

苏州大学实验报告

院、系	计算机科学与技术学院	年级专业	21 计科	姓名	赵鹏	学号	2127405037
课程名称	编译原理实践					成绩	
指导教师	段湘煜	同组实验者	无	实验日期	2023.12.11		

实验名称 TEST 栈式抽象机模拟器

一. 实验题目

在已经实现的将 TEST 语言从源代码转化到汇编指令的基础上, 实现能够执行 TEST 语言的汇编指令的栈式模拟器。

二. 实验原理及流程图

TEST 机的汇编指令都只涉及栈顶元素或栈顶以及次栈顶两个元素, 所以需要维护一个栈。同时, 将所有的变量都存储在内存, 用一个数组来记录每个变量的值。TEST 机的栈可以使用数组进行模拟, 指定数组最大为 MAXN, 并记录栈顶指针。

对与每次执行汇编文件, 首先将所有文件内容读入内存, 预处理每条指令, 对于每个 Label, 将该语句标号置为指向其后跟随的第一条指令即可。

在执行程序时, 对于下图的每条指令, 执行对应的操作即可。

LOAD	D	将 D 中的内容加载到操作数栈;
LOADI	常量	将常量压入操作数栈;
STO	D	将操作数栈栈顶单元内容存入 D, 且栈顶单元内容保持不变;
POP		将操作数栈栈顶出栈;
ADD		将次栈顶单元与栈顶单元内容出栈并相加, 和置于栈顶;
SUB		将次栈顶单元与栈顶单元内容出栈并相减, 差置于栈顶;
MULT		将次栈顶单元与栈顶单元内容出栈并相乘, 积置于栈顶;
DIV		将次栈顶单元与栈顶单元内容出栈并相除, 商置于栈顶;
BR	lab	无条件转移到 lab
BRF	lab	检查栈顶单元逻辑值并出栈, 若为假(0)则转移到 lab;
EQ		将栈顶两单元做等于比较并出栈, 并将结果真或假(1 或 0)置于栈顶;
NOTEQ		将栈顶两单元做不等于比较并出栈, 并将结果真或假(1 或 0)置于栈顶;
GT		次栈顶大于栈顶操作数并出栈, 则栈顶置 1, 否则置 0;
LES		次栈顶小于栈顶操作数并出栈, 则栈顶置 1, 否则置 0;
GE		次栈顶大于等于栈顶操作数并出栈, 则栈顶置 1, 否则置 0;
LE		次栈顶小于等于栈顶操作数并出栈, 则栈顶置 1, 否则置 0;
AND		将栈顶两单元做逻辑与运算并出栈, 并将结果真或假(1 或 0)置于栈顶;
OR		将栈顶两单元做逻辑或运算并出栈, 并将结果真或假(1 或 0)置于栈顶;
NOT		将栈顶的逻辑值取反;
IN		从标准输入设备(键盘)读入一个整型数据, 并入操作数栈;
OUT		将栈顶单元内容出栈, 并输出到标准输出设备上(显示器);
STOP		停止执行;

三. 实验步骤

(一) 定义 TEST 语言栈式抽象机

定义 TVM(TEST Virtual Machine)类,定义成员 `Array<int>stack` 用于模拟栈、`stackTop` 用于给记录栈顶位置、`Array<int>data` 用于模拟内存,存储内存中每个位置的数、`codeCounter` 用于给指令计数。定义 `Array<Code>code` 用于存储所有预处理后的指令、哈希表 `lableJump` 用于记录每个 LABEL 的跳转位置。

```
class TVM
{
public:
    TVM();
    void reset();
    int run(string path);
    int analyse();
    static const int MAXN = 1000;

private:
    int codeCounter = 0;
    Array<int> stack = Array<int>(MAXN), data = Array<int>(MAXN);
    Array<Code> code = Array<Code>(MAXN);
    unordered_map<string, int> lableJump;
    int stackTop = 0;
};
```

其中 `Code` 类型用于存储每条指令,主要用于将指令按行读入后分割为多个部分,使用 `vector<string>`进行存储,便于在模拟执行程序的过程中使用。

`Code` 的结构如下:

```
class Code
{
public:
    vector<string> code;
    bool isLable;
    Code(string s);
    Code();
};
```

TVM 通过 `run(string path)`和 `analyse()`函数执行 TEST 汇编程序, `run` 函数传入汇编程序的文件地址,读取汇编程序,预处理出 `code` 数组和哈希表 `lableJump`。`analyse()`函数用于模拟执行预处理后的汇编程序。

