Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №4

по языкам программирования и методам трансляции

Факультет: ПМИ

Группа: ПМ-01

Студент: Конев А.М., Ряховский М.И.

Вариант: 6

Преподаватель: Еланцева И.Л., Полетаева И.А.

Новосибирск

2013

# Цель лабораторной работы

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

# Входные и выходные данные

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

*Ограничение на входные данные:*

* Поскольку конечная программа на языке assembler для арифметических и логических операций использует арифметический сопроцессор, стек которого состоит из 7-ми регистров, то выражения содержащие более 7 операндов в правой части будут работать не корректно.
* Из-за особенности формирования меток некоторые комбинации с логическими операциями могут обрабатываться не корректно.

# Представление выражений на языке assembler

|  |  |  |
| --- | --- | --- |
| Исходное выражение на C++ | Выражение на assmbler | Комментарий |
| int a; | a dd ? | Соответствие типов |
| char a; | a db ? |
| float a; | a real8 ? |
| a | fld a | идентификатор в ариф. выражении, тип a - float |
| a | fild a | идентификатор в ариф. выражении, тип a – int или char |
| a = b | fld b  fstp a | Присваивание, тип a – float, b- float |
| a = b | fild b  fstp a | Присваивание, тип a – float, b- int или char |
| a = b | fld b  fistp a | Присваивание, тип a – int или char, b- float |
| a == b | fld a  fld b  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  je jmp\_mark\_eq\_ab  ;else  fldz  jmp jmp\_mark\_eq\_ab\_end  ;then  jmp\_mark\_eq\_ab:  fld1  jmp\_mark\_eq\_ab\_end: | Обработка логических операций. Первые две строчки – получение левых и правых операндов, обрабатывается в зависимости от выражения. |
| a + b | fld a  fld b  fsum | Обработка логических операций. Первые две строчки – получение левых и правых операндов, обрабатывается в зависимости от выражения. |
| 1.4 | constname14 | Обработка вещественных констант – превращение их в именованные константы |
| -1.4 | constname\_minus\_14 |
| a + 1.4 | fld a  fld constname14  fsum | Обработка ариф выражений с вещ. константами |

# Тесты

|  |  |
| --- | --- |
| Исходная программа | Программа на assembler’е |
| void main (){  float var1, a = 20, b = 0.1, var2 = -1.4;  int rez1 = 0, rez2 = 2, b1;    //Присваивание целому вещественного и наоборот  b1 = b;  var1 = rez2;  //Логические операции  char bool1 = a==b, bool2, bool3, bool4;  bool2 = rez1 != 0;  bool3 = b > var2;  bool4 = bool3 < bool2;  //Некоторое сложное выражение  b1 = rez2 + b \*(1 - (a - var2));  rez1 = b1 - 3.4;  var2 += 4.6;  //Все типы операций разом  float total\_rez = a != 5 - (b==1 \* 4);  } | .386  .MODEL FLAT, STDCALL  EXTRN ExitProcess@4:NEAR  ;сегмент данных  .data  tmp\_var dd ? ;переменная, для преобразования типов  tmp\_var\_eq dd ? ;переменная, для сравнения  a real8 ?  b real8 ?  b1 dd ?  bool1 dw ?  bool2 dw ?  bool3 dw ?  bool4 dw ?  rez1 dd ?  rez2 dd ?  total\_rez real8 ?  var1 real8 ?  var2 real8 ?  ;Вещественные константы  constname01 real8 0.1  constname\_minus\_14 real8 -1.4  constname34 real8 3.4  constname46 real8 4.6  .code  START:  ;Инициализпция математического со процессора  finit  ;Начало выражения  mov tmp\_var, 20  fild tmp\_var  fstp a  ;Конец выражения  ;Начало выражения  fld constname01  fstp b  ;Конец выражения  ;Начало выражения  fld constname\_minus\_14  fstp var2  ;Конец выражения  ;Начало выражения  mov tmp\_var, 0  fild tmp\_var  fistp rez1  ;Конец выражения  ;Начало выражения  mov tmp\_var, 2  fild tmp\_var  fistp rez2  ;Конец выражения  ;Начало выражения  fld b  fistp b1  ;Конец выражения  ;Начало выражения  fild rez2  fstp var1  ;Конец выражения  ;Начало выражения  fld a  fld b  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  je jmp\_mark\_eq\_ab  ;else  fldz  jmp jmp\_mark\_eq\_ab\_end  ;then  jmp\_mark\_eq\_ab:  fld1  jmp\_mark\_eq\_ab\_end:  fistp bool1  ;Конец выражения  ;Начало выражения  fild rez1  mov tmp\_var, 0  fild tmp\_var  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  jne jmp\_mark\_neq\_rez10  ;else  fldz  jmp jmp\_mark\_neq\_rez10\_end  ;then  jmp\_mark\_neq\_rez10:  fld1  jmp\_mark\_neq\_rez10\_end:  fistp bool2  ;Конец выражения  ;Начало выражения  fld b  fld var2  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  jb jmp\_mark\_l\_bvar2  ;else  fldz  jmp jmp\_mark\_l\_bvar2\_end  ;then  jmp\_mark\_l\_bvar2:  fld1  jmp\_mark\_l\_bvar2\_end:  fistp bool3  ;Конец выражения  ;Начало выражения  fild bool3  fild bool2  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  jl jmp\_mark\_l\_bool3bool2  ;else  fldz  jmp jmp\_mark\_l\_bool3bool2\_end  ;then  jmp\_mark\_l\_bool3bool2:  fld1  jmp\_mark\_l\_bool3bool2\_end:  fistp bool4  ;Конец выражения  ;Начало выражения  fild rez2  fld b  mov tmp\_var, 1  fild tmp\_var  fld a  fld var2  fsub  fsub  fmul  fadd  fistp b1  ;Конец выражения  ;Начало выражения  fild b1  fld constname34  fsub  fistp rez1  ;Конец выражения  ;Начало выражения  fld constname46  fld var2  fadd  fstp var2  ;Конец выражения  ;Начало выражения  fld a  mov tmp\_var, 5  fild tmp\_var  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  jne jmp\_mark\_neq\_a5  ;else  fldz  jmp jmp\_mark\_neq\_a5\_end  ;then  jmp\_mark\_neq\_a5:  fld1  jmp\_mark\_neq\_a5\_end:  fld b  mov tmp\_var, 1  fild tmp\_var  fstp tmp\_var\_eq  fcom tmp\_var\_eq  fstp tmp\_var\_eq  fstsw ax  sahf  je jmp\_mark\_eq\_b1  ;else  fldz  jmp jmp\_mark\_eq\_b1\_end  ;then  jmp\_mark\_eq\_b1:  fld1  jmp\_mark\_eq\_b1\_end:  mov tmp\_var, 4  fild tmp\_var  fmul  fsub  fstp total\_rez  ;Конец выражения  CALL ExitProcess@4  END START |

