Mars Lite – The Humbler Version

A Simple calculator operations:

1. add destination register int int
2. addr destination register source register target register
3. load destination register int
4. loadr destination register source register
5. stor destination register int
6. storr destination register source registers
7. loadm destination register address
8. loadb destination register binary string
9. sub destination register int int
10. subr destination register source register target register
11. bounce int (pc counter to bounce to)

Mars Lite Source Code

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw2;

import hw2.Operands.Instruction;
import java.util.ArrayList;
import javax.swing.JTable;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;
import sun.swing.table.DefaultTableCellHeaderRenderer;

/**
 *
 * @author cahn
 */
public class MainDisplay extends javax.swing.JFrame {

    private String instructionRaw = new String();
    private String[] instructionLines = new String[300];
    private final ArrayList<Instruction> instructions = new ArrayList<>();
    private int instructionIndex;
    public static final int REGISTER_TABLE_VALUE = 2;
    public static final int MEMORY_TABLE_VALUE = 1;
    private final String[][] memoryModelTable = new String[128][9];
    private final TableModel memoryModel;
    public final static String[] REGISTER_NAMES
```

```
        = {"$zero", "$at", "$v0", "$v1",
           "$a0", "$a1", "$a2", "$a3", "$t0",
           "$t1", "$t2", "$t3", "$t4", "$t5",
           "$t6", "$t7", "$s0", "$s1", "$s2",
           "$s3", "$s4", "$s5", "$s6",
           "$k0", "$k1", "$gp", "$sp",
           "$fp", "$ra", "pc", "hi", "lo"};
public final static String[] REGISTER_VALUES = new String[REGISTER_NAMES.length];

public final int EditPage = 0;
public final int OpcodePage = 1;

/**
 * Creates new form MainDisplay
 */
public MainDisplay() {
    memoryModel = CreateMemoryModel();
    initComponents();
    initModels();
    stepOneButton.setEnabled(false);
    runButton.setEnabled(false);

    for (int i = 0; i < REGISTER_VALUES.length; i++) {
        REGISTER_VALUES[i] = Integer.toBinaryString(i);
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    compileButton = new javax.swing.JButton();
    stepOneButton = new javax.swing.JButton();
    runButton = new javax.swing.JButton();
    clearButton = new javax.swing.JButton();
    jTabbedPane2 = new javax.swing.JTabbedPane();
    jScrollPane1 = new javax.swing.JScrollPane();
    textEditor = new javax.swing.JTextArea();
    jPanel1 = new javax.swing.JPanel();
    jScrollPane4 = new javax.swing.JScrollPane();
    codeModel = new javax.swing.JTable();
    jScrollPane3 = new javax.swing.JScrollPane();
    memoryTable = new javax.swing.JTable();
    jScrollPane5 = new javax.swing.JScrollPane();
```

```
        registerBuffer = new javax.swing.JTable();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        compileButton.setText("Compile");
        compileButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                compileButtonActionPerformed(evt);
            }
        });

        stepOneButton.setText("Step One");
        stepOneButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                stepOneButtonMouseClicked(evt);
            }
        });
        stepOneButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                stepOneButtonActionPerformed(evt);
            }
        });

        runButton.setText("Run");
        runButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                runButtonActionPerformed(evt);
            }
        });

        clearButton.setText("Clear");
        clearButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                clearButtonActionPerformed(evt);
            }
        });

        textEditor.setColumns(20);
        textEditor.setRows(5);
        textEditor.setText("add $v1 1 9\nadd $t0 0 16\naddr $a0 $v1 $t0\nsubr $a1 $v1 $t0\nstorr $v1
0($t0)");
        textEditor.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                textEditorKeyTyped(evt);
            }
        });
        jScrollPane1.setViewportView(textEditor);

        jTabbedPane2.addTab("Edit", jScrollPane1);
```

```
jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

codeModel.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null},
        {null}
    },
    new String [] {
        "Instructions"
    }
) {
    boolean[] canEdit = new boolean [] {
        false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jScrollPane4.setViewportView(codeModel);
if (codeModel.getColumnModel().getColumnCount() > 0) {
    codeModel.getColumnModel().getColumn(0).setResizable(false);
}

jPanel1.add(jScrollPane4, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 740, 280));

memoryTable.setModel(memoryModel);
jScrollPane3.setViewportView(memoryTable);

jPanel1.add(jScrollPane3, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 320, 740, 290));

jTabbedPane2.addTab("Execute", jPanel1);

registerBuffer.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
        {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null}
        },
        new String [] {
            "Name", "Number", "Value"
        }
    ) {
        boolean[] canEdit = new boolean [] {
            false, true, false
        };

