**ПРИЛОЖЕНИЕ А – Код программы**

Класс MainActivity:

**package** com.worm2fed.kursach\_prog;

**import** com.worm2fed.kursach\_prog.OpenFileDialog.OpenDialogListener;

**import** android.app.Activity;

**import** android.app.AlertDialog;

**import** android.content.DialogInterface;

**import** android.content.Intent;

**import** android.os.Bundle;

**import** android.os.Environment;

**import** android.view.View;

**import** android.widget.TextView;

**import** android.widget.Toast;

**public** **class** MainActivity **extends** Activity {

**private** String file\_path = "", path = "";

Intent asembler\_result\_activity, run\_activity;

String[] nm;

**int** mem\_size, mode\_id = 0;

// хранит результат ассемблирования

**private** **boolean** asm\_res = **false**;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

// создаём активити

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_main***);

// размер памяти

mem\_size = 256;

}

// обработчик нажатия на кнопку 'выбрать файл'

**public** **void** onOpenFileClick(View view) {

String message = "";

// проверка на доступность SD-карты

**if** (!Environment.*getExternalStorageState*().equals(

Environment.***MEDIA\_MOUNTED***)) {

message = getApplicationContext().getResources().getString(

R.string.***sdcard\_error***);

Toast.*makeText*(getApplicationContext(), message, Toast.***LENGTH\_SHORT***)

.show();

} **else** {

// создаём список с файлами для выбора

OpenFileDialog fileDialog = **new** OpenFileDialog(**this**);

fileDialog.show();

// создаём слушатель для получения путя к файлу при его выборе из

// спсика

fileDialog.setOpenDialogListener(**new** OpenDialogListener() {

@Override

**public** **void** OnSelectedFile(String fileName) {

// **TODO** Auto-generated method stub

// Связываем объект GUI с переменной

TextView filePath = (TextView) findViewById(R.id.***file\_path***);

// выводим путь файла рядом с кнопкой

filePath.setText(fileName);

// переменная, хранящаяя путь к файлу

file\_path = fileName;

path = "";

nm = file\_path.split("/");

**for** (**int** i = 0; i < (nm.length - 1); i++)

path = path + nm[i] + "/";

}

});

}

}

// обработчик нажатия на кнопку 'преобразовать в объектный код'

**public** **void** onAssemblClick(View view) {

String message = "";

**if** (file\_path.equals(""))

message = getApplicationContext().getResources().getString(

R.string.***notchosen\_error***);

**else** {

Assembler assembler = **new** Assembler(file\_path);

String mes = assembler.toAssembl();

// вытасикаем сообщение о результате ассемблирования из ресурсов

**switch** (mes) {

**case** "ext\_err":

**case** "read\_err":

**case** "close\_err":

message = getApplicationContext().getResources().getString(

R.string.***extention\_error***);

**break**;

**case** "lis\_err":

message = getApplicationContext().getResources().getString(

R.string.***listing\_error***);

**break**;

**case** "obj\_err":

message = getApplicationContext().getResources().getString(

R.string.***object\_error***);

**break**;

**case** "404\_err":

message = getApplicationContext().getResources().getString(

R.string.***notfound\_error***);

**break**;

**case** "suc":

message = getApplicationContext().getResources().getString(

R.string.***success\_asm***);

asm\_res = **true**;

**break**;

}

}

Toast.*makeText*(getApplicationContext(), message, Toast.***LENGTH\_SHORT***)

.show();

}

// обработчик нажатия на кнопку 'показать объектный код'

**public** **void** onShowObjectCodeClick(View view) {

String message = "";

// записываем путь к файлу объектного кода

message = path + "object\_code.w2f";

// инициализируем будущий переход между активити

asembler\_result\_activity = **new** Intent(MainActivity.**this**,

AssemblerResultActivity.**class**);

// передаём значение переменной в другое активити

asembler\_result\_activity.putExtra("result\_type", message);

// запускаем активити

startActivity(asembler\_result\_activity);

}

// обработчик нажатия на кнопку 'показать листинг'

**public** **void** onShowListingClick(View view) {

String message = "";

// записываем путь к файлу листинга

message = path + "listing.w2f";