# Код модуля генерации кода

## Файл «translator.h»

string generate\_part\_float(tree\_el \*beg); //генерирует часть дерева

int check\_tree\_type(tree\_el \*beg); //определяет тип выражения

map<string ,string> float\_constants; //каждой константе (вещественной) присвоим имя

## Файл «translator.cpp»

void translator::generate\_code(string code\_file){

ofstream code\_gen\_code(code\_file.c\_str()); //поток для вывода кода в файл

//Формируем верх

code\_gen\_code << ".386\n.MODEL FLAT, STDCALL\n\n";

code\_gen\_code << "EXTRN\tExitProcess@4:NEAR\n";

//Формируем блок данных

string type\_str; // строка, описывающая тип переменной

code\_gen\_code << "\n\n;сегмент данных\n.data\n";

code\_gen\_code << "\ttmp\_var\tdd\t?" << " ;переменная, для преобразования типов" << endl;

code\_gen\_code << "\ttmp\_var\_eq\tdd\t?" << " ;переменная, для сравнения" << endl;

for(int i = 10; i < 32; i++){

bool is\_els = true; //есть ли элементы

int j = 0; //номер элемента в цепочке

lexeme check\_lex; //получаемая лексема

while(is\_els){

is\_els = identifier.get\_lexeme(i, j, check\_lex);

if(is\_els){

//Формируем тип

switch(check\_lex.type){

case 1:{

type\_str = "dd";

} break;

case 2:{

type\_str = "dw";

} break;

case 3:{

type\_str = "real8";

}break;

};

//Выводим объявление перменной

code\_gen\_code << "\t" + check\_lex.name + "\t" + type\_str + " ?" << endl;

j++;

}

};

}

//А тут мы обрабатываем наши константы

code\_gen\_code << "\n\t;Вещественные константы\n";

for(int i = 0; i < 10; i++){

bool is\_els = true; //есть ли элементы

int j = 0; //номер элемента в цепочке

lexeme check\_lex; //получаемая лексема

while(is\_els){

is\_els = consts.get\_lexeme(i, j, check\_lex);

if(is\_els){

if(check\_lex.name.find\_first\_of('.') != string::npos){ // если вещственная константа

consts.set\_type\_address(i, j, 3);

