Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №4

по дисциплине

«Языки программирования и методы трансляции»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.

Шепрут И.

Преподаватели Еланцева И.Л.

Вариант 8

Новосибирск 2019

1. Цель работы

Изучить методы генерации кода с учетом различных промежуточных форм представления программы. Изучить методы управления памятью и особенности их использования на этапе генерации кода.

Научиться проектировать генератора кода.

1. Задание

Подмножество языка С++ включает:

* данные типа int, float, struct из элементов указанных типов
* инструкции описания переменных
* операторы присваивания в любой последовательности
* операции +, -, \*

1. Входные и выходные данные

Входные данные представляют собой дерево, полученного в результате синтаксического разбора в лабораторной работе №3. Выходные данные – это программа на языке ассемблера

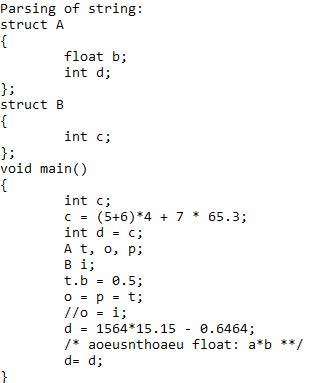
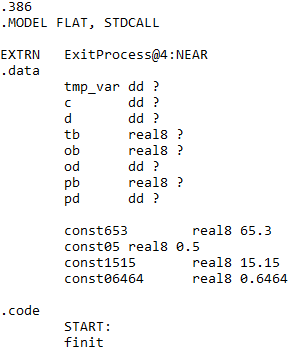
***Ограничения:***

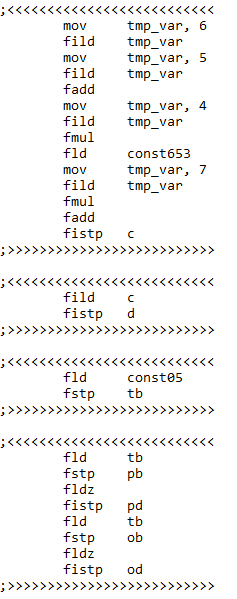
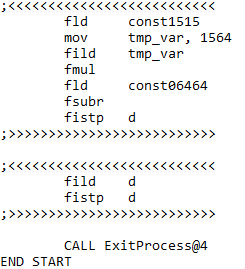
* Поскольку конечная программа на языке ассемблер для арифметических и логических операций использует арифметический сопроцессор, стек которого составляет 7 регистров, то выражения содержащие более 7 операндов в правой части могу работать не корректно.

1. **Представление выражений на языке ассемблер**

|  |  |  |
| --- | --- | --- |
| Исходное выражение на C++ | Выражение на языке ассмемблер | Комментарий |
| int a | a dd ? | Соответствие типов |
| float a | a real8 ? |
| a | fld(fild) a | Идентификатор в ариф. Выражении, тип float(int) |
| a = b | fld(fild) b  fstp(fistp) a | Присваивание тип float(int)  (если структуры то последовательно все их поля) |
| a + b | fld(fild) b  fld(fild) a  fadd | Обработка ариф. Выражений  (вместо a и b могут выступать константы) |
| a – b | fld(fild) b  fld(fild) a  fsubr |
| a \* b | fld(fild) b  fld(fild) a  fmul |
| a + 5 | mov tmp\_var, 5  fild tmp\_var  fld(fild) a  fadd | Константа целого числа |
| 0.5 | const05 real8 0.5 | Объявление вещественной константы |
| a + 0.5 | fld const05  fadd  fld(fild) a | Операции с вещественными числами |

1. **Тесты**

1. **Текст программы**

**/\*code\_gen.h\*/**

#include <vector>

#include <string>

#include <tables.h>

std::string Code\_gen(const std::vector<std::vector<Token>>& postfix, Tables& tables, std::vector<std::string>& ident, std::vector<std::string>& cons);

/\*code\_gen.cpp\*/

#include <string>

#include <code\_gen.h>

#include <fstream>

#include <iostream>

#include <optional>

#include <stack>

class CodeGen

{

public:

