的结果图如图 1-3 所示。

代码 1-2　测试代码 test2.c

```
1.  void main(){
2.      int a=1;
3.      int b=2;
4.      int c;
5.      c=a+b;
6.  }
7.  #
```

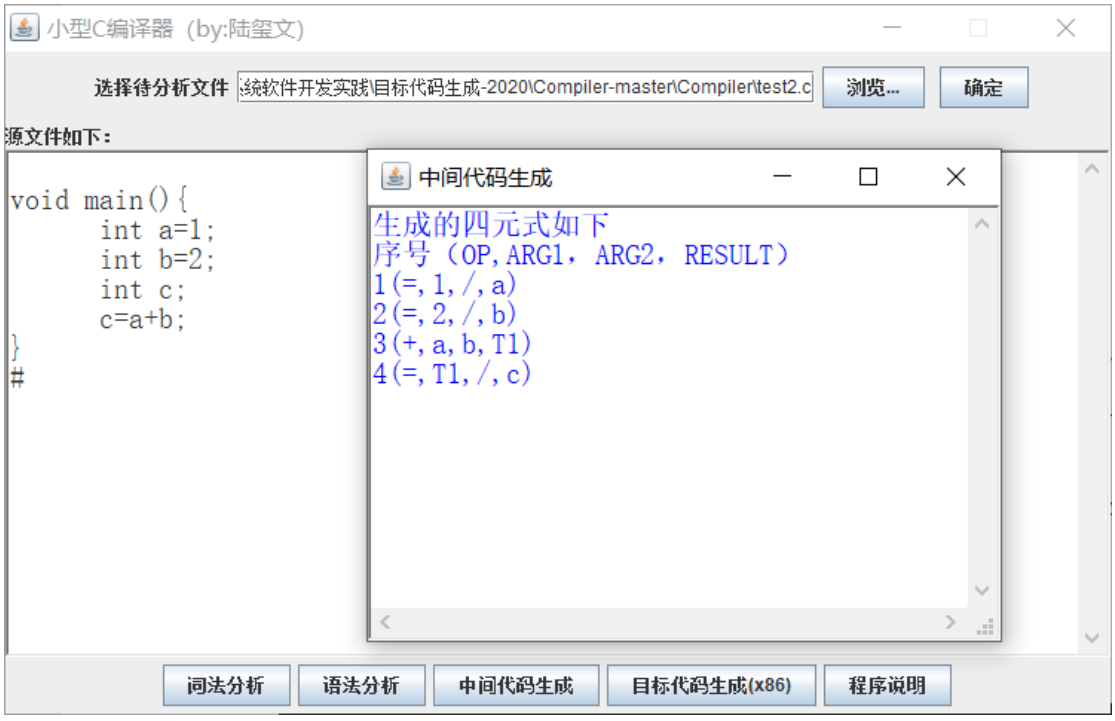

图 1-3　四元式生成图

1.4.2 汇编语句生成

在上面的四元式基础上，继续编写针对四元式输入的汇编代码输出语句。其中需要考虑寄存器选择以及指令选择。

囿于能力以及经验上的欠缺，寄存器选择部分，仅采用了 map 结构将所有出现的变量绑定寄存器，并未针对寄存器进行优化以及分配，因而只能在变量数较少时获得较好的演示效果。寄存器分配代码如代码 1-3 所示

代码 1-3 寄存器分配代码

```
1.  //  寄存器分配，用于存储变量与寄存器对应编号
2.  //  寄存器分配还需好好设计，暂且使用数值自增
3.      private Map<String,String> registerVariateMap = new
    HashMap<String, String>();
4.      private int registerNum=65;
5.      public String registerAllocation(String vatiate) {
6.          if(isNumericzidai(vatiate)) {
7.              return vatiate;
8.          }
9.          if(registerVariateMap.get(vatiate)!=null) {
10.
11.         } else{
12.             StringBuffer sbu = new StringBuffer();
13.             sbu.append((char)registerNum++);
14.             sbu.append('X');
15.             System.out.print("vatiate: "+vatiate);
16.             System.out.println(", "+sbu.toString());
17.             registerVariateMap.put(vatiate, sbu.toString());
18.         }
19.         return registerVariateMap.get(vatiate);
20.     }
```

