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](#_Toc41329323)

[1.1 实验目的 1](#_Toc41329324)

[1.2 实验内容 1](#_Toc41329325)

[1.3 任务1：X86体系学习 1](#_Toc41329326)

[1.3.1 Emu8086模拟器使用 1](#_Toc41329327)

[1.3.2 体系结构相关知识 2](#_Toc41329328)

[1.4 任务2：汇编程序生成 2](#_Toc41329329)

[1.4.1 四元式生成 2](#_Toc41329330)

[1.4.2 汇编语句生成 8](#_Toc41329331)

[1.4.3 模拟器验证 11](#_Toc41329332)

[1.5 任务3：错误处理 11](#_Toc41329333)

[1.5.1 词法分析错误 12](#_Toc41329334)

[1.5.2 语法分析错误 14](#_Toc41329335)

[1.6 任务4：生成可运行程序 15](#_Toc41329336)

[1.7 实验总结 15](#_Toc41329337)

[1.7.1 遇到的难题 15](#_Toc41329338)

[1.7.2 实验收获 15](#_Toc41329339)

# 目标代码生成

## 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。

## 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。

## 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。

### Emu8086模拟器使用

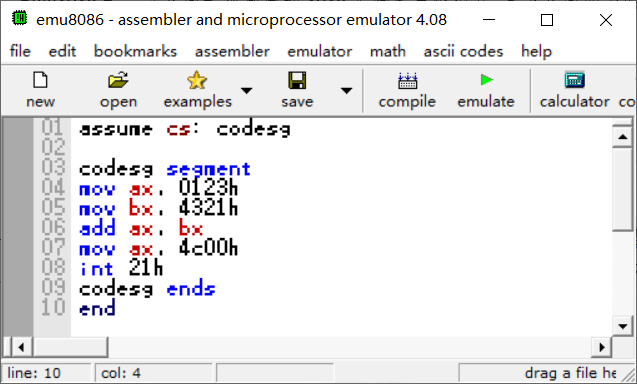


图 ‑1 Emu8086使用界面

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。

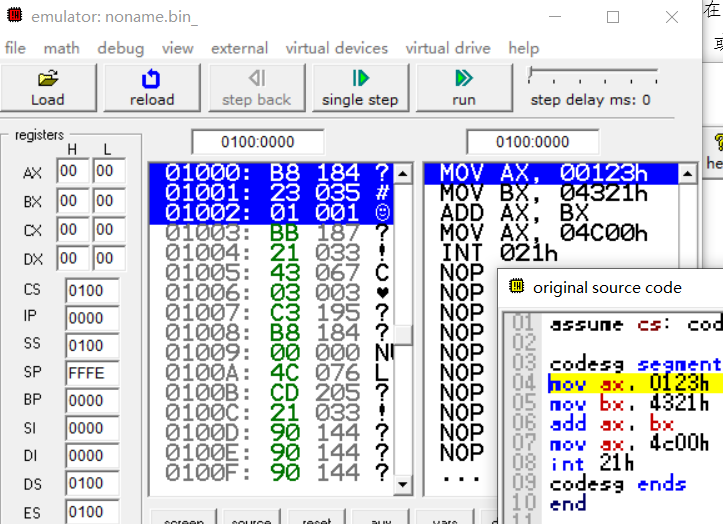


图 ‑2 识别文法G[S1]特定项目集S

### 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。

## 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> 。

### 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()）

|  |  |
| --- | --- |
|  | **private** **void** actionSignOP(){ |
|  | **if**(top.name.equals("@ADD\_SUB")){ |
|  | **if**(OP!=**null**&&(OP.equals("+")||OP.equals("-"))){ |
|  | ARG2=semanticStack.pop(); |
|  | ARG1=semanticStack.pop(); |
|  | RES=newTemp(); |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,ARG2,RES); |
|  | fourElemList.add(fourElem); |
|  | L.value=RES; |
|  | semanticStack.push(L.value); |
|  | OP=**null**; |
|  | } |
|  | analyseStack.remove(0); |
|  |  |
|  | }**else** **if**(top.name.equals("@ADD")){ |
|  | OP="+"; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@SUB")){ |
|  | OP="-"; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@DIV\_MUL")){ |
|  | **if**(OP!=**null**&&(OP.equals("\*")||OP.equals("/"))){ |
|  | ARG2=semanticStack.pop(); |
|  | ARG1=semanticStack.pop(); |
|  | RES=newTemp(); |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,ARG2,RES); |
|  | fourElemList.add(fourElem); |
|  | T.value=RES; |
|  | semanticStack.push(T.value); |
|  | OP=**null**; |
|  | } |
|  | analyseStack.remove(0); |
|  | } |
|  | **else** **if**(top.name.equals("@DIV")){ |
|  | OP="/"; |
|  | analyseStack.remove(0); |
|  | } |
|  | **else** **if**(top.name.equals("@MUL")){ |
|  | OP="\*"; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@TRAN\_LF")){ |
|  | F.value=L.value; |
|  | //semanticStack.push(F.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@ASS\_F")){ |
|  | F.value=firstWord.value; |
|  | semanticStack.push(F.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@ASS\_R")){ |
|  | R.value=firstWord.value; |
|  | semanticStack.push(R.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@ASS\_Q")){ |
|  | Q.value=firstWord.value; |
|  | semanticStack.push(Q.value); |
|  | analyseStack.remove(0); |
|  | } |
|  | **else** **if**(top.name.equals("@ASS\_U")){ |
|  | U.value=firstWord.value; |
|  | semanticStack.push(U.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@SINGLE")){ |