CodeGen(Tables& tables); //Конструктор

std::string DataCode(std::vector<std::string>& ident, std::vector<std::string>& cons);

std::string CreateCode(std::vector<Token> input);

std::string StartCode();

std::string EndCode();

private:

Tables& tables; //Хэш таблицы

std::string PrintLastStekElem(std::vector<Token>& stack);

void LieAdd(std::vector<Token>& stack);

//std::string& DelPoint(std::string& str);

std::string DelPoint(std::string str);

};

//=============================================================================

//=============================================================================

//=============================================================================

CodeGen::CodeGen(Tables& tables) : tables(tables) {

}

//std::string& CodeGen::DelPoint(std::string &a)

//{

// size\_t i = a.find('.', 0);

// if (i != string::npos)

// a.erase(i, 1);

// return a;

//}

std::string CodeGen::DelPoint(std::string a)

{

size\_t i = a.find('.', 0);

if (i != string::npos)

a.erase(i, 1);

return a;

}

std::string CodeGen::DataCode(std::vector<std::string>& ident, std::vector<std::string>& cons)

{

std::string str = ".data\n\ttmp\_var dd ?\n";

for (auto& i : ident)

{

auto t = tables.find(i);

auto arg = tables.get<Identifier>(t);

if (arg.type != TYPE\_STRUCT)

{

str += "\t" + DelPoint(i) + "\t";

if (arg.type == TYPE\_FLOAT) str += "real8 ?\n";

if (arg.type == TYPE\_INT) str += "dd ?\n";

}

}

str += "\n";

for (auto& i : cons)

{

auto arg = tables.get<Constant>(tables.find(i));

if(arg.type == TYPE\_FLOAT)

str += "\tconst" + DelPoint(i) + "\treal8 " + i + "\n";

}

str += "\n.code\n\tSTART:\n\tfinit\n\n";

return str;

}

std::string CodeGen::CreateCode(std::vector<Token> input)

{

string str = ";<<<<<<<<<<<<<<<<<<<<<<<<<<\n";

vector<Token> stack;

for(auto& i : input)

{

if (i.table == TABLE\_IDENTIFIERS || i.table == TABLE\_CONSTANTS)

stack.push\_back(i);

if (i.table == TABLE\_OPERATIONS)

{

string name = tables.getStr(i);

if (name == "=")

{

Token right = stack[stack.size() - 1];

str += PrintLastStekElem(stack);

Token left = stack[stack.size() - 1]; stack.pop\_back();

auto arg = tables.get<Identifier>(left);

if (arg.type == TYPE\_STRUCT)

{

string namel = tables.getStr(left);

string namer = tables.getStr(right);

auto args = tables.get<Structure>(tables.find(arg.nameStruct));

for (auto& elem : args.elems)

{

//auto& aaa = tables.get<Identifier>(tables.find(namel + "." + elem.name));

//aaa.isInitialized = true;

//aaa.type = elem.type;

if (elem.type == TYPE\_FLOAT)

{

if (tables.get<Identifier>(tables.find(namer + "." + elem.name)).isInitialized)

str += "\tfld\t" + namer + elem.name + "\n";

else

str += "\tfldz\n";

str += "\tfstp\t" + namel + elem.name + "\n";

}

if (elem.type == TYPE\_INT)

{

if (tables.get<Identifier>(tables.find(namer + "." + elem.name)).isInitialized)

str += "\tfild\t" + namer + elem.name + "\n";

else

str += "\tfldz\n";

str += "\tfistp\t" + namel + elem.name + "\n";

}

}

if(stack.size() != 0)

stack.push\_back(right);

}

else

{

if (stack.size() == 0)

{

if (arg.type == TYPE\_INT) str += "\tfistp\t" + DelPoint(tables.getStr(left)) + "\n";

if (arg.type == TYPE\_FLOAT) str += "\tfstp\t" + DelPoint(tables.getStr(left)) + "\n";