// инициализируем будущий переход между активити

asembler\_result\_activity = **new** Intent(MainActivity.**this**,

AssemblerResultActivity.**class**);

// передаём значение переменной в другое активити

asembler\_result\_activity.putExtra("result\_type", message);

// запускаем активити

startActivity(asembler\_result\_activity);

}

// обработчик нажатия на кнопку 'выполнить'

**public** **void** onRunClick(View view) {

String message = "";

**if** (asm\_res == **false**) {

message = getApplicationContext().getResources().getString(

R.string.***notasm\_error***);

Toast.*makeText*(getApplicationContext(), message, Toast.***LENGTH\_SHORT***)

.show();

} **else** {

// открываем диалог для выбора режима выполнения

openRunDialog();

}

}

// окошко для выбора режима выполнения

**protected** **void** openRunDialog() {

**final** String[] m\_сhoose = getResources().getStringArray(

R.array.***mode\_arr***);

AlertDialog.Builder run\_dialog = **new** AlertDialog.Builder(**this**);

run\_dialog.setTitle(R.string.***mode***).setCancelable(**false**);

// добавляем переключатели

run\_dialog.setSingleChoiceItems(m\_сhoose, 0,

**new** DialogInterface.OnClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog, **int** item) {

mode\_id = item;

}

});

run\_dialog.setPositiveButton(android.R.string.***yes***,

**new** DialogInterface.OnClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog, **int** which) {

// **TODO** Auto-generated method stub

// инициализируем будущий переход между активити

run\_activity = **new** Intent(MainActivity.**this**,

RunActivity.**class**);

// передаём значение переменных в другое активити

run\_activity.putExtra("mem\_size", mem\_size);

run\_activity.putExtra("mode\_id", mode\_id);

run\_activity.putExtra("path", path);

// запускаем активити

startActivity(run\_activity);

}

});

// добавляем одну кнопку для закрытия диалога

run\_dialog.setNegativeButton(android.R.string.***cancel***,

**new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** id) {

dialog.cancel();

}

});

run\_dialog.show();

}

}

Класс Assembler:

**package** com.worm2fed.kursach\_prog;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** Assembler {

**private** String path, name, mes, listing\_name, object\_code\_name;

**public** Assembler(String file\_path) {

mes = "";

path = "";

// переменная хранит разбитую строку

String[] nm = file\_path.split("/");

// записываем имя файла

name = nm[nm.length - 1];

// формируем путь к файлу (без его имени)

**for** (**int** i = 0; i < (nm.length - 1); i++)

path = path + nm[i] + "/";

// имя для файла листинга

listing\_name = "listing.w2f";

// имя для файла с объектным кодом

object\_code\_name = "object\_code.w2f";

}

// функция проверки расширения

**private** **boolean** checkName(String name) {

// перменная хранит расширение файла

String[] check = name.split("\\.");

// если файл имеет не подходяещее разрешение - то "давай досвидания" ;D

**if** (check[check.length - 1].equals("txt")

|| check[check.length - 1].equals("w2f"))

**return** **true**;

**else**

**return** **false**;

}

// функция для преобразования операции из мнемокода в двоичный код

**private** String mnemoCodeToBinaryCode(String mnemo) {

String binary\_comand;

// поиск соответствия

**switch** (mnemo) {

**case** "LOAD":

binary\_comand = "000";

**break**;

**case** "STORE":

binary\_comand = "001";

**break**;

**case** "ADD":

binary\_comand = "010";

**break**;

**case** "SUB":

binary\_comand = "011";

**break**;

**case** "AND":

binary\_comand = "100";

**break**;

**case** "JO":

binary\_comand = "101";

**break**;

**case** "REV":

binary\_comand = "110";

**break**;

**case** "DUP":

binary\_comand = "111";

**break**;

**default**:

binary\_comand = "err";

}

**return** binary\_comand;

}

// функция проверки корректнсти операндов

**private** **boolean** checkAreOperandsCorect(String operand[], **int** num,

String mnemo) {

**byte** data = 0;

// если один операнд

**if** (num == 1) {

**if** (operand[0].equals("reg1") == **false**

&& operand[0].equals("reg2") == **false**)