//Формируем имя переменной

string tmp\_str, tmp\_str2, var\_name;

if(check\_lex.name[0] == '-'){ //минус в имени переменной - плохо

tmp\_str.assign(check\_lex.name, 1, check\_lex.name.find\_first\_of('.')-1);

tmp\_str = "\_minus\_" + tmp\_str;

}

else

tmp\_str.assign(check\_lex.name, 0, check\_lex.name.find\_first\_of('.'));

tmp\_str2.assign(check\_lex.name.begin()+check\_lex.name.find\_first\_of('.')+1, check\_lex.name.end());

var\_name = "constname" + tmp\_str + tmp\_str2; //имя переменной

code\_gen\_code <<"\t" + var\_name + "\treal8\t" + check\_lex.name << endl; //выводим этот ужас

//И запомниаем в map, чтобы не считать ещё раз

float\_constants[check\_lex.name] = var\_name;

}

else

consts.set\_type\_address(i, j, 2);

j++;

}

};

}

//И вот мы начинам формировать код...

code\_gen\_code << "\n\n.code\n";

code\_gen\_code << "\tSTART:" << endl;

code\_gen\_code << ";Инициализпция математического со процессора\n\tfinit\n\n" << endl;

tree\_cur = tree\_begin;

while(tree\_cur != NULL){

code\_gen\_code << ";Начало выражения\n";

string tmp\_str;

int check\_type = check\_tree\_type(tree\_cur);

tmp\_str = generate\_part\_float(tree\_cur);

code\_gen\_code << tmp\_str << endl;

code\_gen\_code << ";Конец выражения\n\n\n";

//Передвигаемся на следующее поддерево

while(tree\_cur->id != ";")

tree\_cur = tree\_cur->right;

tree\_cur = tree\_cur->right;

};

code\_gen\_code << "\tCALL ExitProcess@4\n";

code\_gen\_code << "END START\n";

code\_gen\_code.close();

}

string translator::generate\_part\_float(tree\_el \*beg){

string rez\_str = "";

if(beg->id == ";"){

tree\_cur = beg;

return rez\_str;

}

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

//== Присваивание ==

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

if(beg->id == "="){

rez\_str += generate\_part\_float(beg->right);

//В зависимости от типа выполняем различные извлечения из стека

lexeme tmp\_lex;

identifier.get\_lexeme(beg->left->id, tmp\_lex);

if(tmp\_lex.type == 3)

rez\_str += "\tfstp " + beg->left->id + "\n\n";

else

rez\_str += "\tfistp " + beg->left->id + "\n\n";

}

if(beg->id == "+="){

rez\_str += generate\_part\_float(beg->right);

//В зависимости от типа выполняем различные извлечения из стека

lexeme tmp\_lex;

identifier.get\_lexeme(beg->left->id, tmp\_lex);

if(tmp\_lex.type == 3){

rez\_str += "\tfld " + beg->left->id + "\n";

rez\_str += "\tfadd\n";

rez\_str += "\tfstp " + beg->left->id + "\n\n";

}

else{

rez\_str += "\tfild " + beg->left->id + "\n";

rez\_str += "\tfadd\n";

rez\_str += "\tfistp " + beg->left->id + "\n\n";

}

}

if(beg->id == "-="){

rez\_str += generate\_part\_float(beg->right);

//В зависимости от типа выполняем различные извлечения из стека

lexeme tmp\_lex;

identifier.get\_lexeme(beg->left->id, tmp\_lex);

if(tmp\_lex.type == 3){

rez\_str += "\tfld " + beg->left->id + "\n";

rez\_str += "\tfsubr\n";

rez\_str += "\tfstp " + beg->left->id + "\n\n";

}

else{

rez\_str += "\tfild " + beg->left->id + "\n";

rez\_str += "\tfsubr\n";

rez\_str += "\tfistp " + beg->left->id + "\n\n";

}

}

if(beg->id == "\*="){

rez\_str += generate\_part\_float(beg->right);

//В зависимости от типа выполняем различные извлечения из стека

lexeme tmp\_lex;

identifier.get\_lexeme(beg->left->id, tmp\_lex);

if(tmp\_lex.type == 3){

rez\_str += "\tfld " + beg->left->id + "\n";

rez\_str += "\tfmul\n";

rez\_str += "\tfstp " + beg->left->id + "\n\n";

}

else{

rez\_str += "\tfild " + beg->left->id + "\n";

rez\_str += "\tfmul\n";

rez\_str += "\tfistp " + beg->left->id + "\n\n";

