# Лабораторная работа 1

## Моделирование работы ассемблера и процессора для вычисления выражений

**Задание:**

Разработать объектно-ориентированное приложение и unit тесты для проверки работы классов, которое моделирует работы ассемблера.

1. Разработать класс для чтения из текстового файла

class IOFile {

public:

virtual bool open(const std::string& fileName) = 0;

virtual void close() = 0;

virtual std::vector<std::string> read (const std::string&) = 0;

virtual write (std::vector<std::string> const std::string&);

};

class IOFileTxt : IOFile {

…

};

class IOFileJson : IOFile {

…

};

1. Разработать класс для парсинга блока данных и блока кода

class Parser {

private:

typedef std::vector<std::string>::iterator VIt;

public:

std::pair<VIt, VIt> getData(std::vector<std::string>&);

std::pair<VIt, VIt> getCode(std::vector<std::string>&);

};

1. Разработать иерархию классов-команд

class Command {

public:

virtual void exec(int&, int&, int&) = 0;

private:

Repo\* data;

};

class Mov : public Command {

public:

void exec(int&, int&, int&);

};

class Add : public Command {

public:

void exec(int&, int&, int&);

};

class Sub : public Command {

public:

void exec(int&, int&, int&);

};

class Mul : public Command {

public:

void exec(int&, int&, int&);

};

struct Div : public Command {

public:

void exec(int&, int&, int&);

};

1. Разработать класс-хранилище данных

class Repo {

private:

std::map<std::string, int> data;

std::vector<Command\*> commands;

std::set<std::string> resultKeys;

public:

void execCode();

};

**Указания:**

1. Процессор использует регистр-аккумулятор **Ak**. Синтаксис языка допускает объявление переменных и резервирование памяти, а также команды бинарной арифметики (составные операторы вида @=) и пересылки.
2. Использовать шаблоны проектирования Репозиторий (url: https://habr.com/ru/articles/248505/) и Команда (url: https://ru.wikipedia.org/wiki/ Команда\_(шаблон\_проектирования)#Пример\_на\_C++).

**Пример:**

Программа для вычисления выражения **ax2-bx+c** имеет следующий вид:

(1 столбец – имя переменной, 2 столбец – ее значение, 3 столбец – дополнительные комментарии, для вычислений не важны)

.data

x00 -1 // для инверсии знака

х01 3 // a

x02 2 // b

x03 1 // c

x04 10 // x

x05 ? // результат

.code

mov Ak x01

mul Ak x04

mul Ak x04

add Ak x03

mov x05 Ak // ax2+c

mov Ak x04

mul Ak x02

mul Ak х00 // -bx

add Ak x05

mov x05 Ak

end

Строки программы вводятся из файла последовательно, интерпретируются функцией моделирования, результат возвращается функцией, результат записывается в выходной файл.

Идентификаторы заданы так, чтобы их можно было хранить в массиве, т.е. индекс легко извлекался из имени.

Команды арифметики можно сделать одноадресные, второй адрес – по умолчанию **Ak**

Далее можно предложить использование массивов, ввод данных и т. п.

## Пример иерархии классов для чтения/записи в файл:

#include <iostream>

#include <fstream>

#include <string>

#include <json/json.h> // Для работы с JSON

class File {

public:

virtual bool open(const std::string& filename) = 0;

virtual void close() = 0;

virtual bool read(std::string& content) = 0;

virtual bool write(const std::string& content) = 0;

};

class TextFile : public File {

private:

std::ifstream file;

public:

bool open(const std::string& filename) override {

file.open(filename);

return file.is\_open();

}

void close() override {

file.close();

}

bool read(std::string& content) override {

if (file.is\_open()) {

std::getline(file, content);

return true;

}

return false;

}

bool write(const std::string& content) override {

if (file.is\_open()) {

file << content;

return true;

}

return false;

}

};

class JsonFile : public File {

private:

Json::Value jsonRoot;

public:

bool open(const std::string& filename) override {

std::ifstream jsonFile(filename);

if (!jsonFile.is\_open()) {

return false;

}

jsonFile >> jsonRoot;

jsonFile.close();

return true;

}

void close() override {

// Ничего не делаем, так как JSON файл не нужно явно закрывать

}

bool read(std::string& content) override {

// Чтение JSON объекта

// Пример: content = jsonRoot["key"].asString();

return true;

}

bool write(const std::string& content) override {

// Запись JSON объекта

// Пример: jsonRoot["key"] = content;

return true;

}

};

int main() {

TextFile textFile;

JsonFile jsonFile;

if (textFile.open("textfile.txt")) {

std::string textContent;

textFile.read(textContent);

std::cout << "Text content: " << textContent << std::endl;

textFile.close();

}

if (jsonFile.open("jsonfile.json")) {

std::string jsonContent;

jsonFile.read(jsonContent);

std::cout << "JSON content: " << jsonContent << std::endl;

jsonFile.close();

}

return 0;

}