**CSCI 6461 Computer System Architecture   
Project Part 2 Instructions and Source Code  
Team 4**Bhavya Sree Gudiseva, Dinesh Reddy Reddem, John Wilkie, Pramukh Nagol

**Filename:** Team4\_ProjectPart2.zip

**Execution Instructions:**

1. Download the Team4\_ProjectPart2.zip file.
2. Extract Zip file and make sure all the below files are present before you run the jar file.

Graphical user interface, text, application

Description automatically generated

1. Double click on ProjectPart2.jar to open the jar file.

**Source Code:**

**Constants**

public class Constants {

public static final int UNSIGNED\_MAX = 65535;

public static final int SIGNED\_MAX = 32767;

public static final int SIGNED\_MIN = -32767;

public static final float FLOAT\_MAX = (float) (2 - Math.pow(2, -7) \* Math.pow(2, 256 - 1)) ;

public static final float FLOAT\_MIN = -(float) (2 - Math.pow(2, -7) \* Math.pow(2, 256 - 1));

public static final int CC\_OVERFLOW = 0;

public static final int CC\_UNDERFLOW = 1;

public static final int CC\_DIVZERO = 2;

public static final int CC\_EQUALORNOT = 3;

}

**MainFrame**

package csci6461;

import java.awt.EventQueue;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.border.EmptyBorder;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.BitSet;

// import java.util.Scanner;

import java.awt.event.ActionEvent;

import javax.swing.SwingConstants;

import java.awt.Color;

public class MainFrame extends JFrame {

// Initialise memory

private static Simulator simulator = Simulator.getInstance();

private static Memory memory = Memory.getInstance();

static int mar;

static int mbr;

static int pc = 10;

static int ir;

static String ir\_binary;

static byte OPCODE;

static byte R;

static byte IX;

static byte I;

static int ADDR;

static int pcx;

private JPanel contentPane;

private static JTextField textField\_GPR0;

private static JTextField textField\_GPR1;

private static JTextField textField\_GPR2;

private static JTextField textField\_input;

private static JTextField textField\_GPR3;

private static JTextField textField\_IXR1;

private static JTextField textField\_IXR2;

private static JTextField textField\_IXR3;

private static JTextField textField\_PC;

private static JTextField textField\_MAR;

private static JTextField textField\_MBR;

private static JTextField textField\_IR;

private static JTextField textField\_MFR;

private static JTextField textField\_Priviledged;

private static JTextField textField\_CC;

private static JTextArea textField\_Keyboard;

private static JTextArea textField\_Printer;

private static JTextArea textField\_Tag;

private static JTextArea textField\_TagValue;

private JTextField textField\_FR0;

private JTextField textField\_FR1;

/\*

\* Method to get the text field object for respective registers

\*/

private static JTextField getRegisterGUI(String regStr) {

if (regStr == "R0") {

return textField\_GPR0;

}

if (regStr == "R1") {

return textField\_GPR1;

}

if (regStr == "R2") {

return textField\_GPR2;

}

if (regStr == "R3") {

return textField\_GPR3;

}

if (regStr == "X1") {

return textField\_IXR1;

}

if (regStr == "X2") {

return textField\_IXR2;

}

if (regStr == "X3") {

return textField\_IXR3;

}

if (regStr == "PC") {

return textField\_PC;

}

if (regStr == "MAR") {

return textField\_MAR;

}

if (regStr == "MBR") {

return textField\_MBR;

}

if (regStr == "IR") {

return textField\_IR;

}

if (regStr == "CC") {

return textField\_CC;

}

return null;

}

/\*\*

\* Method to update Registers display on UI

\*

\* @param regStr

\* @param value

\* @param length

\*/

public static void updateRegUI(String regStr, BitSet value, int length) {

JTextField reg = getRegisterGUI(regStr);

// no update if no corresponding GUI element

if (reg == null) {

return;

}

String result = Integer.toBinaryString(Util.bitSet2Int(value));

int zeros = length - result.length();

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

result = "0" + result;

}

reg.setText(result);

}

/\*\*

\* Method to update Cache on UI

\*

\* @param tag

\* @param value

\*/

public static void updateCacheUI(ArrayList<Integer> tag, ArrayList<Word> value) {

String strTag = "";

String strVal = "";

for (int i = 0; i < tag.size(); i++) {

strTag += tag.get(i).toString() + '\n';

strVal += Util.bitSet2Int(value.get(i)) + "\n";

}

textField\_Tag.setText(strTag);

textField\_TagValue.setText(strVal);

}

/\*\*

\* Method to Get input from UI keyboard

\* @return

\*/

public static int getKeyboard() {

String text = textField\_Keyboard.getText();

String token = text.split(",")[0]; // get first element

int result;

try {

result = Integer.parseInt(token);

} catch (NumberFormatException n) {

result = Integer.parseInt(String.valueOf(token.charAt(0)));

}

if (text.indexOf(",") != -1)

text = text.substring(text.indexOf(",") + 1);

else text = "";

textField\_Keyboard.setText(text);

return result;

}

/\*\*

\* Method to set printer on UI using integer output

\* @param output

\*/

public static void setPrinter(int output) {

textField\_Printer.setText(textField\_Printer.getText() + output);

}

/\*\*

\* Method to set printer on UI using string output

\* @param output

\*/

public static void setPrinter(String output) {

textField\_Printer.setText(textField\_Printer.getText() + output);

}

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

// System.out.println(pc);

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

MainFrame frame = new MainFrame();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the frame.

\*/

public MainFrame() {

setTitle("Simulator UI");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 1256, 764);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

JLabel lblNewLabel = new JLabel("GPR 0");

lblNewLabel.setBounds(25, 32, 63, 24);

contentPane.add(lblNewLabel);

JLabel lblNewLabel\_1 = new JLabel("GPR 1");

lblNewLabel\_1.setBounds(25, 82, 45, 13);

contentPane.add(lblNewLabel\_1);

JLabel lblNewLabel\_2 = new JLabel("GPR 2");

lblNewLabel\_2.setBounds(25, 118, 45, 13);

contentPane.add(lblNewLabel\_2);

textField\_GPR0 = new JTextField();

textField\_GPR0.setEditable(false);

textField\_GPR0.setBounds(136, 35, 245, 19);

textField\_GPR0.setText("0000000000000000");

contentPane.add(textField\_GPR0);

textField\_GPR0.setColumns(10);

textField\_GPR1 = new JTextField();

textField\_GPR1.setEditable(false);

textField\_GPR1.setBounds(136, 79, 245, 19);

textField\_GPR1.setText("0000000000000000");

contentPane.add(textField\_GPR1);

textField\_GPR1.setColumns(10);

textField\_GPR2 = new JTextField();

textField\_GPR2.setEditable(false);

textField\_GPR2.setBounds(136, 115, 245, 19);

textField\_GPR2.setText("0000000000000000");

contentPane.add(textField\_GPR2);

textField\_GPR2.setColumns(10);

