CACT- PR003 目标代码生成

任务说明

在之前的实验中，我们已经完成了对`.cact`源程序的各种检查。在本次实验中，我们需要继续完成编译器后端的内容，包括从`AST`生成中间代码和从中间代码生成`RISC-V`汇编。

生成中间代码时我们需要考虑的问题有：

1. 中间代码的表示形式；
2. 如何处理`if-else`语句，`while`语句，变量声明语句；
3. 如何处理函数定义语句；
4. 如何传递参数，调用函数。

生成`RISC-V`汇编时我们需要考虑的问题有：

1. 注意寄存器约定和调用约定；
2. 浮点数和整数需要使用不同的汇编指令。

成员组成

第7组成员：

陈飞羽 2018K8009929031

彭思叡 2018K8009908040

袁欣怡 2018K8009929021

实验设计

设计思路

1. 中间代码设计

   我们设计了IRcode这个数据结构来表示中间代码。一个struct IRcode可以表示一条三地址代码。

   class IR为本次实验定义的中间代码类。该类的主要作用为：存储中间代码，记录临时变量的数量和label的数量，记录true_list和false_list。该类维护的主要变量如下：

   - 三地址代码IR_codes;
   - 临时变量的数量tmp_var_cnt;
   - label的数量label_cnt;
   - 临时变量的名字共用的开头tmp_var_head;
   - label名字共用的开头label_head;
   - 条件分支语句的false_list: b_false_list;
   - 条件分支语句的true_list: b_true_list。

   提供的主要方法如下：

   - add_IRC：添加一条IRcode指令;
   - print_IRC：打印一条IRcode指令;
   - get_label：获得一个新label的名字;
   - gen_temp：获得一个新临时变量的名字;
   - gen_temp_array：获得一个新临时数组变量的名字。

2. 汇编代码生成

   得到IRcode后，设计函数RISCV_gen，该函数的功能是将IRcode翻译为RISC-V指令。其中需要注意，操作数是否可能为浮点数，若为float或double类型，需要使用对应的浮点指令。

   在程序开始和结束时，需要注意栈的管理和返回值的设置。

实验实现

中间代码设计

我们为中间代码设计的数据结构为struct IRcode，涵盖了设计思路中提到的内容，具体代码实现为（src/IRcode.h）：

```
struct IRcode{
  std::string result; // 操作结果
  int opcode;          // 操作类型
  std::string arg1;   // 参数1
  std::string arg2;   // 参数2
  int cls;                 // 结果类型: int, float, double, bool
};
```

其中opcode是本条三地址代码的操作类型，一共定义了32种（src/IRcode.h）：

```
enum allowed_opcode {
  //e.g. [result="r",opcode"...",arg1="a1",arg2="a2",cls=INT]
  //--> r = a1 ... a2, and all are int
  OP_ADD,
  OP_SUB,
  OP_MUL,
  OP_DIV,
  OP_MOD,
  OP_AND,
  OP_OR,
  OP_NOT,
  OP_GT,
  OP_GE,
  OP_LT,
  OP_LE,
  OP_EQ,
  OP_NE,
  //e.g. [result="reg",opcode="LOAD",arg1="20",arg2="regAddr",cls=INT]
  //--> reg = 20(regAddr), and reg is int
  OP_LOAD,
  //e.g. [result="reg",opcode="STORE",arg1="20",arg2="regAddr",cls=INT]
  //--> 20(regAddr) = reg, and reg is int
  OP_STORE,
  OP_MOVE,
  //e.g. [result="reg",opcode="LI",arg1="20",arg2="",cls=INT]
  //--> reg = 20, and reg is int
  OP_LI,

  OP_G_ASSIGN,//assign for global var
  //e.g. [result="arr",opcode"G-ARR-ASSIGN",arg1="2",arg2="10",cls=INT]
  //--> arr[2] = 10, and arr is array of int
  OP_G_ARR_ASSIGN,//assign for global array
  //e.g. [result="arr",opcode"G-ARR-ASSIGN",arg1="2",arg2="10",cls=INT]
  //--> arr[2] = 10, and arr is array of int
  OP_G_ARR_DECL,
  OP_ARR_DECL,
  OP_ARR_ASSIGN,
  OP_LOAD_ARR,

  //e.g. [result="name",opcode="GOTO",arg1="",arg2="",cls=0]
  //--> goto name
  OP_GOTO,
  OP_LABEL,
  OP_CALL,
  OP_RET,
  OP_PARAM,
  OP_BEQZ,
```

```
48
49    FUNC_BEGIN,
50    FUNC_END
51  };
```

