

ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

канд. техн. наук, доцент

должность, уч. степень, звание

подпись, дата

Н.В. Кучин

инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ №6-7

ГЕНЕРАЦИЯ И ОПТИМИЗАЦИЯ ОБЪЕКТНОГО КОДА

по курсу: СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4041

подпись, дата

Н. А. Выборнов

инициалы, фамилия

Санкт-Петербург 2023

1 Задание по лабораторной работе

Получен вариант 2.

Входной язык содержит логические выражения, разделенные символом ; (точка с запятой). Логические выражения состоят из идентификаторов, констант true («истина») и false («ложь»), знака присваивания (:=), знаков операций or, xor, and, not и круглых скобок.

Комментарий # ... #

$S \rightarrow a := F;$

$F \rightarrow F \text{ or } T \mid F \text{ xor } T \mid T$

$T \rightarrow T \text{ and } E \mid E$

$E \rightarrow (F) \mid \text{not } (F) \mid a$

2 Цель работы

Изучение основных принципов генерации компилятором объектного кода, выполнение генерации объектного кода программы на основе результатов синтаксического анализа для заданного входного языка.

Изучение основных принципов оптимизации компилятором объектного кода для линейного участка программы, ознакомление с методами оптимизации результирующего объектного кода с помощью методов свертки объектного кода и исключения лишних операций.

3. Код программы (Python 3.10)

Файл main.py

```
import parser
import lexer
import input_triads

def fold_triads(triads):
    """Оптимизация методом свёртки"""
    Table = {}

    new_triads = {}
    for index, triad in triads.items():
        # op1 = 1 if triad[1] == 'true' else 0 if triad[1] == 'false'
        else triad[1]
        # op2 = 1 if triad[2] == 'true' else 0 if triad[2] == 'false'
        else triad[2]
        op1 = triad[1]
        op2 = triad[2]
        operator = triad[0]

        # Заменяем операнды на значения из таблицы T, если они там есть
        if op1 in Table and (isinstance(op1, int) or op1 not in ('true',
            'false')) and operator != ':=':
            op1 = Table[op1]

        if op2 in Table and (isinstance(op2, int) or op2 not in ('true',
            'false')) and operator != ':=':
            op2 = Table[op2]

        #таблица действий
        wrap_table = {
            "or": lambda op1, op2: op1 or op2,
            "and": lambda op1, op2: op1 and op2,
            "xor": lambda op1, op2: op1 ^ op2,
        }
        # сворачиваем триаду
        if operator == ":=":
            new_triads[index] = [operator, op1, op2]
            Table[op1] = op2
            continue
        elif operator == "not":
            new_triads[index] = ["C", -op1, 0]
        elif operator in wrap_table:
            if op1 not in ('true', 'false'):
                op1 = Table[op1]
            calc = str(wrap_table[operator](op1, op2))
            new_triads[index] = ["C", calc, 0]
            Table[index] = calc
        else:
            new_triads[index] = [operator, op1, op2]
    return new_triads

def delete_extra_operations(triads):
    # Создаем словарь зависимостей для переменных и триад
    traid_dep = {}
    def var_dep():
        dependencies = {} # числа зависимости для переменных
        for i, triad in triads.items():
            dep1 = dependencies.get(triad[1], 0)
            dep2 = dependencies.get(triad[2], 0)
            traid_dep[i] = 1 + max(dep1, dep2)
```

```

        # Если текущая триада есть присвоение, обновляем число
зависимости
        if triad[0] == "!=":
            dependencies[triad[1]] = i
        return dependencies

def process_ident_triads():
    f = True
    #словарик поиска дублей
    second_double = {}
    for i, triad in triads.items():
        op_i, op_1_i, op2_i = triad
        #если один из операторов ссылается на сэйм
        if isinstance(op_1_i, int) and triads[op_1_i][0] == 'SAME':
            value = second_double.get((op_i, triads[op_1_i][1],
op2_i,)), -1)
            if value != -1:
                triads[i] = ['SAME', value, 0]
            else:
                triads[i] = [op_i, triads[op_1_i][1], op2_i]
                second_double[(op_i, triads[op_1_i][1], op2_i,)] = i

        elif isinstance(op2_i, int) and triads[op2_i][0] == 'SAME':
            key_tuple = (op_i, op_1_i, triads[op2_i][1])
            value = second_double.get(key_tuple, -1)

            if value != -1:
                triads[i] = ['SAME', value, 0]
            else:
                triads[i] = [op_i, op_1_i, triads[op2_i][1]]
                second_double[key_tuple] = i

        #ищем в пределах шага 3, если количество зависимостей
совпадает
        #идем по алгоритму
        for j in range(i):
            if triad_dep[j] == triad_dep[i] and \
                op_i == triads[j][0] and op_1_i == triads[j][1]
and op2_i == triads[j][2]:
                triads[i] = ['SAME', j, 0]
                f = False
                break
        if f:
            triads[i] = triad

def for_print(dependencies):
    print('Числа зависимости переменных')
    print(dependencies)
    print('Числа зависимости триад')
    print(triad_dep)

dependencies = var_dep()
process_ident_triads()
for print(dependencies)

return triads

def generate_assembly(triads):
    assembly = 'section .data\n'