**return** **false**;

**else** **if** (!mnemo.equals("STORE") && !mnemo.equals("LOAD"))

**return** **false**;

**else**

**return** **true**;

}

// если два операнда

**else** **if** (num == 2) {

**if** (!operand[0].equals("reg0"))

**return** **false**;

**else** **if** (!mnemo.equals("STORE"))

**return** **false**;

**else** {

// преобразуем второй операнд в число

**try** {

data = Byte.*parseByte*(operand[1], 16);

} **catch** (Exception e) {

**return** **false**;

}

// проверим диапазон

**if** (data > 127 || data < -127)

**return** **false**;

**else**

**return** **true**;

}

}

// если не 1 и не 2

**else**

**return** **false**;

}

// функция подготовки данных для файла листинга

**private** String prepareDataForListingFile(String data) {

String mnemo, listing\_line;

String[] operands = **new** String[2];

**int** operands\_num = 0;

// разбиваем строку по пробелам

String[] data\_split = data.split(" ");

// выделяем первое слово, которое должно содержать команду

mnemo = data\_split[0];

// переменная хранит двоичный код команды

String binary\_comand = mnemoCodeToBinaryCode(mnemo);

// проверка команды на корректность

**if** (binary\_comand.equals("err"))

listing\_line = data + ";" + " > UNKNOWN COMAND";

**else** {

// проверяем наличие операндов, и если они есть - выделяем их

**if** (data\_split.length == 1)

listing\_line = binary\_comand + " " + mnemo;

**else** {

**for** (**int** i = 1; i < data\_split.length; i++) {

operands\_num = i;

**if** (i >= 3)

listing\_line = data + ";" + " > INCORECT OPERANDS NUM";

**else**

operands[i - 1] = data\_split[i];

}

**if** (checkAreOperandsCorect(operands, operands\_num, mnemo) == **false**)

listing\_line = data + ";" + " > UNKNOWN OPERANDS";

**else** {

**if** (operands\_num == 1)

listing\_line = binary\_comand + "\_" + operands[0] + " "

+ mnemo;

**else**

listing\_line = binary\_comand + "\_" + operands[0] + "\_"

+ operands[1] + " " + mnemo;

}

}

}

**return** listing\_line;

}

// функция создания файла листинга

**private** **boolean** makeListingFile(String data[]) {

String listing\_line = **null**;

BufferedWriter bf\_writer = **null**;

**try** {

// инициализируем запись файла

FileWriter file\_writer = **new** FileWriter(path + listing\_name);

bf\_writer = **new** BufferedWriter(file\_writer);

**for** (**int** i = 0; data[i] != **null**; i++) {

// подготавливаем строку для записи

listing\_line = prepareDataForListingFile(data[i]);

bf\_writer.write(listing\_line);

bf\_writer.newLine();

**if** (listing\_line.contains(">"))

**break**;

}

} **catch** (IOException e) {

**return** **false**;

}

closeFile(bf\_writer);

// если строка содержит характерный символ ошбики - возращаем ошибку

**if** (listing\_line.contains(">"))

**return** **false**;

**else**

**return** **true**;

}

// функция подготовки данных для файла с объектным кодом

**private** String prepareDataForObjectCodeFile(String data) {

String mnemo, object\_code\_line;

String operands[] = **new** String[2];

String op = **null**;

**int** operands\_num = 0;

// разбиваем строку по пробелам

String data\_split[] = data.split(" ");

// выделяем первое слово, которое должно содержать команду

mnemo = data\_split[0];

// переменная хранит двоичный код команды

String binary\_comand = mnemoCodeToBinaryCode(mnemo);

// проверяем наличие операндов, и если они есть - выделяем их

**if** (data\_split.length == 1)

object\_code\_line = "00000" + binary\_comand;

**else** {

**for** (**int** i = 1; i < data\_split.length; i++) {

operands\_num = i;

operands[i - 1] = data\_split[i];

}

**if** (operands\_num == 1) {

**if** (operands[0].equals("reg1"))

op = "01";

**else** **if** (operands[0].equals("reg2"))

op = "10";

**else**

op = "11";

object\_code\_line = "00" + op + "0" + binary\_comand;

} **else** {

// получаем наши данные в двоичном виде

op = Integer.*toBinaryString*(Integer.*parseInt*(operands[1], 16));