(二) 实现 run 函数

`run()` 传入汇编程序的文件地址，打开文件按行读入汇编程序，对于每行创建 `Code` 对象，如果 `isLabel` 为真则记录 `labelJump` 为当前的 `codeCounter` 并继续读入下一条语句，否则将这一 `Code` 对象放入 TVM 的 `code` 数组中。不断执行这一过程，直至文件结束。

随后调用 `analyse` 模拟执行 `code` 数组中的指令，在执行结束后调用 `reset` 函数清空 `labelJump` 表和将 `stackTop` 指针置 0。

`analyse` 函数用于模拟执行汇编代码，定义程序计数器 `PC` 用于指示当前将执行的指令的位置，初始时刻为 0。在指令没有执行完，即 `PC < codeCounter` 时不断取下一条指令。随后读取每条指令，根据 `code[0]` 即可判断指令类型，执行该类型的相应操作即可。

(三) 增加虚拟机程序的健壮性

考虑到虚拟机程序在执行汇编代码过程中可能因为某些原因导致访问数组时下标越界，若使用普通数组进行存储，在下标越界时可能出现无法预知的错误。为了提高 TEST 虚拟机的健壮性，自定义了 `Array` 模板类。其结构如下：

```
template <typename T>
class Array
{
    T *mArray;
    int mLen;

public:
    Array(int n);
    ~Array();
    T &operator[](int index);
};
```

`Array` 模板类的每个对象记录了其数组长度，使用指针动态申请数组长度的空间，并通过重载 `[]` 符号保证安全性，在使用 `[]` 访问数组中的元素时，首先会检查下标的合法性，若 `idx < 0` 或 `idx >= mLen` 则说明越界，直接抛出运行错误的异常。这样便避免了数组越界访问的异常，提升了程序的健壮性。

四. 实验结果及分析

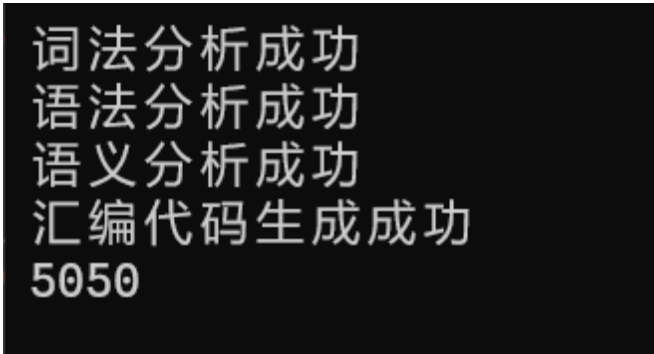
测试一(计算 1 到 100 的和):

输入:

```
{
    int ans;
    int i;
    for(i=0;i<=100;i=i+1)
    {
```

```
ans=ans+i;
}
write ans;
}
```

输出:



词法分析成功
语法分析成功
语义分析成功
汇编代码生成成功
5050

测试二(斐波那契数列前 n 项)

输入:

```
{
    int n;
    int a;
    int b;
    int c;
    int i;
    read n;
    a = 1;
    b = 1;
    c = 0;
    write a;
    if(n >= 2)
        write b;

    for(i = 3; i <= n; i = i + 1){
        c = a + b;
        a = b;
        b = c;
        write c;
    }
}
```

输出:

```
词法分析成功
语法分析成功
语义分析成功
汇编代码生成成功
```

```
20
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

五. 实验总结

通过本次实验，我实现了 TEST 语言虚拟机的程序，能够正确地执行 TEST 语言的汇编代码，在测试中正确的计算了 1-100 以内数的和、计算斐波那契数列等在 TEST 语言中相对复杂的操作。可惜的是，TEST 语言并不支持数组、函数等功能，只能够实现一些相对简单的功能。尽管 TEST 语言非常简单，但回顾本学期的一系列实验，从 TEST 语言的词法分析做到生成汇编代码并执行，在这个过程中我不仅不断加深了对编译原理理论知识的理解，也对词法分析、语法分析、语义分析等过程有了新的认识，我的代码能力和面向对象的设计能力也都有了一定的提高，相比于以往其他的课程的实验课，本学期编译原理的实验是我迄今做的最有趣、有意义的实验。

六. 代码

本次实验涉及到的文件较多，已将所有文件打包添加到附件。