JButton GPR0\_load = new JButton("LD");

GPR0\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("R0", textField\_input.getText());

}

});

GPR0\_load.setBounds(391, 35, 85, 21);

contentPane.add(GPR0\_load);

JButton GPR1\_load = new JButton("LD");

GPR1\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("R1", textField\_input.getText());

}

});

GPR1\_load.setBounds(391, 79, 85, 21);

contentPane.add(GPR1\_load);

JButton GPR2\_load = new JButton("LD");

GPR2\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("R2", textField\_input.getText());

}

});

GPR2\_load.setBounds(391, 115, 85, 21);

contentPane.add(GPR2\_load);

textField\_input = new JTextField();

textField\_input.setBounds(90, 410, 358, 19);

contentPane.add(textField\_input);

textField\_input.setColumns(10);

JLabel lblNewLabel\_3 = new JLabel("Input");

lblNewLabel\_3.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_3.setBounds(90, 437, 358, 13);

contentPane.add(lblNewLabel\_3);

JLabel lblNewLabel\_2\_1 = new JLabel("GPR 3");

lblNewLabel\_2\_1.setBounds(25, 155, 45, 13);

contentPane.add(lblNewLabel\_2\_1);

textField\_GPR3 = new JTextField();

textField\_GPR3.setEditable(false);

textField\_GPR3.setColumns(10);

textField\_GPR3.setBounds(136, 152, 245, 19);

textField\_GPR3.setText("0000000000000000");

contentPane.add(textField\_GPR3);

JButton GPR3\_load = new JButton("LD");

GPR3\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("R3", textField\_input.getText());

}

});

GPR3\_load.setBounds(391, 152, 85, 21);

contentPane.add(GPR3\_load);

JLabel lblNewLabel\_2\_1\_1 = new JLabel("IXR 1");

lblNewLabel\_2\_1\_1.setBounds(25, 196, 45, 13);

contentPane.add(lblNewLabel\_2\_1\_1);

textField\_IXR1 = new JTextField();

textField\_IXR1.setEditable(false);

textField\_IXR1.setColumns(10);

textField\_IXR1.setBounds(136, 193, 245, 19);

textField\_IXR1.setText("0000000000000000");

contentPane.add(textField\_IXR1);

JButton IXR1\_load = new JButton("LD");

IXR1\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("X1", textField\_input.getText());

}

});

IXR1\_load.setBounds(391, 193, 85, 21);

contentPane.add(IXR1\_load);

JLabel lblNewLabel\_2\_1\_1\_1 = new JLabel("IXR 2");

lblNewLabel\_2\_1\_1\_1.setBounds(25, 233, 45, 13);

contentPane.add(lblNewLabel\_2\_1\_1\_1);

textField\_IXR2 = new JTextField();

textField\_IXR2.setEditable(false);

textField\_IXR2.setColumns(10);

textField\_IXR2.setBounds(136, 230, 245, 19);

textField\_IXR2.setText("0000000000000000");

contentPane.add(textField\_IXR2);

JButton IXR2\_load = new JButton("LD");

IXR2\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("X2", textField\_input.getText());

}

});

IXR2\_load.setBounds(391, 230, 85, 21);

contentPane.add(IXR2\_load);

textField\_IXR3 = new JTextField();

textField\_IXR3.setEditable(false);

textField\_IXR3.setColumns(10);

textField\_IXR3.setBounds(136, 272, 245, 19);

textField\_IXR3.setText("0000000000000000");

contentPane.add(textField\_IXR3);

JLabel lblNewLabel\_2\_1\_1\_1\_1 = new JLabel("IXR 3");

lblNewLabel\_2\_1\_1\_1\_1.setBounds(25, 275, 45, 13);

contentPane.add(lblNewLabel\_2\_1\_1\_1\_1);

JButton IXR3\_load = new JButton("LD");

IXR3\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("X3", textField\_input.getText());

}

});

IXR3\_load.setBounds(391, 272, 85, 21);

contentPane.add(IXR3\_load);

JLabel lblPc = new JLabel("PC");

lblPc.setBounds(673, 32, 63, 24);

contentPane.add(lblPc);

textField\_PC = new JTextField();

textField\_PC.setEditable(false);

textField\_PC.setColumns(10);

textField\_PC.setBounds(784, 35, 313, 19);

textField\_PC.setText("000000000000");

contentPane.add(textField\_PC);

JButton PC\_load = new JButton("LD");

PC\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("PC", textField\_input.getText());

}

});

PC\_load.setBounds(1107, 35, 85, 21);

contentPane.add(PC\_load);

JLabel lblNewLabel\_4\_1 = new JLabel("MAR");

lblNewLabel\_4\_1.setBounds(673, 82, 63, 24);

contentPane.add(lblNewLabel\_4\_1);

textField\_MAR = new JTextField();

textField\_MAR.setEditable(false);

textField\_MAR.setColumns(10);

textField\_MAR.setBounds(784, 85, 313, 19);

textField\_MAR.setText("000000000000");

contentPane.add(textField\_MAR);

JButton MAR\_load = new JButton("LD");

MAR\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("MAR", textField\_input.getText());

}

});

MAR\_load.setBounds(1107, 85, 85, 21);

contentPane.add(MAR\_load);

JLabel lblNewLabel\_4\_1\_1 = new JLabel("MBR");

lblNewLabel\_4\_1\_1.setBounds(673, 118, 63, 24);

contentPane.add(lblNewLabel\_4\_1\_1);

textField\_MBR = new JTextField();

textField\_MBR.setEditable(false);

textField\_MBR.setColumns(10);

textField\_MBR.setBounds(784, 121, 313, 19);

textField\_MBR.setText("0000000000000000");

contentPane.add(textField\_MBR);

JButton MBR\_load = new JButton("LD");

MBR\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("MBR", textField\_input.getText());

}

});

MBR\_load.setBounds(1107, 121, 85, 21);

contentPane.add(MBR\_load);

JLabel lblNewLabel\_4\_1\_1\_1 = new JLabel("IR");

lblNewLabel\_4\_1\_1\_1.setBounds(673, 160, 63, 24);

contentPane.add(lblNewLabel\_4\_1\_1\_1);

textField\_IR = new JTextField();

textField\_IR.setEditable(false);

textField\_IR.setColumns(10);

textField\_IR.setBounds(784, 163, 313, 19);

textField\_IR.setText("0000000000000000");

contentPane.add(textField\_IR);