// дополняем нулями недостающие разряды

**while** (op.length() != 8)

op = "0" + op;

object\_code\_line = "00" + op + "001" + binary\_comand;

}

}

**return** object\_code\_line;

}

// функция преобразования исходного файла в объектный код

**private** **boolean** makeObjectCodeFile(String[] data) {

String object\_code\_line = **null**;

BufferedWriter bf\_writer = **null**;

**try** {

FileWriter file\_writer = **new** FileWriter(path + object\_code\_name);

bf\_writer = **new** BufferedWriter(file\_writer);

**for** (**int** i = 0; data[i] != **null**; i++) {

object\_code\_line = prepareDataForObjectCodeFile(data[i]);

bf\_writer.write(object\_code\_line);

bf\_writer.newLine();

}

} **catch** (IOException e) {

**return** **false**;

}

closeFile(bf\_writer);

**return** **true**;

}

// функция для закрытия потока записи

**private** **boolean** closeFile(BufferedWriter bf\_writer) {

**try** {

**if** (bf\_writer != **null**)

bf\_writer.close();

} **catch** (IOException e) {

**return** **false**;

}

**return** **true**;

}

// функция для закрытия потока чтения

**private** **boolean** closeFile(BufferedReader bf\_reader) {

**try** {

**if** (bf\_reader != **null**)

bf\_reader.close();

} **catch** (IOException e) {

**return** **false**;

}

**return** **true**;

}

// основная функция класса, непосредственно выполняющая ассемблирование

**public** String toAssembl() {

String line = **null**;

**int** i = 0;

String[] data\_for\_file = **new** String[255];

BufferedReader bf\_reader = **null**;

// проверка расширения файла

**if** (checkName(name) == **false**)

mes = "ext\_err";

**else** {

**try** {

// открываем поток для чтения

FileReader file\_reader = **new** FileReader(path + name);

bf\_reader = **new** BufferedReader(file\_reader);

// читаем содержимое

**while** ((line = bf\_reader.readLine()) != **null**) {

// убираем комментарии

**if** (!line.contains("") && !line.contains(";")) {

mes = "lis\_err";

**return** mes;

} **else** {

String line\_done[] = line.split(";");

// если строка пуста - пропускаем её

**if** (line\_done[0].equals(""))

**continue**;

**else**

// тогда зановсим в массив наши данные попутно

// удаляя

// лишние пробелы

data\_for\_file[i] = line\_done[0].replaceAll(

"[\\s]{2,}", " ");

i++;

}

}

**if** (makeListingFile(data\_for\_file) == **false**)

mes = "lis\_err";

**else** {

**if** (makeObjectCodeFile(data\_for\_file) == **false**)

mes = "obj\_err";

**else**

mes = "suc";

}

} **catch** (FileNotFoundException e) {

mes = "404\_err";

} **catch** (IOException e) {

mes = "read\_err";

} **finally** {

**if** (closeFile(bf\_reader) == **false**)

mes = "close\_err";

}

}

**return** mes;

}

}

Класс AssemblerResultActivity:

**package** com.worm2fed.kursach\_prog;

**import** java.io.BufferedReader;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** android.app.Activity;

**import** android.os.Bundle;

**import** android.widget.TextView;

**import** android.widget.Toast;

**public** **class** AssemblerResultActivity **extends** Activity {

**protected** **void** onCreate(Bundle savedInstanceState) {

// создаём активити

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_assembler\_result***);

String file = getIntent().getExtras().getString("result\_type");

String line = **null**, mes = **null**, text = "";

BufferedReader bf\_reader = **null**;

**try** {

// открываем поток для чтения

FileReader file\_reader = **new** FileReader(file);

bf\_reader = **new** BufferedReader(file\_reader);

// читаем содержимое

**while** ((line = bf\_reader.readLine()) != **null**) {

text = text + line + "\n";

}

} **catch** (FileNotFoundException e) {

mes = "File " + file + " not found!";

Toast.*makeText*(getApplicationContext(), mes, Toast.***LENGTH\_SHORT***)

.show();

} **catch** (IOException e) {

mes = "Error due reading file " + file + "!";

Toast.*makeText*(getApplicationContext(), mes, Toast.***LENGTH\_SHORT***)