        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit [columnIndex];
        }
    });
    registerBuffer.getTableHeader().setReorderingAllowed(false);
    jScrollPane5.setViewportView(registerBuffer);
    if (registerBuffer.getColumnModel().getColumnCount() > 0) {
        registerBuffer.getColumnModel().getColumn(0).setResizable(false);
        registerBuffer.getColumnModel().getColumn(1).setResizable(false);
        registerBuffer.getColumnModel().getColumn(2).setResizable(false);
    }

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jTabbedPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 742,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(14, 14, 14)
```

```
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(runButton, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(clearButton, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(stepOneButton, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(compileButton, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 211,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(24, 24, 24)
                .addComponent(compileButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(stepOneButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(runButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(clearButton)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jTabbedPane2)
                    .addComponent(jScrollPane5))
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void compileButtonActionPerformed(java.awt.event.ActionEvent evt) {
        Clear();
        instructionRaw = textEditor.getText();
        instructionLines = instructionRaw.split("\n");
        for (int i = 0; i < instructionLines.length; i++) {
            Instruction inst = new Instruction(instructionLines[i], registerBuffer, memoryTable);
            instructions.add(inst);
            codeModel.getModel().setValueAt(inst.getInstructionString(), i, 0);
        }
        if (instructions.size() > 0) {
            stepOneButton.setEnabled(true);
            runButton.setEnabled(true);
        }
        jTabbedPane2.setSelectedIndex(OpcodePage);
    }

    private void Clear() {
        instructionIndex = 0;
        instructions.clear();
        for (int i = 0; i < registerBuffer.getModel().getRowCount(); i++) {
            registerBuffer.getModel().setValueAt("0", i, REGISTER_TABLE_VALUE);
        }
        for (int i = 0; i < memoryTable.getModel().getRowCount(); i++) {
            for (int j = 1; j < memoryTable.getModel().getColumnCount(); j++) {
                memoryTable.getModel().setValueAt("0", i, j);
            }
        }
        for (int i = 0; i < codeModel.getModel().getRowCount(); i++) {
            codeModel.getModel().setValueAt("", i, 0);
        }
    }

    private void stepOneButtonMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
    }

    private void stepOneButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if (instructionIndex < instructions.size()) {
            Instruction inst = instructions.get(instructionIndex);
            if (inst.getIsWriteOperation()) {
                memoryTable.getModel().setValueAt(inst.getResult(),
                        inst.getDestination(), MEMORY_TABLE_VALUE);
            } else {
                registerBuffer.getModel().setValueAt(inst.getResult(),
                        inst.getDestination(), REGISTER_TABLE_VALUE);
            }
            if (inst.isBounceInstruction()) {
                updatePC(instructionIndex);
                int oldIndex = instructionIndex;
                instructionIndex = inst.getBounceLocation();
                if (oldIndex > instructionIndex) {
                    for (int i = instructionIndex; i <= oldIndex; i++) {
                        String old = codeModel.getModel().getValueAt(i, 0).toString();
                        String replaced = old.replace("> ", "");
                        codeModel.getModel().setValueAt(replaced, i, 0);
                    }
                }
            } else if (instructionIndex + 1 < instructions.size()) {
                instructionIndex++;
                updatePC(instructionIndex - 1);
            } else {
                updatePC(instructionIndex);
                stepOneButton.setEnabled(false);
                runButton.setEnabled(false);
            }
        }
    }

    private void runButtonActionPerformed(java.awt.event.ActionEvent evt) {
        while (instructionIndex < instructions.size()) {
            Instruction inst = instructions.get(instructionIndex);
            if (inst.getIsWriteOperation()) {
                memoryTable.getModel().setValueAt(inst.getResult(),
                        inst.getDestination(), MEMORY_TABLE_VALUE);
            } else {
                registerBuffer.getModel().setValueAt(inst.getResult(),
                        inst.getDestination(), REGISTER_TABLE_VALUE);
            }
            if (inst.isBounceInstruction()) {
                updatePC(instructionIndex);
                int oldIndex = instructionIndex;
                instructionIndex = inst.getBounceLocation();
                if (oldIndex > instructionIndex) {
                    for (int i = instructionIndex; i <= oldIndex; i++) {
                        String old = codeModel.getModel().getValueAt(i, 0).toString();
                        codeModel.getModel().setValueAt(old.replace("> ", ""), i, 0);
                    }
                }
            } else if (instructionIndex + 1 < instructions.size()) {
                updatePC(instructionIndex);
                instructionIndex++;
            }
        }
        stepOneButton.setEnabled(false);
        runButton.setEnabled(false);
    }