```

```

# всех используемых переменных
variables = set()
for key in triads:
    op1 = triads[key][1]
    op2 = triads[key][2]
    if op1 not in ['true', 'false'] and type(op1) != type(1):
        variables.add(op1)
    if op2 not in ['true', 'false'] and type(op2) != type(1):
        variables.add(op2)
variables.discard(None)
for el in variables:
    assembly += f'\t{el} db 0\n'

assembly += '\nsection .text\n\tglobal _start\n\n'
assembly += '_start:\n'

for triad in triads:
    operation = triads[triad][0]
    op1 = triads[triad][1]
    op2 = triads[triad][2]
    code = ''

    if operation == '==':
        if op2 in variables:
            code = f'\tmov al, {op2}\n'
            code += f'\tmov [{op1}], {op2}\n'
        elif op2 in ['true', 'false']:
            op2 = 1 if op2 == 'true' else 0
            code = f'\tmov [{op1}], {op2}\n'
        else:
            code = f'\tpop al\n'
            code += f'\tmov [{op1}], al\n'

    elif operation in ['and', 'or', 'xor']:
        if op1 in variables and op2 in variables:
            code = f'\tmov al, {op1}\n'
            code += f'\tmov bl, {op2}\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op1 in variables and op2 in ['true', 'false']:
            op2 = 1 if op2 == 'true' else 0
            code = f'\tmov al, {op1}\n'
            code += f'\tmov bl, {op2}\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op1 in variables:
            code = f'\tmov al, {op1}\n'
            code += f'\tpop bl\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op1 in ['true', 'false'] and op2 in variables:
            op1 = 1 if op1 == 'true' else 0
            code = f'\tmov al, {op1}\n'
            code += f'\tmov bl, {op2}\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op1 in ['true', 'false'] and op2 in ['true', 'false']:
            op1 = 1 if op1 == 'true' else 0
            op2 = 1 if op2 == 'true' else 0
            code = f'\tmov al, {op1}\n'
            code += f'\tmov bl, {op2}\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'

```

```

        elif op1 in ['true', 'false']:
            op1 = 1 if op1 == 'true' else 0
            code = f'\tmov al, {op1}\n'
            code += f'\tpop bl\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op2 in variables:
            code = f'\tmov al, {op2}\n'
            code += f'\tpop bl\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        elif op2 in ['true', 'false']:
            op2 = 1 if op2 == 'true' else 0

            code = f'\tmov al, {op2}\n'
            code += f'\tpop bl\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'
        else:
            code = f'\tpop al\n'
            code += f'\tpop bl\n'
            code += f'\t{operation} al, bl\n'
            code += f'\tpush al\n'

    elif operation == 'not':
        if op1 in variables:
            code = f'\tmov al, {op1}\n'
            code += f'\tnot al\n'
            code += f'\tpush al\n'
        elif op1 in ['true', 'false']:
            op1 = 1 if op1 == 'true' else 0
            code = f'\tmov al, {op1}\n'
            code += f'\tnot al\n'
            code += f'\tpush al\n'
        else:
            code = f'\tpop al\n'
            code += f'\tnot al\n'
            code += f'\tpush al\n'

    assembly += code
print(assembly)

if __name__ == "__main__":
    with open("code.txt") as file:
        lex_lst = lexer(file.read())
    #Вызываем метод parse, передавая список лексем входной цепочки
    tree = parser.parse(lex_lst)
    triads = {}
    for node in tree:
        part = input_triads.generate_triads(node)
        triads.update(part)
    print('Полученные триады:')
    for i, j in triads.items():
        print(i, j)
    print('Оптимизация методом свертки с промежуточным результатом')
    fold_triads_with_special = fold_triads(triads)
    fold_triads_without_special = {}
    for i, j in fold_triads_with_special.items():
        print(i, j)
        if j[0] != 'C':
            fold_triads_without_special[i] = j
    print('Оптимизация методом свертки без особых триад')

