编译原理 lab3 实验报告

程序编译相关

本次实验的完成没有使用模版

实验环境：wsl2，其他工具均按实验指导安装，无需额外的库

文件结构：

```
compiler-lab1-lab2-g14.zip
 ├── src/
 ├── Makefile
 └── report
```

在`Makefile`所在文件夹执行`make`，生成可执行文件`compiler`

本次实验的完成没有使用模版

实现功能和实现方法

实现功能

在lab1和lab2的基础上，把语法树转换成中间代码。

实现翻译函数 `translate_expr` 和 `translate_stmt` 的功能。

实现符号表 `sym_table` 管理，但使用的是比较离散的处理。

实现在结构化控制流 If 和 While 中条件表达式短路语义的翻译。

实现方法

符号表

符号表与lab2比较相似，使用`unordered_map`实现，但是没有使用lab2中复杂的定义和操作，以更简单方便的方式进行使用，如不再使用复杂的pType类。这部分和lab2相似，此处不在赘述。

中间代码储存

使用一个全局变量`ircode`进行储存，翻译过程中不断`push_back()`完成中间代码的生成

translate 函数

还是遍历语法树，在需要转换的时候，将对应的语句转换为中间代码指令。关键的两个函数如下：

```
string tranExp(pNode node);
void tranStmt(pNode node);
```

分别实现对expression和statement的翻译，思路上就是遍历语法树，在需要转换的地方进行翻译和pushback。

举例说明：

当遍历语法树到这里时：

```
UnaryExp:PrimaryExp
    | ID LP FuncRParams RP
    | ID LP RP
    | UnaryOp UnaryExp
    ;
```

对应的第三条，基本与无参数函数调用相对应，利用：

```
string func_name = node->child->value;
string ss = "let %"+to_string(regnum++)+": i32 = call @"+func_name;
ircode.push_back(ss)
```

> 其中regnum是一个虚拟寄存器的计数int

将之翻译成中间代码并pushback

控制流实现

嵌套处理：

```
int if_scope = 0;
int while_scope = 0;
```

定义两个全局变量，分别控制嵌套的if和while的控制流，每次遇到if或while，在设置标签时自增，以区分不同的跳转标签，举例如下：

```
  if(a == 5){
    if (b == 10)
      a = 25;
    else
      a = a + 15;
  }
```

对应中间代码：

```
%if_then_1:
    let %5: i32 = load %b: i32*
    let %6: i32 = eq %5: i32, 10
    br %6: i32, label %if_then_2, label %if_else_2
%if_then_2:
    let %7: () = store 25,%a: i32*
    jmp label %if_end_2
%if_else_2:
    let %8: i32 = load %a: i32*
    let %9: i32 = add %8: i32, 15
    let %10: () = store %9: i32,%a: i32*
    jmp label %if_end_2
%if_end_2:
    jmp label %if_end_1
%if_end_1:
    let %11: i32 = load %a: i32*
    let %12: () = store %11: i32, %ret_val.addr: i32*
    jmp label %exit
```

短路处理：

主要需要在翻译到如下节点时处理：

```
LAndExp:EqExp
    | LAndExp AND EqExp
    ;
LOrExp: LAndExp
    | LOrExp OR LAndExp
    ;
```

为此专门定义一个全局变量`shortscope`来记录当前的scope。

主要的思路在于：

- 如果是and，需要先计算前半部分去判断结果，如果是false之间跳转到false板块

- 如果是or，也需要先计算前半部分去判断结果，如果是true之间跳转到true板块

通过：short_val，short.rhs，short.end配合shortscope处理（参考的学长给的rust compiler的处理方法）

全局变量

定义比较简单，之间翻译即可，主要说一下定义+赋值的部分。思路仍参考rust compiler，在main函数的最开始对全局变量进行赋值操作。举例如下：

```
//全局变量定义
int a;
int b[2];
int c[4][3];
int i = 2024;
```

中间代码：

```
...
@a: region i32, 1
@b: region i32, 2
@c: region i32, 12
@i: region i32, 1
fn @main() -> i32 {
%entry:
let %ret_val.addr: i32* = alloca i32, 1
let %1:() = store 2024,@i: i32*
...
```

测试结果

```
(base) linbojunzi@LAPTOP-0F36G23P:~/accipit/tests$ python3 test.py ./compiler lab3
Running lab3 test...
./lab3/unary_op.sy                PASSED
./lab3/while_test1.sy             PASSED
./lab3/array_parameter.sy         PASSED
./lab3/func_call.sy               PASSED
./lab3/div.sy                     PASSED
./lab3/if_complex_expr.sy         PASSED
./lab3/if_test2.sy                PASSED
./lab3/global_test1.sy            PASSED
./lab3/combinator.sy              PASSED
./lab3/rem.sy                     PASSED
./lab3/add.sy                     PASSED
./lab3/if_test1.sy                PASSED
./lab3/mul.sy                     PASSED
./lab3/binary_search.sy           PASSED
./lab3/sort_array.sy              PASSED
./lab3/array_hard.sy              PASSED
./lab3/array2.sy                  PASSED
./lab3/while_if_test1.sy          PASSED
./lab3/array1.sy                  PASSED
./lab3/op_priority1.sy            PASSED
./lab3/while_if.sy                PASSED
./lab3/op_priority3.sy            PASSED
./lab3/sudoku.sy                  PASSED
./lab3/two_dimension_array.sy     PASSED
./lab3/short_circuit2.sy          PASSED
./lab3/short_circuit1.sy          PASSED
./lab3/quick_sort.sy              PASSED
./lab3/if_test3.sy                PASSED
./lab3/op_priority2.sy            PASSED
./lab3/while_if_test2.sy          PASSED
./lab3/merge_sort.sy              PASSED
./lab3/factorial.sy               PASSED
./lab3/stmt_expr.sy               PASSED

All tests passed!
2024-05-18 14:26:08
```

心得

本次实验非常复杂，需要考虑的细节很多，后续debug的时候一点点向上加东西，经常会发现之前写的东西，没有考虑到后续的某些操作，需要推倒重写（可能是没有用模板，自己的定义一开始不够严谨导致的）。修bug的过程也非常漫长，往往一个复杂的test需要2-3h才能找到+修好，工程量还是很大的。