|  | **if**(for\_op.peek()!=**null**){ |
|  | ARG1=semanticStack.pop(); |
|  | RES=ARG1; |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,for\_op.pop(),ARG1,"/",RES); |
|  | fourElemList.add(fourElem); |
|  | } |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@SINGLE\_OP")){ |
|  | for\_op.push(firstWord.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@EQ")){ |
|  | OP="="; |
|  | ARG1=semanticStack.pop(); |
|  | RES=semanticStack.pop();; |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,"/",RES); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | } |
|  | **else** **if**(top.name.equals("@EQ\_U'")){ |
|  | OP="="; |
|  | ARG1=semanticStack.pop(); |
|  | RES=semanticStack.pop();; |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,"/",RES); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@COMPARE")){ |
|  | ARG2=semanticStack.pop(); |
|  | OP=semanticStack.pop(); |
|  | ARG1=semanticStack.pop(); |
|  | RES=newTemp(); |
|  | //fourElemCount++; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,ARG2,RES); |
|  | fourElemList.add(fourElem); |
|  | G.value=RES; |
|  | semanticStack.push(G.value); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@COMPARE\_OP")){ |
|  | D.value=firstWord.value; |
|  | semanticStack.push(D.value); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@IF\_FJ")){ |
|  | OP="FJ"; |
|  | ARG1=semanticStack.pop(); |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,"/",RES); |
|  | if\_fj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@IF\_BACKPATCH\_FJ")){ |
|  | backpatch(if\_fj.pop(), fourElemCount+2); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@IF\_RJ")){ |
|  | OP="RJ"; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,"/","/","/"); |
|  | if\_rj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@IF\_BACKPATCH\_RJ")){ |
|  | backpatch(if\_rj.pop(), fourElemCount+1); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@WHILE\_FJ")){ |
|  | OP="FJ"; |
|  | ARG1=semanticStack.pop(); |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,"/","/"); |
|  | while\_fj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@WHILE\_RJ")){ |
|  | OP="RJ"; |
|  | RES=(while\_fj.peek()-1)+""; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,"/","/",RES); |
|  | for\_rj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@WHILE\_BACKPATCH\_FJ")){ |
|  | backpatch(while\_fj.pop(), fourElemCount+1); |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@FOR\_FJ")){ |
|  | OP="FJ"; |
|  | ARG1=semanticStack.pop(); |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,ARG1,"/","/"); |
|  | for\_fj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@FOR\_RJ")){ |
|  | OP="RJ"; |
|  | RES=(for\_fj.peek()-1)+""; |
|  | FourElement fourElem=**new** FourElement(++fourElemCount,OP,"/","/",RES); |
|  | for\_rj.push(fourElemCount); |
|  | fourElemList.add(fourElem); |
|  | OP=**null**; |
|  | analyseStack.remove(0); |
|  | }**else** **if**(top.name.equals("@FOR\_BACKPATCH\_FJ")){ |
|  | backpatch(for\_fj.pop(), fourElemCount+1); |
|  | analyseStack.remove(0); |
|  | } |
|  | } |

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

|  |  |
| --- | --- |
|  | **void** main**(){** |
|  | **int** a**=**1**;** |
|  | **int** b**=**2**;** |
|  | **int** c**;** |
|  | c**=**a**+**b**;** |
|  | **}** |
|  | # |

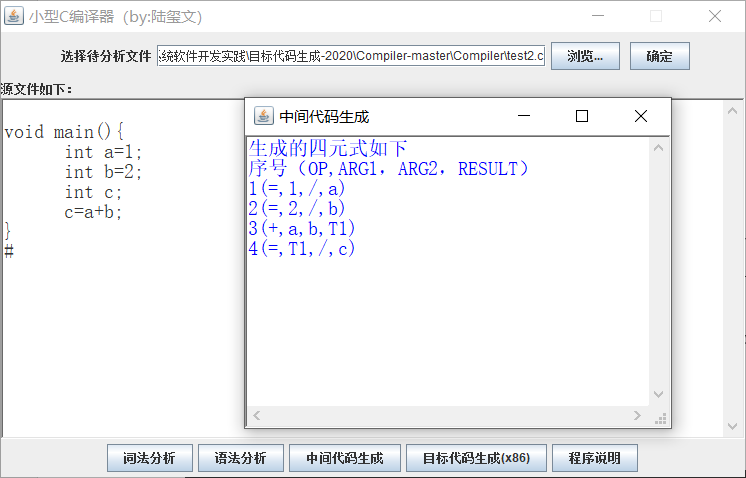


图 ‑3 四元式生成图

### 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

|  |  |
| --- | --- |
|  | // 寄存器分配，用于存储变量与寄存器对应编号 |
|  | // 寄存器分配还需好好设计，暂且使用数值自增 |
|  | **private** Map<String,String> registerVariateMap = **new** HashMap<String, String>(); |
|  | **private** **int** registerNum=65; |
|  | **public** String registerAllocation(String vatiate) { |
|  | **if**(*isNumericzidai*(vatiate)) { |
|  | **return** vatiate; |
|  | } |
|  | **if**(registerVariateMap.get(vatiate)!=**null**) { |
|  |  |
|  | } **else**{ |
|  | StringBuffer sbu = **new** StringBuffer(); |
|  | sbu.append((**char**)registerNum++); |
|  | sbu.append('X'); |
|  | System.***out***.print("vatiate: "+vatiate); |
|  | System.***out***.println(", "+sbu.toString()); |
|  | registerVariateMap.put(vatiate, sbu.toString()); |
|  | } |
|  | **return** registerVariateMap.get(vatiate); |
|  | } |