JButton btnNewButton\_3 = new JButton("Load");

btnNewButton\_3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.load();

}

});

btnNewButton\_3.setBounds(673, 353, 74, 37);

contentPane.add(btnNewButton\_3);

JButton btnNewButton\_4 = new JButton("Store");

btnNewButton\_4.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.store();

}

});

btnNewButton\_4.setBounds(673, 396, 74, 37);

contentPane.add(btnNewButton\_4);

JButton SS\_button = new JButton("SS");

SS\_button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.singleStep();

}

});

SS\_button.setBounds(854, 365, 94, 59);

contentPane.add(SS\_button);

JButton Run\_button = new JButton("Run");

Run\_button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.runProgram();

}

});

Run\_button.setBounds(958, 365, 94, 59);

contentPane.add(Run\_button);

JButton Init\_Button = new JButton("INIT");

Init\_Button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.init("./src/csci6461/boot.txt");

}

});

Init\_Button.setBounds(754, 376, 74, 37);

contentPane.add(Init\_Button);

JLabel lblNewLabel\_4\_1\_1\_2 = new JLabel("MFR");

lblNewLabel\_4\_1\_1\_2.setBounds(673, 196, 63, 24);

contentPane.add(lblNewLabel\_4\_1\_1\_2);

textField\_MFR = new JTextField();

textField\_MFR.setText("0000");

textField\_MFR.setEditable(false);

textField\_MFR.setColumns(10);

textField\_MFR.setBounds(784, 199, 313, 19);

contentPane.add(textField\_MFR);

JLabel lblNewLabel\_4\_1\_1\_3 = new JLabel("Priviledged");

lblNewLabel\_4\_1\_1\_3.setBounds(673, 238, 85, 24);

contentPane.add(lblNewLabel\_4\_1\_1\_3);

textField\_Priviledged = new JTextField();

textField\_Priviledged.setText("0");

textField\_Priviledged.setEditable(false);

textField\_Priviledged.setColumns(10);

textField\_Priviledged.setBounds(784, 241, 313, 19);

contentPane.add(textField\_Priviledged);

JLabel lblNewLabel\_4\_1\_1\_3\_1 = new JLabel("CC");

lblNewLabel\_4\_1\_1\_3\_1.setBounds(673, 288, 85, 24);

contentPane.add(lblNewLabel\_4\_1\_1\_3\_1);

textField\_CC = new JTextField();

textField\_CC.setText("0000");

textField\_CC.setEditable(false);

textField\_CC.setColumns(10);

textField\_CC.setBounds(784, 291, 313, 19);

contentPane.add(textField\_CC);

JButton CC\_Button = new JButton("LD");

CC\_Button.setBounds(1107, 290, 85, 21);

CC\_Button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("CC", textField\_input.getText());

}

});

contentPane.add(CC\_Button);

textField\_Keyboard = new JTextArea();

textField\_Keyboard.setLineWrap(true);

textField\_Keyboard.setBounds(673, 479, 223, 183);

contentPane.add(textField\_Keyboard);

textField\_Keyboard.setColumns(10);

JLabel lblNewLabel\_4 = new JLabel("Keyboard Console");

lblNewLabel\_4.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_4.setBounds(673, 672, 223, 13);

contentPane.add(lblNewLabel\_4);

textField\_Printer = new JTextArea();

JScrollPane scrollPane = new JScrollPane (textField\_Printer,

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

textField\_Printer.setEditable(false);

textField\_Printer.setLineWrap(true);

textField\_Printer.setColumns(10);

scrollPane.setBounds(939, 479, 223, 183);

contentPane.add(scrollPane);

JLabel lblNewLabel\_4\_2 = new JLabel("Printer Display");

lblNewLabel\_4\_2.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_4\_2.setBounds(939, 670, 223, 17);

contentPane.add(lblNewLabel\_4\_2);

textField\_Tag = new JTextArea();

JScrollPane scrollPaneTag = new JScrollPane (textField\_Tag,

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

textField\_Tag.setBackground(Color.WHITE);

textField\_Tag.setEditable(false);

textField\_Tag.setColumns(10);

scrollPaneTag.setBounds(25, 483, 223, 183);

contentPane.add(scrollPaneTag);

textField\_TagValue = new JTextArea();

JScrollPane scrollPaneTagValue = new JScrollPane (textField\_TagValue,

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

textField\_TagValue.setBackground(Color.WHITE);

textField\_TagValue.setEditable(false);

textField\_TagValue.setColumns(10);

scrollPaneTagValue.setBounds(279, 483, 223, 183);

contentPane.add(scrollPaneTagValue);

JLabel lblNewLabel\_5 = new JLabel("CACHE");

lblNewLabel\_5.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_5.setBounds(158, 676, 223, 23);

contentPane.add(lblNewLabel\_5);

JLabel lblNewLabel\_6 = new JLabel("Tag");

lblNewLabel\_6.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_6.setBounds(25, 460, 223, 13);

contentPane.add(lblNewLabel\_6);

JLabel lblNewLabel\_6\_1 = new JLabel("Value");

lblNewLabel\_6\_1.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel\_6\_1.setBounds(279, 460, 223, 13);

contentPane.add(lblNewLabel\_6\_1);

JButton Program1\_Button = new JButton("Program 1");

Program1\_Button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.init("./src/csci6461/program1.txt");

if (textField\_Keyboard.getText().isEmpty()) {

setPrinter("using default value...\n");

textField\_Keyboard.setText("17827,8255,23642,1412,870,3177,2639,231,2899,18,1109,31,88,973,100,1281,1241,713,119,240,5000");

}

String msg = "Array of 20 integers:\n";

setPrinter(msg);

int k;

// write array to memory and print in console output

for (int i = 0; i < 20; i++) {

k = getKeyboard();

memory.write(Util.int2Word(k), 800 + i + 1);

setPrinter(k);

if (i < 19) setPrinter(",");

else setPrinter("\n");

}

// write search number to memory and print in console output

k = getKeyboard();

memory.write(Util.int2Word(k), 400);

msg = "Search number: ";

setPrinter(msg);

setPrinter(k);

setPrinter("\n");

}

});

Program1\_Button.setBounds(1075, 353, 157, 21);

contentPane.add(Program1\_Button);

JLabel lblNewLabel\_2\_1\_1\_1\_1\_1 = new JLabel("FR 0");

lblNewLabel\_2\_1\_1\_1\_1\_1.setBounds(25, 315, 45, 13);

contentPane.add(lblNewLabel\_2\_1\_1\_1\_1\_1);

textField\_FR0 = new JTextField();

textField\_FR0.setText("0000000000000000");

textField\_FR0.setEditable(false);

textField\_FR0.setColumns(10);

textField\_FR0.setBounds(136, 312, 245, 19);

contentPane.add(textField\_FR0);