    private void textEditorKeyTyped(java.awt.event.KeyEvent evt) {
        // EMPTY
    }

    private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
        Clear();
        jTabbedPane2.setSelectedIndex(EditPage);
    }

    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("GTK+".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;

                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(MainDisplay.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(MainDisplay.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(MainDisplay.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(MainDisplay.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MainDisplay().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton clearButton;
    private javax.swing.JTable codeModel;
    private javax.swing.JButton compileButton;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JTabbedPane jTabbedPane2;
private javax.swing.JTable memoryTable;
public javax.swing.JTable registerBuffer;
private javax.swing.JButton runButton;
private javax.swing.JButton stepOneButton;
private javax.swing.JTextArea textEditor;
// End of variables declaration

private TableModel CreateMemoryModel() {
    TableModel model = new TableModel() {
        @Override
        public int getRowCount() {

            return memoryModelTable.length;
        }

        @Override
        public int getColumnCount() {
            return memoryModelTable[0].length;
        }

        String[] colNames = {"Address", "Value(+0)", "Value(+4)", "Value(+8)",
            "Value(+12)", "Value(+16)", "Value(+20)", "Value(+24)", "Value(+30)"};

        @Override
        public String getColumnName(int arg0) {
            return colNames[arg0];
        }

        @Override
        public Class<?> getColumnClass(int arg0) {
            return String.class;
        }

        @Override
        public boolean isCellEditable(int arg0, int arg1) {
            return false;
        }

        @Override
        public Object getValueAt(int arg0, int arg1) {
            return memoryModelTable[arg0][arg1];
        }

        @Override
        public void setValueAt(Object arg0, int arg1, int arg2) {
            memoryModelTable[arg1][arg2] = arg0.toString();
            }

            @Override
            public void addTableModelListener(TableModelListener arg0) {
                // Empty
            }

            @Override
            public void removeTableModelListener(TableModelListener arg0) {
                // Empty
            }
        };
        return model;
    }

    private void initModels() {
        for (int i = 0; i < registerBuffer.getModel().getRowCount(); i++) {
            registerBuffer.getModel().setValueAt(String.valueOf(i), i, REGISTER_TABLE_VALUE - 1);
            registerBuffer.getModel().setValueAt(REGISTER_NAMES[i], i, REGISTER_TABLE_VALUE
- 2);
            registerBuffer.getModel().setValueAt("", i, REGISTER_TABLE_VALUE);
        }
        int memAddress = 0;
        for (int i = 0; i < memoryTable.getModel().getRowCount(); i++) {

            memoryTable.getModel().setValueAt(String.format("0x%1$08x", memAddress), i,
MEMORY_TABLE_VALUE - 1);
            memAddress += 4;
        }
    }

    private void updatePC(int index) {
        registerBuffer.getModel().setValueAt(index, 29, REGISTER_TABLE_VALUE);
        String old = codeModel.getModel().getValueAt(index, 0).toString();
        codeModel.getModel().setValueAt("> " + old, index, 0);
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw2.Operands;

import hw2.MainDisplay;
import hw2.RegisterLookup;
import java.util.Arrays;
import javax.swing.JTable;
```

```
/**
 *
 * @author cahn
 */
public class Instruction {

    private Operand operand;
    private int destination;
    private Object result;
    private byte opcode;
    private JTable registerTable;
    private JTable memoryTable;
    private final boolean isWriteOperation;
    private INSTRUCTION_TYPE instructionType;
    private int rsValue = 0;
    private int immAddress;
    private int rtValue;
    private int bounceLocation;

    public Instruction(String str, JTable registerTable, JTable memoryTable) {
        this.immAddress = 0;
        String[] individualWords = str.split(" ");
        assert (individualWords.length >= 3);

        this.registerTable = registerTable;
        this.memoryTable = memoryTable;
        assert (this.registerTable != null && this.memoryTable != null);