完整的读入四元式并最终转为汇编的代码在

代码 1-4 中展示了出来。

代码 1-4 输出汇编代码

```
1.      public String outputHuiBian() throws IOException{
2.
3.          File file=new File("./output/");
4.          if(!file.exists()){
5.              file.mkdirs();
6.              file.createNewFile();//如果这个文件不存在就创建它
7.          }
```

```
8.          String path=file.getAbsolutePath();
9.          FileOutputStream fos=new FileOutputStream(path+"/HuiBian.txt");
10.         BufferedOutputStream bos=new BufferedOutputStream(fos);
11.         OutputStreamWriter osw1=new OutputStreamWriter(bos,"utf-8");
12.         PrintWriter pw1=new PrintWriter(osw1);
13.         pw1.println("ASSUME CS:codesg");
14.         pw1.println("codesg segment");
15.
16. //      FourElement temp;
17. //      for(int i=0;i<fourElemList.size();i++){
18. //          temp=fourElemList.get(i);
19. //
        pw1.println(temp.id+"("+temp.op+","+temp.arg1+","+temp.arg2+","+tem
    p.result+")");
20. //      }
21.
22.         try {
23.             String str[] =readFile(fileName).split("\n");
24.             String temp1 = null;
25.             for(int i = 2; i < str.length; i++)
26.             {
27.                 String temp[] = str[i].split(",");
28.                 if(temp[0].charAt(temp[0].length() -1) == '='){
29.                     String src = temp[3].substring(0,temp[3].length() -
    1);
30.                     temp1 = "MOV " + registerAllocation(src) + "," +
    registerAllocation(temp[1]) + "\n";
31.                 }
32.                 else if(temp[0].charAt(temp[0].length() -1) == '+' &&
    temp[0].charAt(temp[0].length() -2) == '+'){
33.                     temp1 = "INC "  + registerAllocation(temp[1]) +
    "\n";
34.                 }
35.                 else if(temp[0].charAt(temp[0].length() -1) == '+'){
36.                     String dst = registerAllocation(temp[1]);
37.                     String src = registerAllocation(temp[2]);

38.                     String dsf =
    registerAllocation(temp[3].substring(0,temp[3].length() - 1));
39.                     temp1 = "ADD " + dst + "," + src + "\n";
40.                     temp1+= "\t"+"MOV " + dsf + "," + dst + "\n";
41.                 }
42.                 else if(temp[0].charAt(temp[0].length() -1) == '-'){
43.                     temp1 = "SUB " +
```

```
    temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
44.                 }
45.                 else if(temp[0].charAt(temp[0].length() -1) == '*'){
46.                     temp1 = "MUL " +
    temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
47.                 }
48.                 else if(temp[0].charAt(temp[0].length() -1) == '/'){
49.                     temp1 = "DIV "  +
    temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
50.                 }
51.                 else if(temp[0].charAt(temp[0].length() -1) == 'J' &&
    temp[0].charAt(temp[0].length() -2) == 'R'){
52.                     temp1 = "JMP " +
    temp[3].substring(0,temp[3].length() - 1)  + "\n";
53.                 }
54.                 else if(temp[0].charAt(temp[0].length() -1) == 'J' &&
    temp[0].charAt(temp[0].length() -2) == 'F'){
55.                     temp1 = "JZ " + temp[3].substring(0,temp[3].length()
    - 1)  + "\n";
56.                 }
57.                 else if(temp[0].charAt(temp[0].length() -1) == '>'){
58.                     temp1 = "JG " + temp[3].substring(0,temp[3].length()
    - 1)  + "\n";
59.                 }
60.                 else if(temp[0].charAt(temp[0].length() -1) == '<'){
61.                     temp1 = "JL " + temp[3].substring(0,temp[3].length()
    - 1)  + "\n";
62.                 }
63.                 pw1.print("\t"+temp1);
64.             }
65.         } catch (IOException e) {
66.             e.printStackTrace();
67.         }
68.
69.         pw1.println("codesg ends");
70.         pw1.println("end");
71.         pw1.close();
72.
73.         return path+"/HuiBian.txt";
74.     }
```

其运行效果图如图 1-4 所示。

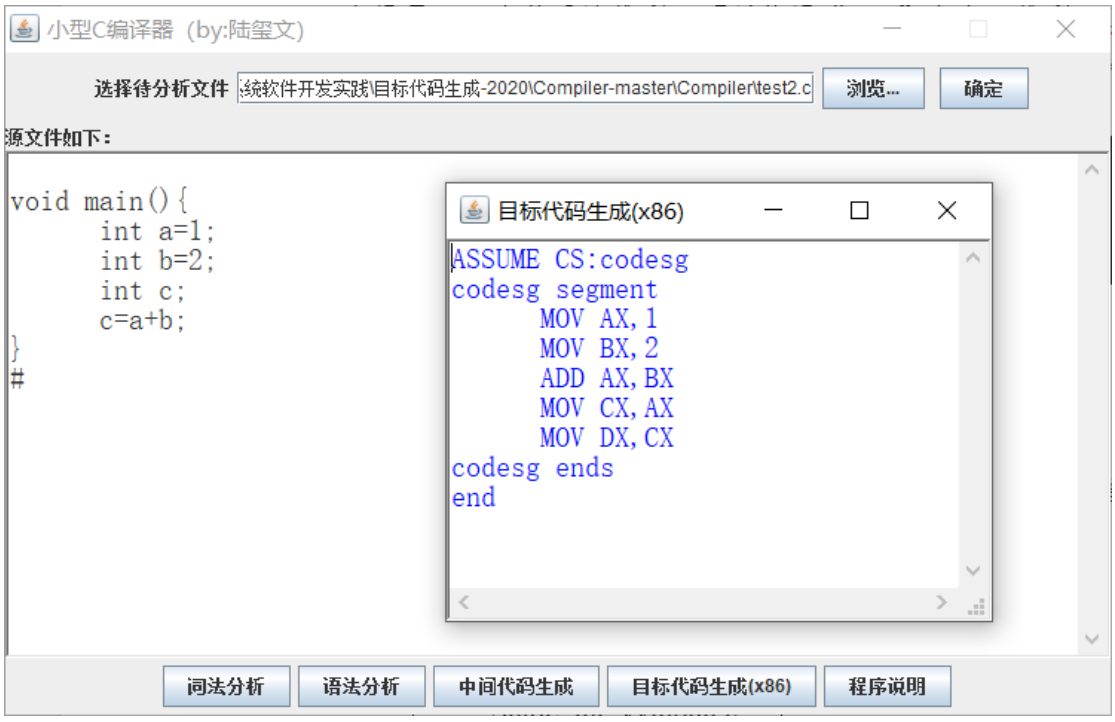

图 1-4　汇编目标代码

1.4.3 模拟器验证

将所生成的 x86 汇编代码输入 emu8086，最终运行结果如图 1-5 所示，可以看到寄存器中很好地显示了合理的运算逻辑以及结果。

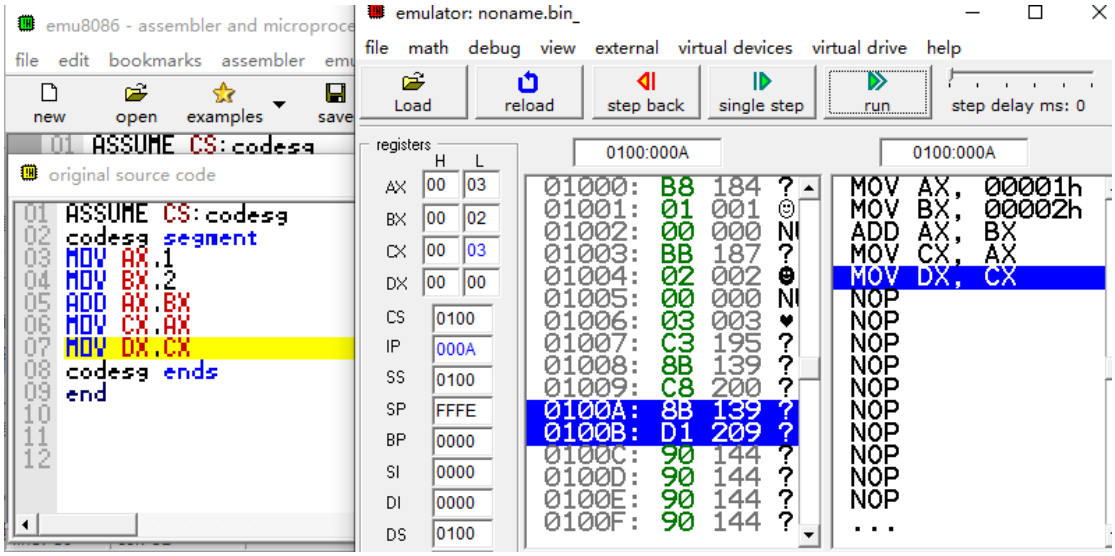

图 1-5　汇编代码模拟运行示意图

1.5 任务 3：错误处理

在实验的编译器中，实现的错误主要包含词法错误、语法错误。为了有相对统一的错误提示，编写统一错误类 error 类，如代码 1-5 所示。

代码 1-5 error

```
1.  public class Error {
2.      int id ;//错误序号;
3.      String info;//错误信息;
4.      int line ;//错误所在行
5.      Word word;//错误的单词
6.      public Error(){}
7.
8.      public Error(int id,String info,int line,Word word){
9.          this.id=id;
10.         this.info=info;
11.         this.line=line;
12.         this.word=word;
13.     }
14. }
```

1.5.1 词法分析错误

词法分析的出错提示，主要依靠自动机的原理，当所读入的字符不符合任何有效的输入时，便提示错误。出错处理部分代码如代码 1-6 所示，图 1-6 展示了示意图。

代码 1-6 词法分析部分出错处理代码

```
1.  Error error;
2.          // boolean flag=false;
3.          char temp;
4.          while (index < length) {
5.              temp = str.charAt(index);
6.              if (!noteFlag) {
7.                  if (isLetter(temp) || temp == '_') {// 判断是不是标志符
8.                      beginIndex = index;
9.                      index++;
10.                     // temp=str.charAt(index);
11.                     while ((index < length)
12.                             &&
    (!Word.isBoundarySign(str.substring(index,
13.                                     index + 1)))
14.                             && (!Word.isOperator(str
15.                                     .substring(index, index + 1)))
16.                             && (str.charAt(index) != ' ')
17.                             && (str.charAt(index) != '\t')
18.                             && (str.charAt(index) != '\r')
19.                             && (str.charAt(index) != '\n')) {
20.                         index++;
```

```
21.                         // temp=str.charAt(index);
22.                     }
23.                     endIndex = index;
24.                     word = new Word();
25.                     wordCount++;
26.                     word.id = wordCount;
27.                     word.line = line;
28.                     word.value = str.substring(beginIndex, endIndex);
29.                     if (Word.isKey(word.value)) {
30.                         word.type = Word.KEY;
31.                     } else if (isID(word.value)) {
32.                         word.type = Word.IDENTIFIER;
33.                     } else {
34.                         word.type = Word.UNIDEF;
35.                         word.flag = false;
36.                         errorCount++;
37.                         error = new Error(errorCount, "非法标识符",
    word.line, word);
38.                         errorList.add(error);
39.                         lexErrorFlag = true;
40.                     }
41.                     index--;
42. }
```