JButton FR0\_load = new JButton("LD");

FR0\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("X3", textField\_input.getText());

}

});

FR0\_load.setBounds(391, 312, 85, 21);

contentPane.add(FR0\_load);

JLabel lblNewLabel\_2\_1\_1\_1\_1\_2 = new JLabel("FR 1");

lblNewLabel\_2\_1\_1\_1\_1\_2.setBounds(25, 353, 45, 13);

contentPane.add(lblNewLabel\_2\_1\_1\_1\_1\_2);

textField\_FR1 = new JTextField();

textField\_FR1.setText("0000000000000000");

textField\_FR1.setEditable(false);

textField\_FR1.setColumns(10);

textField\_FR1.setBounds(136, 350, 245, 19);

contentPane.add(textField\_FR1);

JButton FR1\_load = new JButton("LD");

FR1\_load.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

simulator.loadRegisterFromInput("X3", textField\_input.getText());

}

});

FR1\_load.setBounds(391, 350, 85, 21);

contentPane.add(FR1\_load);

simulator.init("./src/csci6461/boot.txt");

}

public static void clearPrinter() {

textField\_Printer.setText("");

}

}

**Memory**

package csci6461;

import java.util.ArrayList;

public class Memory {

private static final Memory INSTANCE = new Memory();

private static Word[] memory;

private static Cache cache;

// private constructor to prevent initialization from outside the class

private Memory() {

memory = new Word[4096];

for (int i = 0; i < 4096; i++) {

memory[i] = new Word();

}

// Arrays.fill(memory, new Word());

cache = new Cache();

}

public static Memory getInstance() {

return INSTANCE;

}

public Word read(int address) {

//return memory[address];

return cache.read(address);

}

private class Cache {

public ArrayList<Integer> address;

public ArrayList<Word> content;

public int length;

public Cache() {

address = new ArrayList<>();

content = new ArrayList<>();

length = 0;

}

public void add(int addr, Word cont) {

if (length >= 16) {

address.remove(0);

content.remove(0);

length--;

}

address.add(Integer.valueOf(addr));

content.add(cont);

length++;

MainFrame.updateCacheUI(address, content);

}

public Word read(int addr) {

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

if (address.get(i).intValue() == addr) {

// HIT

content.set(i, memory[addr]);

return content.get(i);

}

}

// MISS

this.add(addr, memory[addr]);

return memory[addr];

}

}

public void write(Word inp, int address) {

memory[address] = inp;

}

public void write(int inp, int address) {

memory[address] = Util.int2Word(inp);

}

}

**Opcodes**

public class OpCodes {

public static final byte HLT = 00;

public static final byte LDR = 01;

public static final byte STR = 02;

public static final byte LDA = 03;

public static final byte LDX = 41;

public static final byte STX = 42;

public static final byte JZ = 10;

public static final byte JNE = 11;

public static final byte JCC = 12;

public static final byte JMA = 13;

public static final byte JSR = 14;

public static final byte RFS = 15;

public static final byte SOB = 16;

public static final byte JGE = 17;

public static final byte AMR = 04;

public static final byte SMR = 05;

public static final byte AIR = 06;

public static final byte SIR = 07;

public static final byte MLT = 20;

public static final byte DVD = 21;

public static final byte TRR = 22;

public static final byte AND = 23;

public static final byte ORR = 24;

public static final byte NOT = 25;

public static final byte SRC = 31;

public static final byte RRC = 32;

public static final byte IN = 61;

public static final byte OUT = 62;

}

**Register**

package csci6461;

import java.util.BitSet;

public class Register extends BitSet {

/\*\*

\*

\*/

private static final long serialVersionUID = -972511712136575293L;

private int size;

public Register(int size) {

super(size);

this.size = size;

}

public int getSize() {

return size;

}

}

**Simulator**

package csci6461;

import java.io.File;

import java.util.BitSet;

import java.util.Scanner;

/\*\*

\* Simulator class implements operations based on opcodes

\* Also processes required General, Index and float registers

\*

\* @author Dinesh Reddy

\*

\*/

public class Simulator {

private static final Simulator INSTANCE = new Simulator();

private Register R0;

private Register R1;

private Register R2;

private Register R3;

private Register X1;

private Register X2;

private Register X3;

private Register PC;

private Register IR;

private Register CC;

private Register MAR;

private Register MBR;

private Register MFR;

private Register FR0;

private Register FR1;

private byte opcode;

private byte ix;

private byte r;

private byte i;

private byte addr;

private byte ry;

private byte al;

private byte lr;

private byte count;

private byte dev\_id;

private byte trap\_code;

private File f;

// an incremental variable to count the number of instructions read

private int lines;

private Simulator() {

// initialize registers

R0 = new Register(16);

R1 = new Register(16);

R2 = new Register(16);

R3 = new Register(16);

X1 = new Register(16);

X2 = new Register(16);

X3 = new Register(16);

PC = new Register(12);

IR = new Register(16);

CC = new Register(4);

MAR = new Register(12);

MBR = new Register(16);

MFR = new Register(4);

FR0 = new Register(16);

FR1 = new Register(16);

lines = 0; // keep track of lines of instructions read from file

}

public static Simulator getInstance() {

return INSTANCE;

}

private static Memory memory = Memory.getInstance();

BitSet Str2BitSet(String s) {

assert(s.length() == 16);

BitSet b = new BitSet();

for (int i = 0; i < 16; i++) {

if (s.charAt(i) == '1') b.set(15 - i);

}

return b;

}

void BitSet2Str(BitSet bs) {

for (int i = 0 ; i < 16; i++) {

if (bs.get(15- i)) System.out.print('1');

else System.out.print('0');

}

System.out.println();

}

void test() {

float[] flt = {

5.09f,

6.39f,

3.20f,

7.92f,

2.10f,

8.29f,

1.29f,

9.39f,

4.20f,

2.10f,

3.90f,

8.29f

};

for (int i = 0; i< 12; i++) {

BitSet bs = Util.float2BitSet(flt[i]);

BitSet2Str(bs);

float f = Util.bitSet2Float(bs);

System.out.println(f);

}

}

/\*\*

\* Method to initialise Simulator

\*

\* @param path

\*/

public void init(String path) {

//test();

lines = 0;

loadFile(path);

try {

Scanner s = new Scanner(f);

while (s.hasNextLine()) {

String s1 = s.nextLine();

String[] sa = s1.split(" ");

// setting the memory

int addr = Integer.parseInt(sa[0].trim(), 16);

if (lines == 0) {

setRegister(PC, addr);

}

Word content = Util.int2Word(Integer.parseInt(sa[1].trim(), 16));

memory.write(content, addr);

lines++;

}

s.close();

} catch (Exception ex) {

System.out.println("Exception occured in input file" + ex);

}

setRegister(R0, 0);

setRegister(R1, 0);

setRegister(R2, 0);

setRegister(R3, 0);

setRegister(X1, 0);

setRegister(X2, 0);

setRegister(X3, 0);

setRegister(MAR, 0);

setRegister(MBR, 0);

setRegister(MFR, 0);

setRegister(IR, 0);

setRegister(CC, 0);

setRegister(FR0, 0);

setRegister(FR1, 0);

}

/\*\*

\* Method to implement program single step at a time.

\* It implements one instruction at a time and PC points to the next instruction.

\*

\* Once all instructions are executed, singleStep is disabled.

\* @return

\*/

public int singleStep() {

// fetch instruction

loadInstruction();

// ir decode

int ir = Util.bitSet2Int(IR);

irDecode(ir);

// operation

return getInstance().operation();

}

/\*\*

\* Runs the complete program in one click.

\* Chooses program file according to the given input file

\*/

public int runProgram() {

while (singleStep() == 0);

if (f.getName().equals("program1.txt")) {

MainFrame.setPrinter("closest number: " + Util.bitSet2Int(memory.read(202)) + "\n");

}

if (f.getName().equals("program2.txt")) {

int i\_word = Util.bitSet2Int(memory.read(30));

int i\_sentence = Util.bitSet2Int(memory.read(31));

if (i\_word == 0 || i\_sentence == 0) {

MainFrame.setPrinter("Word not found.");

}

MainFrame.setPrinter("Word found at sentence " + i\_sentence + ", word " + i\_word + "\n");

}

if (f.getName().equals("program3.txt")) {

float fadd = Util.bitSet2Float(memory.read(15));

float fsub = Util.bitSet2Float(memory.read(16));

int fixed = Util.bitSet2Int(memory.read(17));

float floating = Util.bitSet2Float(memory.read(18));

MainFrame.setPrinter("FADD: 5.078125 + 6.375 = " + fadd + "\n");

MainFrame.setPrinter("FSUB: 5.078125 - 6.375 = " + fsub + "\n");

MainFrame.setPrinter("CNVRT to fixed " + fixed + "\n");

MainFrame.setPrinter("CNVRT to floating " + floating + "\n");

// int v1[] = {153,506,12,521,3,914};

// int v2[] = {556,268,12,243,521};

// MainFrame.setPrinter("VADD");

}

return 1;

}

public void loadFile(String path) {

f = new File(path);

}

/\*\*

\* load instruction at given address

\*

\* @param input

\*/

public void loadInstruction() {

setRegister(MAR, PC);

setRegister(MBR, memory.read(Util.bitSet2Int(MAR)));

setRegister(IR, MBR);

}

/\*\*

\* Decodes given instruction according to opcode number

\*

\* @param ir

\*/

public void irDecode(int ir) {

// constructing ir as string

String ir\_binary = Integer.toBinaryString(ir);

int zeros = 16 - ir\_binary.length();

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

ir\_binary = "0" + ir\_binary;

}

opcode = (byte) Integer.parseInt(ir\_binary.substring(0, 6), 2);

if ((opcode >= 1 && opcode <= 7) || opcode == 41 || opcode == 42 || (opcode >= 10 && opcode <= 17)

|| (opcode >= 33 && opcode <= 37) || opcode == 50 || opcode == 51) {

// LD and STR, add and sub, floating point and vector operations

r = (byte) Integer.parseInt(ir\_binary.substring(6, 8), 2);

ix = (byte) Integer.parseInt(ir\_binary.substring(8, 10), 2);

i = (byte) Integer.parseInt(ir\_binary.substring(10, 11), 2);

addr = (byte) Integer.parseInt(ir\_binary.substring(11, 16), 2);

} else if (opcode >= 20 && opcode <= 25) {

// MLT and DIV

r = (byte) Integer.parseInt(ir\_binary.substring(6, 8), 2);

ry = (byte) Integer.parseInt(ir\_binary.substring(8, 10), 2);

} else if (opcode == 30) {

// trap

trap\_code = (byte) Integer.parseInt(ir\_binary.substring(12, 16), 2);

} else if (opcode == 31 || opcode == 32) {

// shift and rotate

r = (byte) Integer.parseInt(ir\_binary.substring(6, 8), 2);

al = (byte) Integer.parseInt(ir\_binary.substring(8, 9), 2);

lr = (byte) Integer.parseInt(ir\_binary.substring(9, 10), 2);

count = (byte) Integer.parseInt(ir\_binary.substring(12, 16), 2);

} else if (opcode >= 61 && opcode <= 63) {

// IN and OUT

r = (byte) Integer.parseInt(ir\_binary.substring(6, 8), 2);

dev\_id = (byte) Integer.parseInt(ir\_binary.substring(11, 16), 2);

}

}

/\*\*

\* Calculate effective address from the given indirect(i), indexing(ix), Address

\*

\* @param i

\* @param ix

\* @param address

\* @return

\*/

public int calculateEA(byte i, byte ix, byte address) {

int ea = 0; // return value

// no indirect

if (i == 0) {

if (ix == 0) {

ea = address;

return ea;

} else if (ix <= 3 && ix >= 1) {

ea = Util.bitSet2Int(getIXR(ix)) + address;

return ea;

}

}

// with indirect

if (i == 1) {

if (ix == 0) {

ea = Util.bitSet2Int(memory.read(address));

return ea;

} else if (ix <= 3 && ix >= 1) {

// variable for c(IX) + c(Address Field)

int tmpAddr = Util.bitSet2Int(getIXR(ix)) + address;

// fetch content at given address

ea = Util.bitSet2Int(memory.read(tmpAddr));

return ea;

}

}

return ea;

}

/\*\*

\* Method to map Register name to its string

\*

\* @param r

\* @return

\*/

private String regName2Str(Register r) {

if (r == R0)

return "R0";

if (r == R1)

return "R1";

if (r == R2)

return "R2";

if (r == R3)

return "R3";

if (r == X1)

return "X1";

if (r == X2)

return "X2";

if (r == X3)

return "X3";

if (r == PC)

return "PC";

if (r == MAR)

return "MAR";

if (r == MBR)

return "MBR";

if (r == IR)

return "IR";

if (r == CC)

return "CC";

if (r == FR0)

return "FR0";

if (r == FR1)

return "FR1";

return null;

}

/\*\*

\* Method to map register string to register

\*

\* @param r

\* @return

\*/

private Register regStr2Name(String r) {

if (r == "R0") {

return R0;

}

if (r == "R1") {

return R1;

}

if (r == "R2") {

return R2;

}

if (r == "R3") {

return R3;

}

if (r == "X1") {

return X1;

}

if (r == "X2") {

return X2;

}

if (r == "X3") {

return X3;

}

if (r == "PC") {

return PC;

}

if (r == "MAR") {

return MAR;

}

if (r == "MBR") {

return MBR;

}

if (r == "IR") {

return IR;

}

if (r == "CC") {

return CC;

}

if (r == "FR0") {

return FR0;

}

if (r == "FR1") {

return FR1;

}

return null;

}

/\*\*

\* Method to set register with content

\*

\* @param r

\* @param content

\*/

public void setRegister(Register r, int content) {

BitSet w = Util.int2BitSet(content);

Util.bitSetDeepCopy(w, 16, r, r.getSize());

MainFrame.updateRegUI(regName2Str(r), r, r.getSize());

}

/\*\*

\* Method to set register with bitset

\* @param r

\* @param src

\*/

public void setRegister(Register r, BitSet src) {

int srcData = Util.bitSet2Int(src);

setRegister(r, srcData);

}

/\*\*

\*

\* @param r

\* @param content

\*/

public void setRegisterSigned(Register r, int content) {

BitSet w = Util.int2BitSetSigned(content);

Util.bitSetDeepCopy(w, 16, r, r.getSize());

MainFrame.updateRegUI(regName2Str(r), r, r.getSize());

}

/\*\*

\* Method to set overflow, underflow, divzero and equalornot flags

\*

\* @param i

\* @param bit

\*/

public void setCC(int i, boolean bit) {

int bitIndex = 3 - i;

// OVERFLOW = 0

// UNDERFLOW = 1

// DIVZERO = 2

// EQUALORNOT = 3

CC.set(bitIndex, bit);

MainFrame.updateRegUI("CC", CC, 4);

}

/\*\*

\* Method to get the values of Overflow, underflow, divzero, equalornot flags from CC

\*

\* @param i

\* @return

\*/

public byte getCC(int i) {

int bitIndex = 3 - i;

// OVERFLOW = 0

// UNDERFLOW = 1

// DIVZERO = 2

// EQUALORNOT = 3

boolean r = CC.get(bitIndex);

if (r)

return 1;

else

return 0;

}

/\*\*

\* Method to get General Purpose Register from register number

\*

\* @param r

\* @return

\*/

public Register getGPR(int r) {

switch (r) {

case 0:

return R0;

case 1:

return R1;

case 2:

return R2;

case 3:

return R3;

}

return R0;

}

/\*\*

\* Method to get Index Register from index register number

\*

\* @param ix

\* @return

\*/

public Register getIXR(int ix) {

switch (ix) {

case 1:

return X1;

case 2:

return X2;

case 3:

return X3;

}

return X1;

}

/\*\*

\* Method to get Float Register from float register number

\*

\* @param fr

\* @return

\*/

public Register getFR(int fr) {

switch (fr) {

case 0:

return FR0;

case 1:

return FR1;

}

return FR0;

}

/\*\*

\* Method to load data

\* loads Memory Address Register and sets Memory buffer register

\*/

public void load() {

int dataAddr = Util.bitSet2Int(MAR);

int data = Util.bitSet2Int(memory.read(dataAddr));

setRegister(MBR, data);

}

/\*\*

\* Method to Store Data

\*

\* Reads Memory Buffer register and writes to address in memory address register

\*/

public void store() {

int dataAddr = Util.bitSet2Int(MAR);

Word data = Util.int2Word(Util.bitSet2Int(MBR));

memory.write(data, dataAddr);

}

/\*\*

\* Method to load Register with input register number with input value

\* @param regStr

\* @param input

\*/

public void loadRegisterFromInput(String regStr, String input) {

int value = Integer.parseInt(input, 2); // opcode||R||IX|I|Address

setRegister(regStr2Name(regStr), value);

}

/\*\*

\* Main method to implement operations

\* Switches to various instructions using opcode and performs appropriate action

\*

\*/

public int operation() {

int ea;

switch (opcode) {

// Load Register From Memory

case OpCodes.LDR:

ea = calculateEA(i, ix, addr);

setRegister(MAR, ea);

int dataAddr = Util.bitSet2Int(MAR);

int data = Util.bitSet2IntSigned(memory.read(dataAddr));

setRegisterSigned(MBR, data);

setRegister(getGPR(r), MBR);

setRegister(PC, Util.bitSet2Int(PC) + 1);

System.out.println("LDR R" + r + ", @$" + dataAddr);

break;

// Store Register to memory

case OpCodes.STR:

System.out.println("STR");

ea = calculateEA(i, ix, addr);

setRegister(MAR, ea);

setRegisterSigned(MBR, Util.bitSet2IntSigned(getGPR(r)));

memory.write(Util.bitSet2IntSigned(MBR), Util.bitSet2Int(MAR));

setRegister(PC, Util.bitSet2Int(PC) + 1);

System.out.println(Util.bitSet2IntSigned(MBR) + " @ $" + Util.bitSet2Int(MAR));

break;

// Load Register with address

case OpCodes.LDA:

System.out.println("LDA");

ea = calculateEA(i, ix, addr);

setRegister(MAR, ea);

setRegister(MBR, ea);

setRegister(getGPR(r), MBR);

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Load index register form memory

case OpCodes.LDX:

System.out.println("LDX");

ea = calculateEA((byte) 0, (byte) 0, addr);

setRegister(MAR, ea);

int dataAddr\_1 = Util.bitSet2Int(MAR);

int data\_1 = Util.bitSet2Int(memory.read(dataAddr\_1));

setRegister(MBR, data\_1);

setRegister(getIXR(ix), MBR);

System.out.println(Util.bitSet2Int(getIXR(ix)) + " ea:" + ea + " dataAddr:" + dataAddr\_1);

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Store Index Register to memory

case OpCodes.STX:

System.out.println("STX");

ea = calculateEA((byte) 0, ix, addr);

setRegister(MAR, ea);

setRegister(MBR, getIXR(ix));

memory.write(Util.bitSet2Int(MBR), Util.bitSet2Int(MAR));

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Jump if zero

case OpCodes.JZ:

System.out.println("JZ");

ea = calculateEA(i, ix, addr);

// cr is c(r) register content

int cr = Util.bitSet2Int(getGPR(r));

if (cr == 0) {

setRegister(PC, ea); // PC <- EA

} else {

setRegister(PC, Util.bitSet2Int(PC) + 1); // PC <- PC + 1

}

break;

// Jump if not Equal

case OpCodes.JNE:

System.out.println("JNE");

ea = calculateEA(i, ix, addr);

// cr is c(r) register content

cr = Util.bitSet2Int(getGPR(r));

if (cr != 0) {

setRegister(PC, ea); // PC <- -EA

} else {

setRegister(PC, Util.bitSet2Int(PC) + 1); // PC <- PC + 1

}

break;

// Jump if condition code

// cc replaces r for this instruction

// cc takes values 0, 1, 2, 3 as above and specifies

// the bit in the Condition Code Register to check

case OpCodes.JCC:

System.out.println("JCC");

byte cc = getCC(r);

ea = calculateEA(r, ix, addr);

if (cc == 1) {

setRegister(PC, ea);

} else {

setRegister(PC, Util.bitSet2Int(PC) + 1); // PC <- PC + 1

}

break;

// Unconditional Jump to address

case OpCodes.JMA:

System.out.println("JMA");

ea = calculateEA(i, ix, addr);

// Unconditional Jump To Address

// R is ignored in this instruction

setRegister(PC, ea); // PC <- EA

break;

// Jump and Save Return Address:

case OpCodes.JSR:

System.out.println("JSR");

ea = calculateEA(i, ix, addr);

// Jump and Save Return Address

setRegister(R3, Util.bitSet2Int(PC) + 1);

setRegister(PC, ea);

break;

//Return From Subroutine w/ return code as Immed

// portion (optional) stored in the instruction’s

// address field.

case OpCodes.RFS:

System.out.println("RFS");

setRegister(R0, addr);

setRegister(PC, R3);

break;

// Subtract One and Branch

case OpCodes.SOB:

System.out.println("SOB");

ea = calculateEA(i, ix, addr);

cr = Util.bitSet2IntSigned(getGPR(r));

setRegisterSigned(getGPR(r), cr - 1);

System.out.println(Util.bitSet2IntSigned(getGPR(r)));

if (cr >= 0) {

setRegister(PC, ea);

} else {

setRegister(PC, Util.bitSet2Int(PC) + 1); // PC <- PC + 1

}

break;

// Jump Greater Than or Equal To

case OpCodes.JGE:

System.out.println("JGE");

ea = calculateEA(i, ix, addr);

cr = -1;

cr = Util.bitSet2IntSigned(getGPR(r));

System.out.println(cr);

if (cr >= 0) {

setRegister(PC, ea);

} else {

setRegister(PC, Util.bitSet2Int(PC) + 1); // PC <- PC + 1

}

break;

// Multiply Register by Register

case OpCodes.MLT:

if ((r == 0 || r == 2) && (ry == 0 || ry == 2)) {

int crx = 0;

int cry = 0;

crx = Util.bitSet2Int(getGPR(r));

cry = Util.bitSet2Int(getGPR(ry));

int result = crx \* cry;

int upper = result >> 16;

int lower = result - (upper << 16);

if (upper > (2 ^ 16 - 1)) {

setCC(Constants.CC\_OVERFLOW, true); // setting OVERFLOW cc(0)

upper = upper - ((upper >> 16) << 16);

}

if (r == 0) {

setRegister(R0, upper);

setRegister(R1, lower);

} else {

setRegister(R2, upper);

setRegister(R3, lower);

}

} else {

// some kind of machine fault?

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Divide Register by Register

case OpCodes.DVD:

if ((r == 0 || r == 2) && (ry == 0 || ry == 2)) {

int crx = 0;

int cry = 0;

crx = Util.bitSet2Int(getGPR(r));

cry = Util.bitSet2Int(getGPR(ry));

if (cry == 0) {

setCC(Constants.CC\_DIVZERO, true); // setting DIVZERO

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

}

int result = crx / cry;

int upper = result;

int lower = crx - (cry \* upper);

if (r == 0) {

setRegister(R0, upper);

setRegister(R1, lower);

} else {

setRegister(R2, upper);

setRegister(R3, lower);

}

} else {

// some kind of machine fault?

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Test the Equality of Register and Register

case OpCodes.TRR:

int crx = Util.bitSet2Int(getGPR(r));

int cry = Util.bitSet2Int(getGPR(ry));

if (crx == cry) {

setCC(Constants.CC\_EQUALORNOT, true); // cc(4) <- 1

} else {

setCC(Constants.CC\_EQUALORNOT, false);

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Logical And of Register and Register

case OpCodes.AND:

System.out.println("AND");

getGPR(r).and(getGPR(ry));

setRegister(getGPR(r), getGPR(r));

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Logical Or of Register and Register

case OpCodes.ORR:

System.out.println("ORR");

getGPR(r).or(getGPR(ry));

setRegister(getGPR(r), getGPR(r));

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Logical Not of Register To Register

case OpCodes.NOT:

System.out.println("NOT");

getGPR(r).flip(0, getGPR(r).length());

setRegister(getGPR(r), getGPR(r));

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Add Memory To Register

case OpCodes.AMR:

System.out.println("AMR");

ea = calculateEA(i, ix, addr);

setRegister(MAR, ea);

dataAddr = Util.bitSet2Int(MAR);

data = Util.bitSet2Int(memory.read(dataAddr));

setRegister(MBR, data);

int result = Util.bitSet2IntSigned(getGPR(r)) + Util.bitSet2IntSigned(MBR);

setRegisterSigned(getGPR(r), result);

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Subtract Memory From Register

case OpCodes.SMR:

System.out.println("SMR");

ea = calculateEA(i, ix, addr);

setRegister(MAR, ea);

dataAddr = Util.bitSet2Int(MAR);

data = Util.bitSet2Int(memory.read(dataAddr));

setRegister(MBR, data);

result = Util.bitSet2IntSigned(getGPR(r)) - Util.bitSet2IntSigned(MBR);

System.out.println(ea + " " + i + " " + ix + " " + addr);

System.out.println(data + " @ $" + dataAddr);

System.out.println(Util.bitSet2IntSigned(getGPR(r)) + "-" + Util.bitSet2IntSigned(MBR) + "=" + result);

setRegisterSigned(getGPR(r), result);

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Add Immediate to Register,

case OpCodes.AIR:

System.out.println("AIR");

result = Util.bitSet2IntSigned(getGPR(r)) + addr;

setRegisterSigned(getGPR(r), result);

setRegister(PC, Util.bitSet2Int(PC) + 1);

System.out.println(Util.bitSet2IntSigned(getGPR(r)) + " + " + addr + " = " + result);

break;

// Subtract Immediate from Register

case OpCodes.SIR:

System.out.println("SIR");

result = Util.bitSet2IntSigned(getGPR(r)) - addr;

setRegisterSigned(getGPR(r), result);

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Shift Register by Count

case OpCodes.SRC:

System.out.println("SRC");