        destination = new RegisterLookup(individualWords[1]).getRegisterNumber();

        individualWords[0] = individualWords[0].toLowerCase();

        switch (individualWords[0]) {
            case "add":
                operand = new Add(individualWords[2], individualWords[3]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.R;
                break;
            case "addr":
                operand = new AddRegisters(individualWords[2], individualWords[3]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.R;
                break;
            case "load":
                operand = new Load(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "loadr":
                operand = new LoadRegister(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "stor":
                operand = new Store(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "storr":
                operand = new StoreRegister(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "loadm":
                operand = new LoadMemory(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "loadb":
                operand = new LoadByte(individualWords[2]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.I;
                break;
            case "sub":
                operand = new Subtract(individualWords[2], individualWords[3]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.R;
                break;
            case "subr":
                operand = new SubtractRegisters(individualWords[2], individualWords[3]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.R;
                break;
            case "bounce":
                operand = new Bounce(individualWords[1]);
                opcode = operand.getOpcode();
                instructionType = INSTRUCTION_TYPE.J;
                Bounce b = (Bounce)operand;
                bounceLocation = b.getBounceLocation();
                break;
            default:
                System.out.println("operation not found");
        }

        this.isWriteOperation = operand.isWriteOperation();
        result = operand.action();
        if (instructionType == INSTRUCTION_TYPE.R) {
```

```
            rsValue = Integer.parseInt(operand.getSource1().toString());
            if (!operand.hasOneSource()) {
                rtValue = Integer.parseInt(operand.getSource2().toString());
            }
        }
    }

    public int getBounceLocation(){
        return bounceLocation;
    }

    public boolean isBounceInstruction(){
        return operand.isBounce();
    }

    private void setImmediateAddress(int address) {
        immAddress = address;
    }

    public int getDestination() {
        if (isWriteOperation) {
            Store sr = (Store) operand;
            rsValue = sr.getRegisterNumberToRead();
            String address = registerTable.getModel().getValueAt(sr.getRegisterNumberToRead(),
MainDisplay.REGISTER_TABLE_VALUE).toString();
            int parsedAddress = Integer.parseInt(address);
            parsedAddress += sr.getOffset();
            setImmediateAddress(parsedAddress);
            return parsedAddress;
        }
        return destination;
    }

    public Object getResult() {

        if (isWriteOperation) {
            if (operand.usesConstants()) {
                result = getDestination();
            } else {
                String rawMemory = registerTable.getModel().getValueAt(destination,
MainDisplay.REGISTER_TABLE_VALUE).toString();
                result = Integer.parseInt(rawMemory);
            }
            return result;
        }

        if (instructionType == INSTRUCTION_TYPE.I || instructionType == INSTRUCTION_TYPE.R)
{
            if (operand.loadsMemory()) {
                String rawMemory =
memoryTable.getModel().getValueAt(Integer.parseInt(operand.getSource1().toString()),
MainDisplay.MEMORY_TABLE_VALUE).toString();
                operand.setSource1(Integer.parseInt(rawMemory));
            } else if (!operand.usesConstants()) {
                rsValue = Integer.parseInt(operand.getSource1().toString());
                String rawMemory = registerTable.getModel().getValueAt(rsValue,
MainDisplay.REGISTER_TABLE_VALUE).toString();
                operand.setSource1(Integer.parseInt(rawMemory));
                if (!operand.hasOneSource()) {
                    rtValue = Integer.parseInt(operand.getSource2().toString());
                    rawMemory = registerTable.getModel().getValueAt(rtValue,
MainDisplay.REGISTER_TABLE_VALUE).toString();
                    operand.setSource2(Integer.parseInt(rawMemory));
                }
            }
        }

        result = operand.action();
        return result;
    }

    public byte getOpcode() {
        return opcode;
    }

    public boolean getIsWriteOperation() {
        return isWriteOperation;
    }

    public INSTRUCTION_TYPE getInstructionType() {
        return instructionType;
    }

    public String byteArrayConverter(int value, int length) {
        boolean[] bits = new boolean[length];
        char[] sequence = new char[length];
        for (int i = 0; i < length; i++) {
            bits[i] = (value & (1 << i)) != 0;
            sequence[i] = bits[i] ? '1' : '0';
        }
        String result = new String(sequence);
        return result;
    }