```

```

for i, j in fold_triads_without_special.items():
    print(i, j)

x = delete_extra_operations(triads)

print('Триады после выполнения алгоритма исключения лишних
операций')
for i, j in x.items():
    print(i, j)

print('Генерация ассемблера')
x = generate_assembly(triads)

```

файл input_triad.py

```

def generate_triad(node):

    # левое и правое значение текущего узла
    left = node.left
    right = node.right

    # временный словарь
    part_of_dict = {}
    # если левое значение узел то рекурсивно обновить словарь
    if type(node.left) == type(node):
        cmd_left = code(node.left)
        left = node.left.ID
        part_of_dict.update(cmd_left)
    # если правое значение узел то рекурсивно обновить словарь
    if type(node.right) == type(node):
        cmd_right = code(node.right)
        right = node.right.ID
        part_of_dict.update(cmd_right)
    # запись значений в словарь (создание триады)
    part_of_dict.update({node.ID: [node.operation, left, right]})

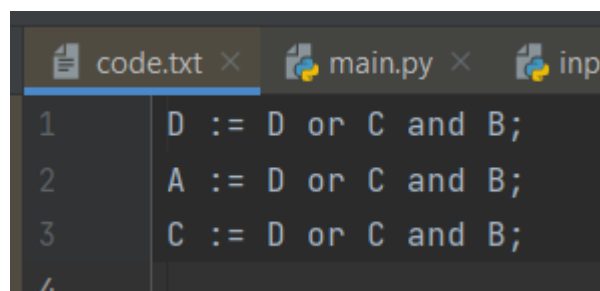
    return part_of_dict

```

Тестирование программы

Тест 1 из методических указаний для проверки алгоритма удаления дублирующих действий

(без других алгоритмов, потому что C и B не инициализированы для первой строки)



```

1 D := D or C and B;
2 A := D or C and B;
3 C := D or C and B;
4

```

Рисунок 1 – Тест 1

```
Полученные триады:
0 ('and', 'C', 'B')
1 ('or', 'D', 0)
2 (':=' , 'D', 1)
3 ('and', 'C', 'B')
4 ('or', 'D', 3)
5 (':=' , 'A', 4)
6 ('and', 'C', 'B')
7 ('or', 'D', 6)
8 (':=' , 'C', 7)
Числа зависимости переменных
{'D': 2, 'A': 5, 'C': 8}
Числа зависимости триад
{0: 1, 1: 1, 2: 1, 3: 1, 4: 3, 5: 1, 6: 1, 7: 3, 8: 1}
Триады после выполнения алгоритма исключения лишних операций
0 ('and', 'C', 'B')
1 ('or', 'D', 0)
2 (':=' , 'D', 1)
3 ['SAME', 0, 0]
4 ['or', 'D', 0]
5 (':=' , 'A', 4)
6 ['SAME', 0, 0]
7 ['SAME', 4, 0]
8 [':=' , 'C', 4]
Генерация ассемблера
section .data
```

Рисунок 1 – Вывод программы

Тест 2 для полной проверки функциональности программы. Файл **code.txt**


```
code.txt × main.py × input_triads.py ×
1 var := false or false;
2 var := true;
3 my_var2 := true and var or var;
```

Рисунок 3 – Тест 2

```
Генерация ассемблера
section .data
    my_var2 db 0
    var db 0

section .text
    global _start

_start:
    mov al, 0
    mov bl, 0
    or al, bl
    push al
    pop al
    mov [var], al
    mov [var], 1
    mov al, 1
    mov bl, var
    and al, bl
    push al
    mov al, var
    pop bl
    or al, bl
    push al
    pop al
    mov [my_var2], al
```

Рисунок 5 – Вывод ассемблера

Вывод

В результате выполнения лабораторной работы, были получены навыки по формированию списка триад по полученному в предыдущих работах дереву синтаксического разбора. Были получены навыки по оптимизации списка полученный триад при помощи метода свертки и метода исключения лишних операций. Написана программная реализация порождения триад и ассемблерного кода и их оптимизации.