// arithmetic

if (al == 0) {

if (lr == 0) {

result = Util.bitSet2IntSigned(getGPR(r)) << count;

if (result > Constants.SIGNED\_MAX || result < Constants.SIGNED\_MIN) {

setCC(Constants.CC\_OVERFLOW, true);

}

setRegisterSigned(getGPR(r), result);

}

else if (lr == 1) {

result = Util.bitSet2IntSigned(getGPR(r)) >> count;

if (Util.bitSet2IntSigned(getGPR(r)) != (result << count)) {

setCC(Constants.CC\_UNDERFLOW, true);

}

setRegisterSigned(getGPR(r), result);

}

}

// logical

else if (al == 1) {

if (lr == 0) {

result = Util.bitSet2Int(getGPR(r)) << count;

if (result > Constants.UNSIGNED\_MAX) {

setCC(Constants.CC\_OVERFLOW, true);

}

setRegister(getGPR(r), result);

} else if (lr == 1) {

result = Util.bitSet2Int(getGPR(r)) >>> count;

if (Util.bitSet2Int(getGPR(r)) != (result << count)) {

setCC(Constants.CC\_UNDERFLOW, true);

}

setRegister(getGPR(r), result);

}

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Rotate Register by Count

case OpCodes.RRC:

System.out.println("RRC");

cr = Util.bitSet2Int(getGPR(r));

if (al == 1) {

if (lr == 0)

setRegister(getGPR(r), (cr << count) | (cr >> (16 - count)));

else if (lr == 1)

setRegister(getGPR(r), (cr >> count) | (cr << (16 - count)));

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Input Character To Register from Device, r = 0..3

case OpCodes.IN:

// Get devid

// devid = 0 keyboard

int inp = 0;

if (dev\_id == 0) {

inp = MainFrame.getKeyboard();

setRegister(getGPR(r), inp);

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Output Character to Device from Register, r = 0..3

case OpCodes.OUT:

// devid = 1 printer

if (dev\_id == 1) {

MainFrame.setPrinter(Util.bitSet2Int(getGPR(r)));

}

setRegister(PC, Util.bitSet2Int(PC) + 1);

break;

// Halt

case OpCodes.HLT:

System.out.println("HLT");

return 1;

}

return 0;

}

}

**Util**

package csci6461;

import java.util.BitSet;

/\*\*

\* General Purpose utility class to implement BitSet and word conversions

\*

\* @author Dinesh Reddy

\*

\*/

public class Util {

public static int bitSet2Int(BitSet bitSet) {

int intValue = 0;

for (int bit = 0; bit < bitSet.length(); bit++) {

if (bitSet.get(bit)) {

intValue |= (1 << bit);

}

}

return intValue;

}

public static int bitSet2IntSigned(BitSet bitSet) {

int intValue = 0;

int maxValue = 1 << bitSet.length()-1;

boolean sign = bitSet.get(15);

bitSet.set(15, false);

for (int bit = 0; bit < bitSet.length(); bit++) {

if (bitSet.get(bit)) {

intValue |= (1 << bit);

}

}

if (sign) {

bitSet.set(15);

return intValue-maxValue;

}

return intValue;

}

public static float bitSet2Float(BitSet bitSet) {

assert(bitSet.length() == 16);

int mantissaRep = 0;

double mantissa = 0.0;

int exponent = 0;

double result = 0.0f;

int maxValue = (1 << 8) - 1;

boolean sign = bitSet.get(15);

bitSet.set(15, false);

for (int bit = 0; bit < 8; bit++) {

if (bitSet.get(bit)) {

mantissaRep |= (1 << bit);

}

}

for (int bit = 8; bit < 15; bit++) {

if (bitSet.get(bit)) {

exponent |= (1 << (bit - 8));

}

}

exponent -= 63; // bias

if (sign) {

bitSet.set(15);

mantissaRep -= maxValue;

mantissa = (-1 + mantissaRep \* Math.pow(2, -8.0));

} else {

mantissa = (1 + mantissaRep \* Math.pow(2, -8.0));

}

result = mantissa \* Math.pow(2, exponent);

return (float)result;

}

public static BitSet int2BitSet(int value) {

BitSet bits = new BitSet();

int index = 0;

while (value != 0) {

if (value % 2 != 0) {

bits.set(index);

}

++index;

value = value >>> 1;

}

return bits;

}

public static BitSet int2BitSetSigned(int value) {

BitSet bits = new BitSet();

if (value < 0) {

bits.set(15);

//value = - 32768 - value; // 2's complement

}

int index = 0;

while (value != 0) {

if (value % 2 != 0) {

bits.set(index);

}

++index;

value = value >>> 1;

}

return bits;

}

public static BitSet float2BitSet(float value) {

BitSet bits = new BitSet();

int intbits = Float.floatToIntBits(value);

String binaryString = Integer.toBinaryString(intbits);

// append leading 0's

while (binaryString.length() < 32) {

binaryString = "0" + binaryString;

}

// signed bit

if (binaryString.charAt(0) == '1') bits.set(15);

String exponentString = binaryString.substring(1, 9);

while (exponentString.length() < 8) {

exponentString = "0" + exponentString;

}

int exponent = Integer.parseInt(exponentString, 2) - 127 + 63;

exponentString = Integer.toBinaryString(exponent);

while (exponentString.length() < 7) {

exponentString = "0" + exponentString;

}

String mantissaString = binaryString.substring(9, 17);

for (int i = 1; i < 8; i++) {

if (exponentString.charAt(i - 1) == '1') {

// 15 [14 13 12 11 10 9 8] 7 6 5 4 3 2 1 0

bits.set(15 - i);

}

}

for (int i = 8; i < 16; i++) {

// 15 14 13 12 11 10 9 8 [7 6 5 4 3 2 1 0]

if (mantissaString.charAt(i - 8) == '1') {

bits.set(15 - i);

}

}

return bits;

}

public static Word int2Word(int value) {

Word w = new Word();

int index = 0;

while (value != 0) {

if (value % 2 != 0) {

w.set(index);

}

++index;

value = value >>> 1;

}

return w;

}

public static Word int2WordSigned(int value) {

Word bits = new Word();

if (value < 0) {

bits.set(15);

//value = - 32768 - value; // 2's complement

}

int index = 0;

while (value != 0) {

if (value % 2 != 0) {

bits.set(index);

}

++index;

value = value >>> 1;

}

return bits;

}

public static void bitSetDeepCopy(BitSet source, int sourceBits,

BitSet destination, int destinationBits) {

if (sourceBits <= destinationBits) {

destination.clear();

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

destination.set(i, source.get(i));

}

} else {

destination.clear();

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

destination.set(i, source.get(i));

}

}

}

}

**Word**

package csci6461;

import java.util.BitSet;

public class Word extends BitSet {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public Word() {

super(16);

}

}