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

|  |  |
| --- | --- |
|  | **public** String outputHuiBian() **throws** IOException{ |
|  |  |
|  | File file=**new** File("./output/"); |
|  | **if**(!file.exists()){ |
|  | file.mkdirs(); |
|  | file.createNewFile();//如果这个文件不存在就创建它 |
|  | } |
|  | String path=file.getAbsolutePath(); |
|  | FileOutputStream fos=**new** FileOutputStream(path+"/HuiBian.txt"); |
|  | BufferedOutputStream bos=**new** BufferedOutputStream(fos); |
|  | OutputStreamWriter osw1=**new** OutputStreamWriter(bos,"utf-8"); |
|  | PrintWriter pw1=**new** PrintWriter(osw1); |
|  | pw1.println("ASSUME CS:codesg"); |
|  | pw1.println("codesg segment"); |
|  |  |
|  | // FourElement temp; |
|  | // for(int i=0;i<fourElemList.size();i++){ |
|  | // temp=fourElemList.get(i); |
|  | // pw1.println(temp.id+"("+temp.op+","+temp.arg1+","+temp.arg2+","+temp.result+")"); |
|  | // } |
|  |  |
|  | **try** { |
|  | String str[] =readFile(*fileName*).split("\n"); |
|  | String temp1 = **null**; |
|  | **for**(**int** i = 2; i < str.length; i++) |
|  | { |
|  | String temp[] = str[i].split(","); |
|  | **if**(temp[0].charAt(temp[0].length() -1) == '='){ |
|  | String src = temp[3].substring(0,temp[3].length() - 1); |
|  | temp1 = "MOV " + registerAllocation(src) + "," + registerAllocation(temp[1]) + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '+' && temp[0].charAt(temp[0].length() -2) == '+'){ |
|  | temp1 = "INC " + registerAllocation(temp[1]) + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '+'){ |
|  | String dst = registerAllocation(temp[1]); |
|  | String src = registerAllocation(temp[2]); |
|  | String dsf = registerAllocation(temp[3].substring(0,temp[3].length() - 1)); |
|  | temp1 = "ADD " + dst + "," + src + "\n"; |
|  | temp1+= "\t"+"MOV " + dsf + "," + dst + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '-'){ |
|  | temp1 = "SUB " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '\*'){ |
|  | temp1 = "MUL " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '/'){ |
|  | temp1 = "DIV " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == 'J' && temp[0].charAt(temp[0].length() -2) == 'R'){ |
|  | temp1 = "JMP " + temp[3].substring(0,temp[3].length() - 1) + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == 'J' && temp[0].charAt(temp[0].length() -2) == 'F'){ |
|  | temp1 = "JZ " + temp[3].substring(0,temp[3].length() - 1) + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '>'){ |
|  | temp1 = "JG " + temp[3].substring(0,temp[3].length() - 1) + "\n"; |
|  | } |
|  | **else** **if**(temp[0].charAt(temp[0].length() -1) == '<'){ |
|  | temp1 = "JL " + temp[3].substring(0,temp[3].length() - 1) + "\n"; |
|  | } |
|  | pw1.print("\t"+temp1); |
|  | } |
|  | } **catch** (IOException e) { |
|  | e.printStackTrace(); |
|  | } |
|  |  |
|  | pw1.println("codesg ends"); |
|  | pw1.println("end"); |
|  | pw1.close(); |
|  |  |
|  | **return** path+"/HuiBian.txt"; |
|  | } |

其运行效果图如图 1‑4所示。

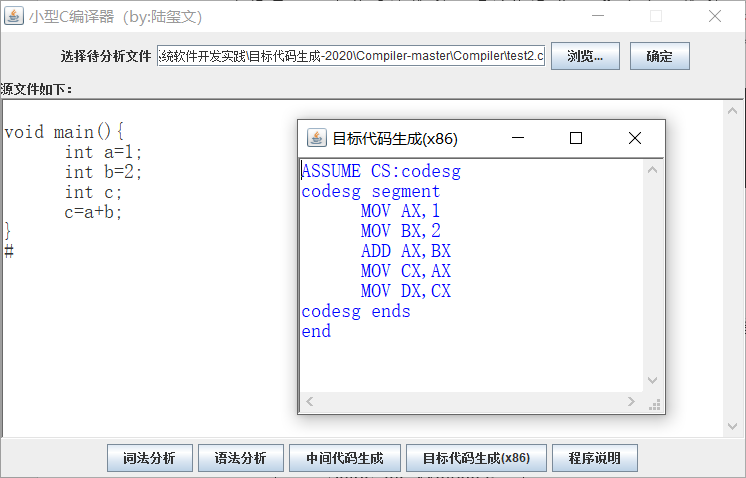


图 ‑4 汇编目标代码

### 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。

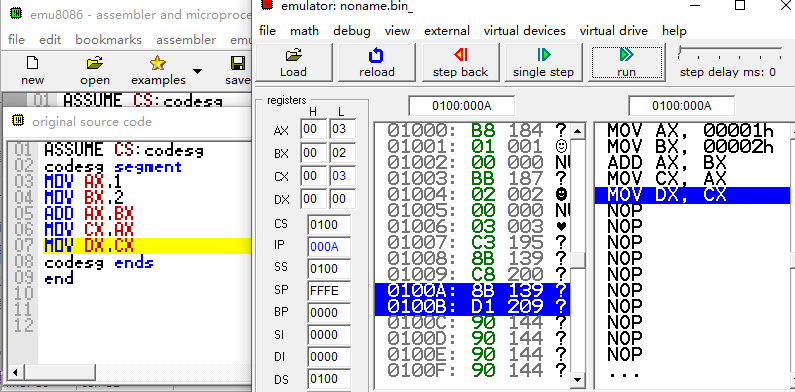


图 ‑5 汇编代码模拟运行示意图

## 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

|  |  |
| --- | --- |
|  | **public** **class** Error { |
|  | **int** id ;//错误序号； |
|  | String info;//错误信息； |
|  | **int** line ;//错误所在行 |
|  | Word word;//错误的单词 |
|  | **public** Error(){} |
|  |  |
|  | **public** Error(**int** id,String info,**int** line,Word word){ |
|  | **this**.id=id; |
|  | **this**.info=info; |
|  | **this**.line=line; |
|  | **this**.word=word; |
|  | } |
|  | } |

