CS323 Project Phrase3

小组成员

12012438 杨可芸， 12011136 陈茜， 12010339 王思懿

运行方式

```
make clean
make splc
./test_phase3.sh
```

即可生成test和test-ex下的所有.spl对应的.ir文件

设计思路

自顶向下解析：基本上对于每一种文法符号都实现了translate_xxx()方法进行解析，并且每一个tree node都有一个ir字段，以`vector<string>`的形式存放中间代码以CompSt举例：

```
CompSt -> LC DefList StmtList RC
```

我们会获取CompSt第二个子节点作为DefList, 第三个子节点作为StmtList, 分别调用translate_DefList(DefList), translate_StmtList(StmtList), 最后将CompSt节点的中间代码赋值为DefList的中间代码+StmtList的中间代码。

优化

translate_Exp 函数的定义：

```
void translate_Exp(Node* node, string& place);
```

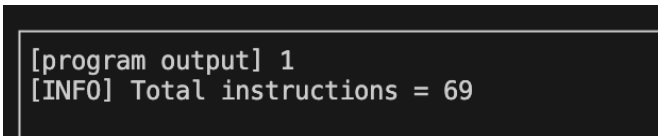
我们将place参数用**引用**的方式传递，便于在Exp -> ID 时将place赋值为变量名，在Exp -> INT时将place赋值为具体的数字，如下图所示：

```
if (node->nodes_num == 1 &&
    node->children[0]->nodetype == Int) {
    // Exp -> INT
    place = "#" + string(node->children[0]->name.c_str());
} else if (node->nodes_num == 1 &&
           node->children[0]->nodetype == Id) {
    // Exp -> ID
    place = string(node->children[0]->char_value);
```

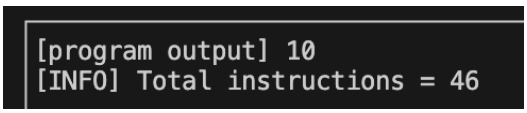
这样省去了每次将ID和INT赋值到一个中间变量上的指令。

运行结果

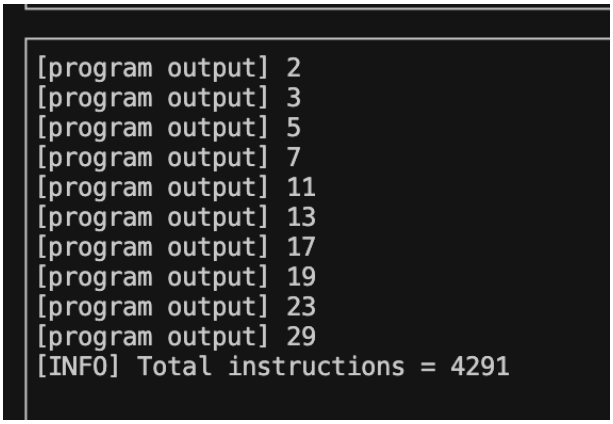
dist/irsim test/test_3_r01.ir -i 121



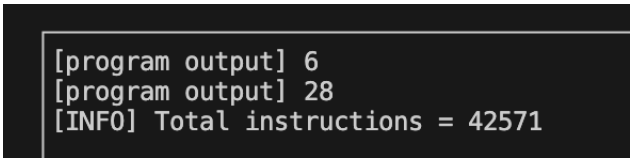
dist/irsim test/test_3_r02.ir -i 5,3



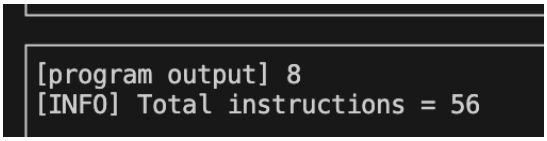
dist/irsim test/test_3_r03.ir



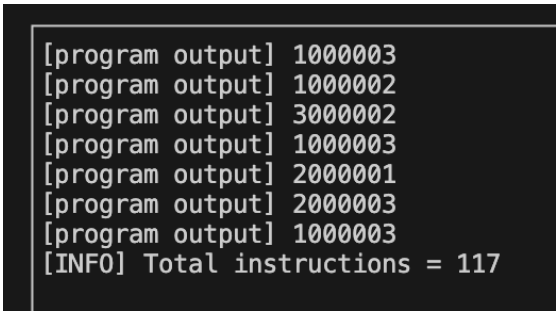
dist/irsim test/test_3_r04.ir



dist/irsim test/test_3_r05.ir -i 6



dist/irsim test/test_3_r06.ir



Optional Features

Struct

引入了结构体，并且可以作为函数的入参，通过引用传递。

```
unordered_map<string, unordered_map<string, int>> class_field_map;
unordered_map<string, string> obj_class_map;
unordered_map<string, string> obj_class_funcmap;
```

class_field_map的key是类名，value是一个map，key是字段名，value代表该字段是第几个字段。
obj_class_map的key是对象名，value是类名，该map用来存储在main方法中创建的对象。
obj_class_funcmap的key是对象名，value是类名，该map用来存储在函数中调用的对象。

调用函数时，如果发现Args中有的ID在obj_class_map中，则代表该变量是结构体，传递参数时的中间代码为"ARG &id"

解析Exp -> Exp DOT ID时，用`class_field_map[obj_class_map[obj_name]][field_name]`来获取该字段是第几个字段，作为offset，将`offset*4`作为地址的偏移量。如果obj_name在obj_class_map中，代表他是一个结构体类型，`address := &obj_name + #4 * offset`；如果obj_name在obj_class_funcmap中，代表他是一个地址的值，`address := obj_name + #4 * offset`

Test：

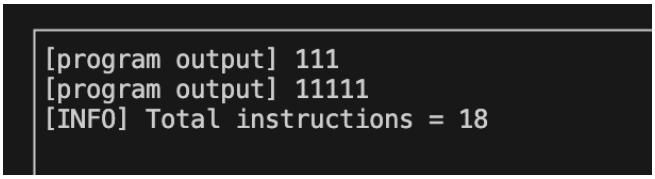
dist/irsim test-ex/test_a.ir

```
struct myStruct
{
    int field1;
    int field2;
};

int test(struct myStruct a){
    a.field1 = 11111;
    return 0;
}

int main(){
    struct myStruct c1,c2;
    c1.field1 = 111;
    write(c1.field1);
    test(c1);
    write(c1.field1);
    return 0;
}
```

Result:



Array

引入了一维数组，并且可以作为函数的入参，通过引用传递。

```
vector<string> arr_funclist;
vector<string> arr_list;
```

arr_list 记录在main函数里创建的每个array的id
arr_funclist 记录在自定义函数里调用的array的id

调用函数时，如果发现Args中有的ID在arr_list中，则代表该变量是数组，传递参数时的中间代码为 "ARG &id"

解析Exp -> Exp1 LB Exp2 RB时，用`translate(Exp2,offset)`来获取下标，将`offset * 4`作为偏移量。如果Exp1 的ID在arr_list中，代表他是一个数组类型，`address := &id + #4 * offset`; 如果obj_name在arr_funclist中，代表他是一个地址的值，`address := id + #4 * offset`

Test:

dist/irsim test-ex/test_b.ir

```
int test(int a[10]){
    a[1] = 111;
    return 0;
}

int main(){
    int arr[10];
    arr[1] = 1;
    write(arr[1]);
    test(arr);
    write(arr[1]);
    return 0;
}
```

Result: