18231047 王肇凯

代码生成

最初设计

中间代码：

设计为四元式的形式，即（运算符，操作数1，操作数2，结果），多个四元式存储在全局的静态变量中。在原先的每个语法分析子程序中增加语义分析的内容生成中间代码。本次作业涉及到的操作符包括加减乘除、读、写、赋值七种。

例如对于`E = A op B op C op D`的形式（op为同级运算符，例如乘除或加减），按照翻译文法生成的序列为：

```
op, A, B, #T1
op, #T1, C, #T2
op, #T2, D, #T3
:=, E, #T3
```

同时，在进入函数时产生`FUNC void main`的四元式用于标识作用域。

MIPS代码：

语法分析结束后，根据中间代码借助符号表生成。将字符串以`.asciiz`存在数据区，全局变量存在`$gp`上方，局部变量和临时变量（四元式的中间结果）分别存在s寄存器和t寄存器，若寄存器无空闲则放在`$sp$`下方。

在进行赋值和四则运算操作时，根据四元式的操作数在内存/寄存器/为常量分情况处理。

```
/* 对于a=b：
* a在寄存器，b在寄存器：move a,b
* a在寄存器，b在内存：lw a,b
* a在寄存器，b为常量：li a,b
*
* a在内存，b在寄存器：sw b,a
* a在内存，b在内存：lw reg,b  sw reg,a
* a在内存，b为常量 li reg,b sw reg,a
*/
```

```
/* 对于a=b+c：
* abc都在寄存器/常量：                    add a,b,c
* ab在寄存器/常量，c在内存（或反过来）：  lw reg2,c  add a,b,reg2
* a在寄存器，bc在内存：                  lw reg1,b  lw reg2,c add a,reg1,reg2
* a在内存，bc在寄存器/常量：             add reg1,b,c  sw reg1,a
* ab在内存，c在寄存器/常量（或反过来）：  lw reg1,b  add reg1,reg1,c  sw reg1,a
* abc都在内存：                 lw reg1,b  lw reg2,c  add reg1,reg1,reg2  sw reg1,a
*/
```

实现与完善

中间代码类实现如下：

```cpp
class MidCode {
public:
    string op;
    string num1;
    string num2;
    string result;

    MidCode(string op, string n1, string n2, string r) :
            op(std::move(op)), num1(std::move(n1)),
num2(std::move(n2)), result(std::move(r)) {};

    MidCode() = default;

    string to_str() const;
};

class MidCodeList {
public:
    static vector<MidCode> codes;
    static int code_index;
    static vector<string> strcons;
    static int strcon_index;

    static string add(const string &op, const string
&n1, const string &n2, const string &r);
    static void refactor();
    static void show();
    static void save_to_file(const string &out_path);
};
```

语义分析时调用静态方法`MidCodeList::add`生成四元式。

MIPS生成类维护变量记录当前函数作用域，方便在符号表中查找。维护变量记录t寄存器和s寄存器的使用情况以方便分配。

翻译过程基本依照前文设计。例如对于四则运算的处理如下：

```
bool a_in_reg = in_reg(code.num1);
bool b_in_reg = in_reg(code.num2);
string a = symbol_to_addr(code.num1);
string b = symbol_to_addr(code.num2);
string reg = "$k0";

if (a_in_reg) {
    if (b_in_reg) {
        generate("move", a, b);
    } else if (is_const(code.num2)) {
        generate("li", a, b);
    } else {
        generate("lw", a, b);
    }
} else {
    if (b_in_reg) {
        generate("sw", b, a);
    } else if (is_const(code.num2)) {
        generate("li", reg, b);
        generate("sw", reg, a);
    } else {
        generate("lw", reg, b);
        generate("sw", reg, a);
    }
}
```

同时为了方便debug，在每次翻译一条中间代码时生成一条注释，以表示连续的几条语句的目的。

```
# === #T170 = #T169 * num2 ===
mul $t2, $t1, $s0
```

s寄存器在进入新的函数时释放，t寄存器在第一次被读取时释放。