### 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

|  |  |
| --- | --- |
|  | Error error; |
|  | // boolean flag=false; |
|  | **char** temp; |
|  | **while** (index < length) { |
|  | temp = str.charAt(index); |
|  | **if** (!noteFlag) { |
|  | **if** (*isLetter*(temp) || temp == '\_') {// 判断是不是标志符 |
|  | beginIndex = index; |
|  | index++; |
|  | // temp=str.charAt(index); |
|  | **while** ((index < length) |
|  | && (!Word.*isBoundarySign*(str.substring(index, |
|  | index + 1))) |
|  | && (!Word.*isOperator*(str |
|  | .substring(index, index + 1))) |
|  | && (str.charAt(index) != ' ') |
|  | && (str.charAt(index) != '\t') |
|  | && (str.charAt(index) != '\r') |
|  | && (str.charAt(index) != '\n')) { |
|  | index++; |
|  | // temp=str.charAt(index); |
|  | } |
|  | endIndex = index; |
|  | word = **new** Word(); |
|  | wordCount++; |
|  | word.id = wordCount; |
|  | word.line = line; |
|  | word.value = str.substring(beginIndex, endIndex); |
|  | **if** (Word.*isKey*(word.value)) { |
|  | word.type = Word.***KEY***; |
|  | } **else** **if** (*isID*(word.value)) { |
|  | word.type = Word.***IDENTIFIER***; |
|  | } **else** { |
|  | word.type = Word.***UNIDEF***; |
|  | word.flag = **false**; |
|  | errorCount++; |
|  | error = **new** Error(errorCount, "非法标识符", word.line, word); |
|  | errorList.add(error); |
|  | lexErrorFlag = **true**; |
|  | } |
|  | index--; |
|  | } |

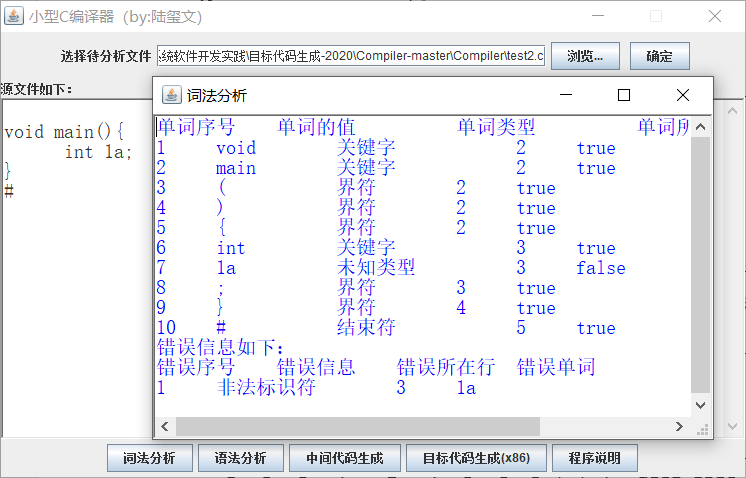


图 ‑6 词法分析错误显示

### 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 ‑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 ‑7 main.c

|  |  |
| --- | --- |
|  | **private** **void** termOP(String term){ |
|  | **if**(firstWord.type.equals(Word.***INT\_CONST***)||firstWord.type.equals(Word.***CHAR\_CONST***)|| |
|  | term.equals(firstWord.value)|| |
|  | (term.equals("id")&&firstWord.type.equals(Word.***IDENTIFIER***) |
|  | )){ |
|  | analyseStack.remove(0); |
|  | wordList.remove(0); |
|  | } |
|  | **else**{ |
|  | errorCount++; |
|  | analyseStack.remove(0); |
|  | wordList.remove(0); |
|  | error=**new** Error(errorCount,"语法错误",firstWord.line,firstWord); |
|  | errorList.add(error); |
|  | graErrorFlag=**true**; |
|  | } |
|  | } |

## 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。

## 实验总结

### 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。

### 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。