}

}

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

//==== Операции ===

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

//Поскольку идём мы сначало влево, потом в право, то первым в стеке будет лежать второй аргумент, а за ним - первый

if(beg->id == "+"){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

rez\_str += "\tfadd\n";

}

if(beg->id == "-"){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

rez\_str += "\tfsub\n";

}

if(beg->id == "\*"){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

rez\_str += "\tfmul\n";

}

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

//===== Логика =====

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

if(beg->id == "=="){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

string mark = "jmp\_mark\_eq\_" + beg->left->id + beg->right->id;

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfcom tmp\_var\_eq\n";

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfstsw ax\n\tsahf\n";

rez\_str += "\tje " + mark + "\n";

//false

rez\_str += "\t;else\n";

rez\_str += "\t\tfldz\n";

rez\_str += "\t\tjmp " + mark + "\_end\n";

//true

rez\_str += "\t;then\n";

rez\_str += "\t" + mark + ":\n";

rez\_str += "\t\tfld1\n\n";

rez\_str += "\t" + mark + "\_end:\n\n";

}

if(beg->id == "!="){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

string mark = "jmp\_mark\_neq\_" + beg->left->id + beg->right->id;

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfcom tmp\_var\_eq\n";

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfstsw ax\n\tsahf\n";

rez\_str += "\tjne " + mark + "\n";

//false

rez\_str += "\t;else\n";

rez\_str += "\t\tfldz\n";

rez\_str += "\t\tjmp " + mark + "\_end\n";

//true

rez\_str += "\t;then\n";

rez\_str += "\t" + mark + ":\n";

rez\_str += "\t\tfld1\n\n";

rez\_str += "\t" + mark + "\_end:\n\n";

}

if(beg->id == "<"){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

string mark = "jmp\_mark\_l\_" + beg->left->id + beg->right->id;

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfcom tmp\_var\_eq\n";

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfstsw ax\n\tsahf\n";

rez\_str += "\tjl " + mark + "\n";

//false

rez\_str += "\t;else\n";

rez\_str += "\t\tfldz\n";

rez\_str += "\t\tjmp " + mark + "\_end\n";

//true

rez\_str += "\t;then\n";

rez\_str += "\t" + mark + ":\n";

rez\_str += "\t\tfld1\n\n";

rez\_str += "\t" + mark + "\_end:\n\n";

}

if(beg->id == ">"){

rez\_str += generate\_part\_float(beg->left);

rez\_str += generate\_part\_float(beg->right);

string mark = "jmp\_mark\_l\_" + beg->left->id + beg->right->id;

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfcom tmp\_var\_eq\n";

rez\_str += "\tfstp tmp\_var\_eq\n";

rez\_str += "\tfstsw ax\n\tsahf\n";

rez\_str += "\tjb " + mark + "\n";

//false

rez\_str += "\t;else\n";

rez\_str += "\t\tfldz\n";

rez\_str += "\t\tjmp " + mark + "\_end\n";

//true

rez\_str += "\t;then\n";

rez\_str += "\t" + mark + ":\n";

rez\_str += "\t\tfld1\n\n";

rez\_str += "\t" + mark + "\_end:\n\n";

}

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

//=== Переменные и константы ====

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

if(beg->type == 1){

lexeme tmp\_lex;

bool is\_id = identifier.get\_lexeme(beg->id, tmp\_lex);

if(is\_id){ //Переменные

if(tmp\_lex.type == 3){

rez\_str += "\tfld " + beg->id + "\n";

}

else{

rez\_str += "\tfild "+ beg->id + "\n";

}

}

else{ //Константы

consts.get\_lexeme(beg->id, tmp\_lex);

if(tmp\_lex.type == 3){

string const\_name = float\_constants.find(beg->id)->second; //получаем имя константы

rez\_str += "\tfld " + const\_name + "\n";

}

else{

rez\_str += "\tmov tmp\_var, " + beg->id + "\n";

rez\_str += "\tfild tmp\_var\n";

}

}

}

return rez\_str;

}

int translator::check\_tree\_type(tree\_el \*beg){

if(beg->type == 0){

int type\_l = check\_tree\_type(beg->left);

int type\_r = check\_tree\_type(beg->right);

if(type\_l == type\_r)

return type\_l;

if(type\_r == 3 || type\_l == 3)

return 3;

if(type\_r == 1 || type\_l == 1)

return 1;

if(type\_r == 2 || type\_l == 2)

return 2;

}

if(beg->type == 1){

lexeme tmp\_lex;

if(identifier.get\_lexeme(beg->id, tmp\_lex))

return tmp\_lex.type;

if(consts.get\_lexeme(beg->id, tmp\_lex))

return tmp\_lex.type;

}

}