}

else

{

if (arg.type == TYPE\_INT) str += "\tfist\t" + DelPoint(tables.getStr(left)) + "\n";

if (arg.type == TYPE\_FLOAT) str += "\tfst\t" + DelPoint(tables.getStr(left)) + "\n";

LieAdd(stack);

}

}

continue;

}

for (size\_t k = 0; k < 2; k++)

str += PrintLastStekElem(stack);

LieAdd(stack);

if (name == "+") str += "\tfadd\n";

if (name == "-") str += "\tfsubr\n";

if (name == "\*") str += "\tfmul\n";

}

}

str += ";>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n";

return str;

}

void CodeGen::LieAdd(std::vector<Token>& stack)

{

Token a;

a.table = TABLE\_NULL;

a.pos.line = 0;

a.pos.pos = 0;

stack.push\_back(a);

}

std::string CodeGen::PrintLastStekElem(std::vector<Token>& stack)

{

std::string str;

Token tok = stack[stack.size() - 1]; stack.pop\_back();

if (tok.table == TABLE\_IDENTIFIERS)

{

auto arg = tables.get<Identifier>(tok);

if (arg.type == TYPE\_INT) str += "\tfild\t" + DelPoint(tables.getStr(tok)) + "\n";

if (arg.type == TYPE\_FLOAT) str += "\tfld\t" + DelPoint(tables.getStr(tok)) + "\n";

}

if (tok.table == TABLE\_CONSTANTS)

{

auto arg = tables.get<Constant>(tok);

if (arg.type == TYPE\_INT)

{

str += "\tmov \ttmp\_var, " + tables.getStr(tok) + "\n";

str += "\tfild \ttmp\_var\n";

}

if (arg.type == TYPE\_FLOAT) str += "\tfld \tconst" + DelPoint(tables.getStr(tok)) + "\n";

}

return str;

}

std::string CodeGen::StartCode()

{

return ".386\n.MODEL FLAT, STDCALL\n\nEXTRN ExitProcess@4:NEAR\n";

}

std::string CodeGen::EndCode()

{

return "\tCALL ExitProcess@4\nEND START";

}

//-----------------------------------------------------------------------------

string Code\_gen(const std::vector<std::vector<Token>>& postfix, Tables& tables, std::vector<std::string>& ident, std::vector<std::string>& cons){

CodeGen gen(tables);

std::string code = gen.StartCode();

code += gen.DataCode(ident, cons);

for (auto& i : postfix)

code += gen.CreateCode(i);

code += gen.EndCode();

return code;

}

/\*test\_code\_gen\*/

#include <iostream>

#include <iomanip>

#include <lex\_analyzer.h>

#include <syn\_analyzer.h>

#include <code\_gen.h>

int main() {

std::string file = "struct A\n{\n\tfloat b;\n\tint d;\n};\nstruct B\n{\n\tint c;\n};\nvoid main()\n{\n\tint c;\n\tc = (5+6)\*4 + 7 \* 65.3;\n\tint d = c;\n\tA t, o, p;\n\tB i;\n\tt.b = 0.5;\n\to = p = t;\n\t//o = i;\n\td = 1564\*15.15 - 0.6464;\n\t/\* aoeusnthoaeu float: a\*b \*\*/\n\td= d;\n}";

file += ' ';

std::cout << "Parsing of string: \n" << file << std::endl;

try

{

Tables tables;

auto tokens = parse(file, tables);

string path = "syn\_table.txt";

std::vector<std::string> ident;

std::vector<std::string> cons;

auto postfix = Analyzer(tokens, tables, path, ident, cons);

//for (auto& i : postfix)

//{

// for (auto& j : i)

// {

// std::cout << tables.getStr(j) << " ";

// }

// std::cout << std::endl;

//}

cout << endl << endl;

auto result = Code\_gen(postfix, tables, ident, cons);

std::cout << result.c\_str() << endl;

}

catch (string &e)

{

std::cerr << e << endl;

}

system("pause");

}