构成的中间代码类为（src/IRcode.h）：

```
1   class IR{
2   public:
3     std::vector<IRcode> IR_codes;
4     int tmp_var_cnt;
5     int label_cnt;
6     std::string tmp_var_head = "_tmp_var";
7     std::string label_head = "__lable";
8     std::vector<std::string> b_false_list;
9     std::vector<std::string> b_true_list;
10
11
12    IR(){
13        tmp_var_cnt = 0;
14        label_cnt = 0;
15    }
16
17    void add_IRC(std::string result,int opcode,std::string arg1,std::string
    arg2,int cls);
18    void print_IRC();
19    std::string get_label(){
20        return label_head+std::to_string(++label_cnt);
21    }
22    std::string gen_temp(int line, int cls, SymbolTable& );
23    std::string gen_temp_array(int line, int cls,int length, SymbolTable& );
24
25  };
```

中间代码类提供的方法如下（src/IRcode.cpp）：

- void IR::add_IRC(std::string result,int opcode,std::string arg1,std::string arg2,int cls)：添加一条新的IRcode指令。

```
void IR::add_IRC(std::string result,int opcode,std::string
arg1,std::string arg2,int cls){
    IRcode tmp;
    tmp.result = result;
    tmp.opcode = opcode;
    tmp.arg1 = arg1;
    tmp.arg2 = arg2;
    tmp.cls = cls;
    this->IR_codes.push_back(tmp);
}
```

- `void IR::print_IRC()`：打印一条IRcode指令。

```
void IR::print_IRC(){
    for(auto p : IR_codes) {
        std::cout << std::right;
        std::string cls =
                        (p.cls==0)? "int":\
                        (p.cls==1)? "double":\
                        (p.cls==2)? "float":\
                        (p.cls==3)? "bool":"";
        std::string opcode_str =
                        (p.opcode==OP_LOAD)? "OP_LOAD":\
                        (p.opcode==OP_STORE)? "OP_STORE":\
                        (p.opcode==OP_MOVE)? "OP_MOVE":\
                        (p.opcode==OP_LI)? "OP_LI":\
                        (p.opcode==OP_G_ASSIGN)? "OP_G_ASSIGN":\
                        (p.opcode==OP_G_ARR_ASSIGN)?
"OP_G_ARR_ASSIGN":\
                        (p.opcode==OP_G_ARR_DECL)?
"OP_G_ARR_DECL":\
                        (p.opcode==OP_GOTO)? "OP_GOTO":\
                        (p.opcode==OP_LABEL)? "OP_LABEL":\
                        (p.opcode==OP_CALL)? "OP_CALL":\
                        (p.opcode==OP_RET)? "OP_RET":\
                        (p.opcode==OP_PARAM)? "OP_PARAM":\
                        (p.opcode==FUNC_BEGIN)? "FUNC_BEGIN":\
                        (p.opcode==FUNC_END)? "FUNC_END":\
                        (p.opcode==OP_ARR_ASSIGN)?"ARR_ASSIGN":\
                        (p.opcode==OP_LOAD_ARR)?"LOAD_ARR":\
                        (p.opcode==OP_ARR_DECL)?"ARR_DECL":\
                        (p.opcode==OP_BEQZ)?"BEQZ":\
                        (p.opcode==OP_GOTO)?"GOTO":\
                        (p.opcode==OP_ADD)?"ADD":\
                        (p.opcode==OP_LT)?"LT":\
                        "";

```

```
33          std::cout << std::setw(10) << opcode_str << "  " <<
    std::setw(14) << p.result << "  "
34              << std::setw(14) << p.arg1 << "  " << std::setw(14) <<
    p.arg2 << "  "
35              << std::setw(7) << p.cls << std::endl;
36      }
37  }
```

- `std::string IR::gen_temp(int line, int cls, SymbolTable& symbol_table)`：生成一个新的临时变量，把它加入符号表中，并返回它的名字。

```
1  std::string IR::gen_temp(int line, int cls, SymbolTable&
   symbol_table) {
2      assert(cls == CLS_INT || cls == CLS_DOUBLE || cls == CLS_FLOAT
   || cls == CLS_BOOL);
3      auto name = tmp_var_head + std::to_string(++tmp_var_cnt);
4      symbol_table.add_symbol(name, cls, TYPE_VAR, 0, line);
5      return name;
6  }
```

- `std::string IR::gen_temp_array(int line, int cls,int length, SymbolTable& symbol_table)`：生成一个新的临时数组变量，把它加入符号表中，并返回它的名字。

```
1  std::string IR::gen_temp_array(int line, int cls,int length,
   SymbolTable& symbol_table){
2      auto name = tmp_var_head + std::to_string(++tmp_var_cnt);
3      symbol_table.add_symbol(name, cls, TYPE_ARRAY, length, line);
4      return name;
5  }
```

生成目标代码

得到IRcode后，调用函数RISCV_gen生成RISC-V代码。这里我们采取的是比较简单的翻译方法，从IRcode直接逐条翻译，得到RISC-V代码。和之前的实验一样，使用auto来简化变量初始化。

```
1  void IRListener::RISCV_gen(std::string input_file_name){
2      std::string name;
3      name = ".file   \"" + input_file_name + ".cact\"";
4      riscv_codes.push_back(name);
5      riscv_codes.push_back(".option nopic");
6      riscv_codes.push_back(".attribute arch,
   \"rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0\"");
7      //RV64GC
8      riscv_codes.push_back(".attribute unaligned_access, 0");
9      riscv_codes.push_back(".attribute stack_align, 16");
```

```
10
11      //text section
12      riscv_codes.push_back("");
13      riscv_codes.push_back(".text");
14
15      rodata.push_back(".section .rodata");
16      for(auto ir_code:ircode_gen.IR_codes){
17          switch(ir_code.opcode){
18              case FUNC_BEGIN:
19                  FUNC_BEGIN_method(&ir_code);
20                  break;
21              case FUNC_END:
22                  FUNC_END_method(&ir_code);
23                  break;
24              case OP_LI:
25                  LI_method(&ir_code);
26                  break;
27              case OP_LOAD:
28                  LOAD_method(&ir_code);
29                  break;
30              case OP_STORE:
31                  STORE_method(&ir_code);
32                  break;
33              case OP_CALL:
34                  CALL_method(&ir_code);
35                  break;
36              case OP_MOVE:
37                  MOVE_method(&ir_code);
38                  break;
39              case OP_ADD:
40                  ADD_method(&ir_code);
41                  break;
42              case OP_SUB:
43                  SUB_method(&ir_code);
44                  break;
45              case OP_MUL:
46                  MUL_method(&ir_code);
47                  break;
48              case OP_DIV:
49                  DIV_method(&ir_code);
50                  break;
51              case OP_MOD:
52                  MOD_method(&ir_code);
53                  break;
54              case OP_LABEL:
55                  LABEL_method(&ir_code);
56                  break;
57              case OP_LT:
58                  LT_method(&ir_code);
```

```
59                  break;
60              case OP_LE:
61                  LE_method(&ir_code);
62                  break;
63              case OP_GT:
64                  GT_method(&ir_code);
65                  break;
66              case OP_GE:
67                  GE_method(&ir_code);
68                  break;
69              case OP_EQ:
70                  EQ_method(&ir_code);
71                  break;
72              case OP_NE:
73                  NE_method(&ir_code);
74                  break;
75              case OP_AND:
76                  AND_method(&ir_code);
77                  break;
78              case OP_OR:
79                  OR_method(&ir_code);
80                  break;
81              case OP_BEQZ:
82                  BEQZ_method(&ir_code);
83                  break;
84              case OP_GOTO:
85                  GOTO_method(&ir_code);
86                  break;
87              case OP_G_ASSIGN:
88                  G_ASSIGN_method(&ir_code);
89                  break;
90              case OP_G_ARR_DECL:
91                  G_ARR_DECL_method(&ir_code);
92                  break;
93              case OP_G_ARR_ASSIGN:
94                  G_ARR_ASSIGN_method(&ir_code);
95                  break;
96              case OP_ARR_ASSIGN:
97                  ARR_ASSIGN_method(&ir_code);
98                  break;
99              case OP_LOAD_ARR:
100                 LOAD_ARR_method(&ir_code);
101                 break;
102             default:
103 
    //riscv_codes.push_back("ERROR!"+std::to_string(ir_code.opcode));
104                 break;
105         }
106 
```

```
        std::ofstream out("./out/" + input_file_name + ".s", std::ios::out|std::ios::trunc);
        for (auto &ins : riscv_codes)
        {
            if (ins.find(':') == -1)
            {
                out << '\t';
            }
            out << ins << std::endl;
        }
        for (auto &ins : rodata)
        {
            if (ins.find(':') == -1)
            {
                out << '\t';
            }
            out << ins << std::endl;

        }
        out.close();
    }
}
```

- `void IRListener::FUNC_BEGIN_method(IRcode * ir_code)`：完成程序开始时的处理。这里要注意对栈的操作。

```
void IRListener::FUNC_BEGIN_method(IRcode * ir_code){
    std::string func_name = ir_code->result;
    auto func_info = symbol_table.lookup_func(func_name);

    riscv_codes.push_back(".text");
    riscv_codes.push_back(".align   1");
    riscv_codes.push_back(".globl   " + func_name);
    riscv_codes.push_back(".type    " + func_name + ", @function");
    riscv_codes.push_back(func_name + ":");

    int size = func_info->frame_len;
    riscv_codes.push_back("#frame size:" + std::to_string(size));
    riscv_codes.push_back("addi    sp,sp,-" + std::to_string(size));
    riscv_codes.push_back("sd      ra," + std::to_string(size - 8) + "(sp)");
    riscv_codes.push_back("sd      s0," + std::to_string(size - 16) + "(sp)");
    riscv_codes.push_back("addi    s0,sp," + std::to_string(size));

```

```
}
```

- `void IRListener::FUNC_END_method(IRcode * ir_code)`：完成程序结束时的处理。这里要注意退栈并回到ra中存放的地址处。

```
void IRListener::FUNC_END_method(IRcode * ir_code){
    std::string func_name = ir_code->result;
    auto func_info = symbol_table.lookup_func(func_name);
    int size = func_info->frame_len;
    riscv_codes.push_back("ld      ra," + std::to_string(size - 8) + "(sp)");
    riscv_codes.push_back("ld      s0," + std::to_string(size - 16) + "(sp)");
    riscv_codes.push_back("addi    sp,sp," + std::to_string(size));
    riscv_codes.push_back("jr ra");
    riscv_codes.push_back(".size   "+func_name+",.-"+func_name);

}
```

- `void IRListener::STORE_method(IRcode * ir_code)`：完成存入内存时的处理。这里需要注意操作数是否可能为`float`和`double`类型。这两种类型需要使用对应的浮点指令。

```
void IRListener::STORE_method(IRcode * ir_code){
    if (ir_code->cls == CLS_INT || ir_code->cls == CLS_BOOL)
    {
        riscv_codes.push_back("sw   " + ir_code->result + ", " + ir_code->arg1);
    }
    else if (ir_code->cls == CLS_FLOAT)
    {
        riscv_codes.push_back("fsw   f" + ir_code->result + ", " + ir_code->arg1);
    }
    else if (ir_code->cls == CLS_DOUBLE)
    {
        riscv_codes.push_back("fsd   f" + ir_code->result + ", " + ir_code->arg1);
    }
}
```

其它

遇到的问题

- 弄错了数组在栈中的排布方向，导致返回地址被覆盖。
- 为方便使用回填技术来进行控制流的中间生成，修改的文法引入了二义性，导致了“悬空else”问题：调整文法规则顺序后解决。如下方代码所示，将`ifElseStmt`置于`ifStmt`之前，这意味着`ifElseStmt`有更高的优先级。

```
stmt
    : lVal '=' exp ';'                              # assignStmt
    | exp? ';'                                      # exprStmt
    | block                                         # blockStmt
    | 'if' '(' ifCond ')' stmt elseStmt             # ifElseStmt
    | 'if' '(' ifCond ')' stmt                      # ifStmt
    | 'while' '(' whileCond ')' stmt                # whileStmt
    | 'return' exp? ';'                             # returnStmt
    ;

```

总结

实验结果总结

- 本次实验通过了全部测试样例；
- 本设计已经成为一个完整的编译器。

分成员总结

袁欣怡

本次实验中主要负责书写实验报告，并对代码进行整理，删除一些冗余部分。本来计划完成代码优化部分，但是时间原因没有完成，希望以后有机会继续完善。

陈飞羽

本次实验中，我负责数组的中间代码生成、函数调用的操作、目标代码生成等工作。

彭思叡

本次实验中，我负责计算局部变量/临时变量在栈中的偏移、设计中间代码数据结构、布尔表达式的中间代码生成、浮点数操作的中间代码生成等部分。