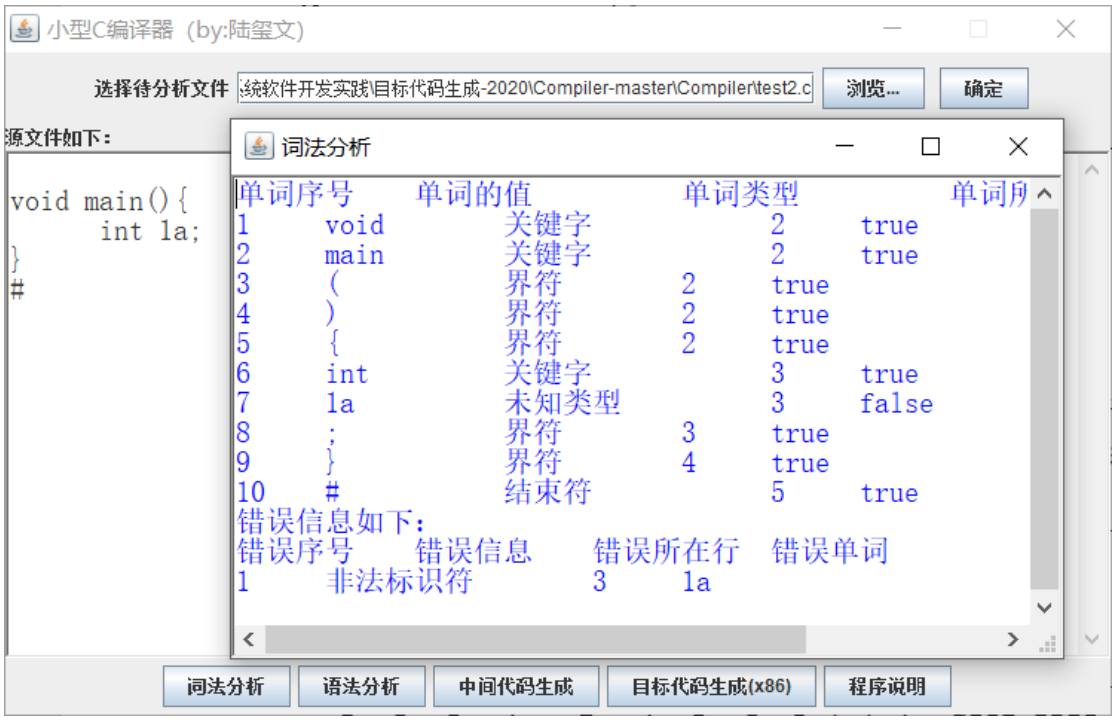

图 1-6　词法分析错误显示

1.5.2 语法分析错误

语法分析部分采用了 LL(1)的分析方法，也是主要出现错误最多的部分，凡是不符合所定义语法规则的语句串，皆会提示错误，如图 1-7 所示。

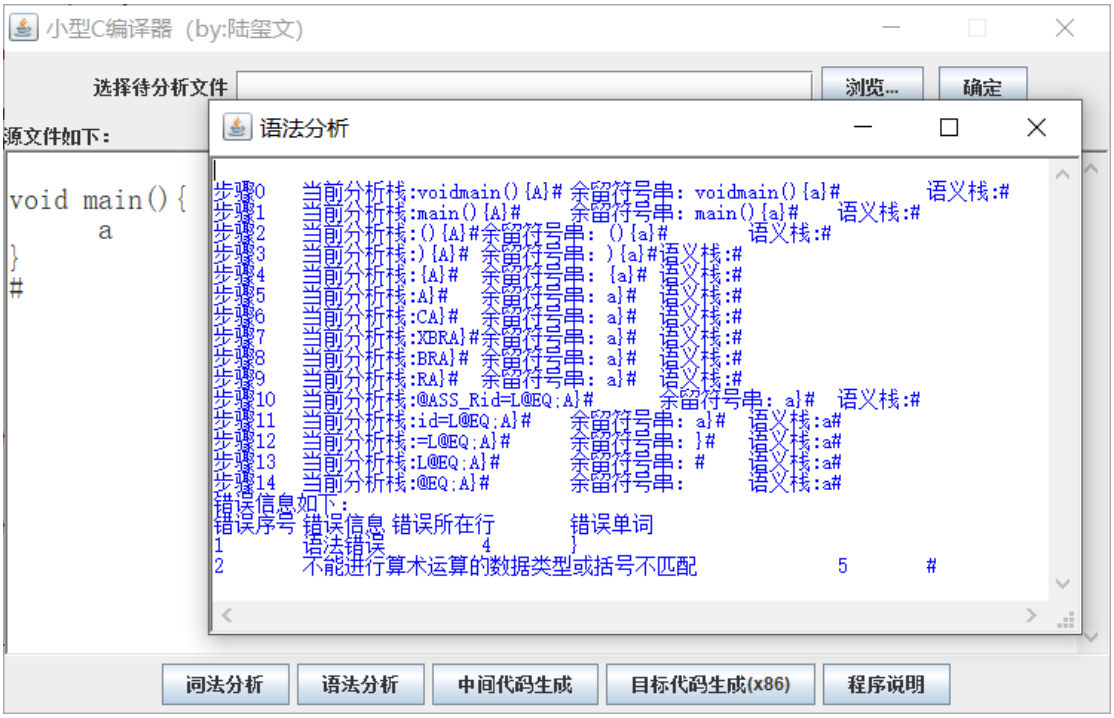

图 1-7 二叉树型

语法分析中，部分相关错误输出代码如代码 1-7 所示。

代码 1-7 main.c

```
1.  private void termOP(String term){
2.      if(firstWord.type.equals(Word.INT_CONST)||firstWord.type.equals(Wor
    d.CHAR_CONST)||
3.              term.equals(firstWord.value)||
4.              (term.equals("id")&&firstWord.type.equals(Word.IDENTIFIER)
5.                      )){
6.          analyseStack.remove(0);
7.          wordList.remove(0);
8.      }
9.          else{
10.         errorCount++;
11.         analyseStack.remove(0);
12.         wordList.remove(0);
13.         error=new Error(errorCount,"语法错误",firstWord.line,firstWord);
14.         errorList.add(error);
15.         graErrorFlag=true;
16.     }
17. }
```

1.6 任务 4：生成可运行程序

为了实现自动生成可运行程序，需要考虑使用链接器对目标代码进行处理，与其他相关动态链接库一起协作，最终得到可以直接在目标机上运行的程序。这一部分有待进一步探索学习。

1.7 实验总结

1.7.1 遇到的难题

详实直接的资料较为匮乏，网上不乏精美完整的小型编译器，然而阅读源代码的过程十分痛苦。此外理论书籍中对于编译器的构造实验部分讲解较为浅显，整体的实验比较磕磕绊绊。

1.7.2 实验收获

这一次实验进一步熟悉巩固了《编译原理》的课程知识，对于一个完整的编译器构造有了更加进一步的过程上的熟悉。同时，在开源代码的基础上，完成了针对 x86 目标机的目标代码生成修改，实现了可视化界面展示词法分析、语法分析、四元式生成、目标代码生成的完整编译步骤。

受限于时间和能力，整个实验中也有许多不足之处，最终的编译器泛化性能仍然十分欠缺，链接部分的知识同样有待进一步加强学习。