.show();

} **finally** {

**if** (closeFile(bf\_reader) == **false**) {

mes = "Error due closing file " + file + "!";

Toast.*makeText*(getApplicationContext(), mes, Toast.***LENGTH\_SHORT***)

.show();

}

}

TextView result = (TextView) findViewById(R.id.***asm\_result***);

result.setText(text);

}

// функция для закрытия потока чтения

**private** **boolean** closeFile(BufferedReader bf\_reader) {

**try** {

**if** (bf\_reader != **null**)

bf\_reader.close();

} **catch** (IOException e) {

**return** **false**;

}

**return** **true**;

}

}

Класс RunActivity:

**package** com.worm2fed.kursach\_prog;

**import** android.app.Activity;

**import** android.app.AlertDialog;

**import** android.content.DialogInterface;

**import** android.os.Bundle;

**import** android.view.View;

**import** android.widget.ArrayAdapter;

**import** android.widget.ListView;

**import** android.widget.TextView;

**import** android.widget.Toast;

**public** **class** RunActivity **extends** Activity {

**private** String mes = **null**, path;

**private** **int** mem\_size, mode\_id;

**private** ALU alu;

TextView reg1, reg2, reg\_flag, mem\_left, last\_ten\_data;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_run***);

// получаем данные из основного активити

mem\_size = getIntent().getExtras().getInt("mem\_size");

mode\_id = getIntent().getExtras().getInt("mode\_id");

path = getIntent().getExtras().getString("path");

// инициализируем наши текстовые поля

reg1 = (TextView) findViewById(R.id.***reg1***);

reg2 = (TextView) findViewById(R.id.***reg2***);

reg\_flag = (TextView) findViewById(R.id.***reg\_flag***);

mem\_left = (TextView) findViewById(R.id.***mem\_left***);

last\_ten\_data = (TextView) findViewById(R.id.***last\_ten\_data***);

// создаём АЛУ

alu = **new** ALU(mem\_size, path);

}

**public** String[] getLeftData() {

**byte** left\_data[] = alu.returnMemory().showLeftData(

alu.returnMemorySizeValue());

String list\_arr[] = **new** String[left\_data.length];

**for** (**int** i = 0; i < left\_data.length; i++)

list\_arr[i] = i + ": " + left\_data[i];

**return** list\_arr;

}

// обработчик нажатия на кнопку 'показать оставшиеся данные'

**public** **void** onShowMemLeftClick(View view) {

AlertDialog.Builder mem\_dialog = **new** AlertDialog.Builder(**this**);

mem\_dialog.setTitle(R.string.***mem\_left***).setCancelable(**false**);

// добавляем список с данными

ListView mem\_list = **new** ListView(**this**);

String[] list\_arr = getLeftData();

ArrayAdapter<String> list\_adap = **new** ArrayAdapter<String>(**this**,

android.R.layout.***simple\_list\_item\_1***, list\_arr);

mem\_list.setAdapter(list\_adap);

mem\_dialog.setView(mem\_list);

// добавляем одну кнопку для закрытия диалога

mem\_dialog.setNegativeButton(android.R.string.***cancel***,

**new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** id) {

dialog.cancel();

}

});

mem\_dialog.show();

}

**public** **void** onRunAluClick(View view) {

// непосрдетсвенно выполнение

**switch** (mode\_id) {

// автоматический режим

**case** 0:

mes = alu.runAutoMode();

**break**;

// ручной режим

**case** 1:

mes = alu.runManualMode();

**break**;

// ошибка

**default**:

mes = getApplicationContext().getResources().getString(

R.string.***mode\_error***);

**break**;

}

// выводим соответствующие сообщения

**switch** (mes) {

**case** "prepare\_err":

mes = getApplicationContext().getResources().getString(

R.string.***prepare\_error***);

**break**;

**case** "jo":

mes = getApplicationContext().getResources().getString(

R.string.***jo\_err***);

**break**;

**case** "ex\_err":

mes = getApplicationContext().getResources().getString(

R.string.***ex\_error***);

**break**;

**case** "suc":

mes = getApplicationContext().getResources().getString(

R.string.***success\_run***);

**break**;

**default**:

mes = **null**;

**break**;

}

**if** (mes != **null**)

Toast.*makeText*(getApplicationContext(), mes, Toast.***LENGTH\_SHORT***)

.show();

reg1.setText(Byte.*toString*(alu.returnReg1Value()));

reg2.setText(Byte.*toString*(alu.returnReg2Value()));

**switch** (alu.returnRegFlagValue()) {

// overflow

**case** 0x01:

reg\_flag.setText("overflow");

**break**;

// zero

**case** 0x08:

reg\_flag.setText("zero");

**break**;

// negative

**case** 0x04:

reg\_flag.setText("negative");

**break**;

// positive

**case** 0x02:

reg\_flag.setText("positive");

**break**;

**default**:

reg\_flag.setText(Integer.*toHexString*(alu.returnRegFlagValue()));

**break**;

}

mem\_left.setText(Integer.*toString*(alu.returnMemorySizeValue()));

String[] data = getLeftData();

String text = "";

**if** (data.length > 10)

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

text = text + data[i] + "\n";

**else**

**for** (**int** i = 0; i < data.length; i++)

text = text + data[i] + "\n";

last\_ten\_data.setText(text);

}

}

Класс Alu:

**package** com.worm2fed.kursach\_prog;

**import** java.io.BufferedReader;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** ALU {

// путь к объектному файлу

**private** String path;

// создаём эмулируемую память

**private** Stack memory;

// создаём регистры общего назначения и регистр флагов

**private** **byte** reg1, reg2, reg\_flag;

// массив, хранящий строки объектного кода

**private** String[] data\_for\_run;

// переменная для ручного режима

**private** **int** count = 0;

**public** ALU(**int** mem\_size, String file) {

// создаём эмулируемую память, размером mem\_size байт

memory = **new** Stack(mem\_size);

path = file + "object\_code.w2f";

}

// функция для закрытия потока чтения

**private** **boolean** closeFile(BufferedReader bf\_reader) {

**try** {

**if** (bf\_reader != **null**)

bf\_reader.close();

} **catch** (IOException e) {

**return** **false**;

}

**return** **true**;

}

// подгатавливаем данные для выполнения

**private** **boolean** prepareDataFormFile() {

String line = **null**;

**int** i = 0;

data\_for\_run = **new** String[255];

BufferedReader bf\_reader = **null**;

**try** {

// открываем поток для чтения

FileReader file\_reader = **new** FileReader(path);

bf\_reader = **new** BufferedReader(file\_reader);

// читаем содержимое

**while** ((line = bf\_reader.readLine()) != **null**) {

// если строка пуста - пропускаем её

**if** (line.equals(""))

**continue**;

// тогда зановсим в массив наши данные

**else**

data\_for\_run[i] = line;

i++;

}

} **catch** (FileNotFoundException e) {

**return** **false**;

} **catch** (IOException e) {

**return** **false**;

} **finally** {

**if** (closeFile(bf\_reader) != **false**)

**return** **true**;

}

**return** **false**;

}

**private** **short** getComandFromString(String string) {

**short** com;

**try** {

com = Short.*parseShort*(string, 2);

// выделяем три бита, отвечающие за команду, остальные обнуляем

com = (**short**) (com & 0x0007);

} **catch** (NumberFormatException e) {

com = -1;

}

**return** com;

}

// функция для установки флагов

**private** **void** setFlags(**short** data) {

/\*

\* XXXX ZNPO X - doesn't matter Z - zero N - negative P - positive O -

\* overflow

\*/

// overflow

**if** (data > 127 || data < -127)

reg\_flag = 0x01;

// zero

**else** **if** (data == 0)

reg\_flag = 0x08;

// negative

**else** **if** (data < 0)

reg\_flag = 0x04;

// positive

**else**

reg\_flag = 0x02;

}

// функция для команды

**private** **boolean** comandRead(String raw\_comand) {

**byte** reg\_num;

// извлекаем данные из памяти

**byte** data = memory.pop();

// проверяем извлеклись ли данные

**if** (data == -128)

**return** **false**;

**else** {

/\*

\* XXRR XXXX X - doesn't matter RR - register code 0x20 - reg1 0x10

\* - reg2 0x30 - err

\*/

**try** {

reg\_num = Byte.*parseByte*(raw\_comand, 2);

} **catch** (NumberFormatException e) {

**return** **false**;

}

reg\_num = (**byte**) (reg\_num & 0x30);

// проверяем в какой регистр нужно записать данные

**if** (reg\_num == 0x10)

reg1 = data;

**else** **if** (reg\_num == 0x20)

reg2 = data;

**else**

**return** **false**;

**return** **true**;

}

}

// функция для команды

**private** **boolean** comandWrite(String raw\_comand) {

// если длина команды 8 символов

**if** (raw\_comand.length() == 8) {

**byte** com = Byte.*parseByte*(raw\_comand, 2);

// получаем бит t

**byte** t = (**byte**) (com & 0x08);

// если бит t не равен 0 - ошибка

**if** (t != 0)

**return** **false**;

**else** {

// получаем номер регистра

**byte** reg\_num = (**byte**) (com & 0x30);

// записываем данные

**if** (reg\_num == 0x10) {

// проверка на успешность записи в память

**if** (!memory.push(reg1))

**return** **false**;

reg1 = 0;

} **else** **if** (reg\_num == 0x20) {

// проверка на успешность записи в память

**if** (!memory.push(reg2))

**return** **false**;

reg2 = 0;

} **else**

**return** **false**;

}

}

// если длина команды 16 символов

**else** **if** (raw\_comand.length() == 16) {

**short** com = Short.*parseShort*(raw\_comand, 2);

// получаем бит t

**short** t = (**short**) (com & 0x0008);

// если бит t не равен 1 - ошибка

**if** (t != 8)

**return** **false**;

**else** {

// получаем номер регистра

**short** reg\_num = (**short**) (com & 0x0030);

**if** (reg\_num != 0)

**return** **false**;

**else** {

**byte** data = (**byte**) (com >> 6);

// проверка на успешность записи в память

**if** (!memory.push(data))

**return** **false**;

}

}

}

// если не 8 и не 16 - ошибка

**else**

**return** **false**;

**return** **true**;

}

// функция для команды

**private** **boolean** comandAdd() {

// сначала записываем результат в переменную типа short для проверки на

// переполнение

**short** res = (**short**) (reg1 + reg2);

// записываем результат в регистр1

reg1 = (**byte**) res;

reg2 = 0;

// устанавливаем флаги

setFlags(res);

**return** **true**;

}

// функция для команды

**private** **boolean** comandSup() {

// сначала записываем результат в переменную типа short для проверки на

// переполнение

**short** res = (**short**) (reg1 - reg2);

// записываем результат в регистр1

reg1 = (**byte**) res;

reg2 = 0;

// устанавливаем флаги

setFlags(res);

**return** **true**;

}

// функция для команды

**private** **boolean** comandAnd() {

// сначала записываем результат в переменную типа short для проверки на

// переполнение

**short** res = (**short**) (reg1 & reg2);

// записываем результат в регистр1

reg1 = (**byte**) res;

reg2 = 0;

// устанавливаем флаги

setFlags(res);

**return** **true**;

}

// функция для команды

**private** **boolean** comandJump() {

// если переподнение - возвращаем неудачное выполнение

**if** (reg\_flag == 0x01)

**return** **false**;

**else**

**return** **true**;

}

// функция для команды REV

**private** **boolean** comandRev() {

**byte** data1 = memory.pop();

**byte** data2 = memory.pop();

// проверяем извлеклись ли данные

**if** (data1 == -128)

**return** **false**;

**else** **if** (data2 == -128) {

// если не извлеклось второе значение - возвращаем первое наместо

memory.push(data1);

**return** **false**;

}

// записываем дублированное значение

**else** **if** (!memory.push(data1))

**return** **false**;

**else** **if** (!memory.push(data2))

**return** **false**;

**else**

**return** **true**;

}

// функция для команды DUP

**private** **boolean** comandDup() {

// получаем данные из памяти

**byte** data = memory.pop();

// если чтение вернуло ошибку, то выход

**if** (data == -128)

**return** **false**;

// возвращаем обратно в память

**else** **if** (memory.push(data) == **false**)

**return** **false**;

// записываем дублированное значение

**else** **if** (memory.push(data) == **false**)

**return** **false**;

**else**

**return** **true**;

}

**private** **boolean** executeComand(**int** index) {

**byte** comand = (**byte**) getComandFromString(data\_for\_run[index]);

**boolean** status = **false**;

// проверяем успешность выполнения получения команды

**if** (comand == -1)

status = **false**;

**else** {

// в зависимости от полученного кода выполняем соответствующую

// операцию

**switch** (comand) {

// чтение

**case** 0x00:

status = comandRead(data\_for\_run[index]);

**break**;

// запись

**case** 0x01:

status = comandWrite(data\_for\_run[index]);

**break**;

// сложение

**case** 0x02:

status = comandAdd();

**break**;

// вычитание

**case** 0x03:

status = comandSup();

**break**;

// логическое И

**case** 0x04:

status = comandAnd();

**break**;

// переход

**case** 0x05:

status = comandJump();

**break**;

// реверсировать

**case** 0x06:

status = comandRev();

**break**;

// дублировать

**case** 0x07:

status = comandDup();

**break**;

// если соответсвий нет - ошибка

**default**:

status = **false**;

}

}

**return** status;

}

**public** String runAutoMode() {

// **TODO** Auto-generated method stub

String status = "suc";

**if** (prepareDataFormFile() == **false**)

status = "prepare\_err";

**else** {

**for** (**int** i = 0; data\_for\_run[i] != **null**; i++)

**if** (!executeComand(i)) {

**if** (reg\_flag == 0x01)

status = "jo";

**else**

status = "ex\_err";

**break**;

}

}

**return** status;

}

**public** String runManualMode() {

// **TODO** Auto-generated method stub

String status = "";

**if** (prepareDataFormFile() == **false**)

status = "prepare\_err";

**else** **if** (data\_for\_run[count] == **null**)

status = "suc";

**else** {

**if** (!executeComand(count)) {

**if** (reg\_flag == 0x01)

status = "jo";

**else**

status = "ex\_err";

}

count++;

}

**return** status;

}

**public** **byte** returnReg1Value() {

**return** reg1;

}

**public** **byte** returnReg2Value() {

**return** reg2;

}

**public** **byte** returnRegFlagValue() {

**return** reg\_flag;

}

**public** **int** returnMemorySizeValue() {

**return** memory.returnStackSizeValue();

}

**public** Stack returnMemory() {

**return** memory;

}

}

Класс Stack:

**package** com.worm2fed.kursach\_prog;

**public** **class** Stack {

// переменная для будущего стека

**private** **byte**[] stack;

// переменная для размера стека

**private** **int** size = 0;

// стек по умолчанию

**public** Stack() {

// создаём стек размером в 16 ячеек.

**this**(16);

}

**public** Stack(**int** n) {

// создаём стек размером в n ячеек.

stack = **new** **byte**[n];

}

// функция проверки стека для записи (false - полный (записывать некуда),

// true - место есть)

**private** **boolean** checkPush(**byte**[] st) {

**if** (size == st.length)

**return** **false**;

**else**

**return** **true**;

}

// функция для записи данных в стек

**public** **boolean** push(**byte** data) {

**if** (!checkPush(stack))

**return** **false**;

**else** {

**if** (size != 0)

**for** (**int** i = (size - 1); i >= 0; i--)

stack[i + 1] = stack[i];

stack[0] = data;

size++;

**return** **true**;

}

}

// функция проверки стека для извлечения (false - пустой (извлекать нечего),

// true - данные есть)

**private** **boolean** checkPop(**byte**[] st) {

**if** (size == 0)

**return** **false**;

**else**

**return** **true**;

}

// функция для извлечения данных из стека

**public** **byte** pop() {

**if** (!checkPop(stack))

**return** (**byte**) -128;

**else** {

**byte** data = stack[0];

**for** (**int** i = 0; i < (size - 1); i++)

stack[i] = stack[i + 1];

size--;

**return** data;

}

}

**public** **int** returnStackSizeValue() {

**return** size;

}

**public** **byte**[] showLeftData(**int** size) {

// **TODO** Auto-generated method stub

**byte**[] left\_mem = **new** **byte**[size];

**for** (**int** i = 0; i < size; i++)

left\_mem[i] = stack[i];

**return** left\_mem;

}

}