    public String getInstructionString() {
        StringBuilder builder = new StringBuilder();
        builder.append(byteArrayConverter(opcode, 6));
        builder.append(" ");
        if (null != getInstructionType()) {
            switch (getInstructionType()) {
                case J:
                    builder.append(byteArrayConverter(immAddress, 26));
                    break;
                case R:
                    builder.append(byteArrayConverter(rsValue, 5));
                    builder.append(" ");
                    builder.append(byteArrayConverter(rtValue, 5));
                    builder.append(" ");
                    builder.append(byteArrayConverter(destination, 5));
                    builder.append(" ");
                    builder.append(String.format("%05d", 0));
                    builder.append(" ");
                    builder.append(String.format("%06d", 0));
                    break;
                case I:
                    builder.append(byteArrayConverter(destination, 5));
                    builder.append(" ");
                    builder.append(byteArrayConverter(rsValue, 5));
                    builder.append(" ");
                    builder.append(byteArrayConverter(immAddress, 16));
                    break;
                default:
                    break;
            }
        }

        return builder.toString();
    }

}

package hw2.Operands;

public class Subtract extends Operand<Integer> {

    public Subtract(String individualWord, String individualWord0) {
        super.setSource1(Integer.parseInt(individualWord));
        super.setSource2(Integer.parseInt(individualWord0));
    }

    @Override
    public Integer action() {
        return source1 - source2;
    }

    @Override
    public byte getOpcode() {
        return 5;
    }

    @Override
    public boolean usesConstants() {
        return true;
    }

    @Override
    public boolean hasOneSource() {
        return false;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory() {
        return false;
    }

    @Override
    public boolean isBounce() {
        return false;
    }

}

package hw2.Operands;

public class Add extends Operand<Integer> {

    public Add(String source1, String source2) {
        super.setSource1(Integer.parseInt(source1));
        super.setSource2(Integer.parseInt(source2));
    }

    @Override
    public Integer action() {
        return (super.source1 + super.source2);
    }

    @Override
    public byte getOpcode() {
        return 1;
    }
    @Override
    public boolean usesConstants() {
        return true;
    }

    @Override
    public boolean hasOneSource() {
        return false;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory() {
        return false;
    }

    @Override
    public boolean isBounce() {
        return false;
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw2.Operands;

import hw2.RegisterLookup;

/**
 *
 * @author cahn
 */
public class SubtractRegisters extends Operand<Integer> {

    public SubtractRegisters(String source1, String source2) {
        super.setSource1(new RegisterLookup(source1).getRegisterNumber());
        super.setSource2(new RegisterLookup(source2).getRegisterNumber());
    }

    @Override
    public boolean usesConstants() {
        return false;
    }

    @Override
    public boolean hasOneSource() {
        return false;
    }

    @Override
    public Integer action() {
        return source1 - source2;
    }

    @Override
    public byte getOpcode() {
        return 8;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory() {
        return false;
    }

    @Override
    public boolean isBounce() {
        return false;
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw2.Operands;

import hw2.RegisterLookup;

/**
 *
 * @author cahn
 */
public class AddRegisters extends Add {

    public AddRegisters(String source1, String source2) {
        super(String.valueOf(new RegisterLookup(source1).getRegisterNumber()),
                String.valueOf(new RegisterLookup(source2).getRegisterNumber()));
    }

    @Override
    public Integer action() {
        return source1 + source2;

    }

    @Override
    public boolean usesConstants() {
        return false;
    }

    @Override
    public boolean hasOneSource() {
        return false;
    }

    @Override
    public byte getOpcode() {
        return 2;
    }

    @Override
    public boolean isBounce() {
        return false;
    }

}

package hw2;

public class RegisterLookup {
    int registerNumber = 0;
    String registerNumberBinary;

    public RegisterLookup(String register) {
        register = register.toLowerCase();
        for (int i = 0; i < MainDisplay.REGISTER_NAMES.length; i++) {
          if(register.equals(MainDisplay.REGISTER_NAMES[i])){
             registerNumber = i;
             break;
          }
       }
        if(registerNumber == 0){
            System.out.println("register not found");
        }
    }

    public int getRegisterNumber() {
        return registerNumber;
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw2.Operands;

/**
 *
 * @author cahn
 */
public class Bounce extends Operand<Integer> {

    public Bounce(String val){
        super.setSource1(Integer.parseInt(val));
    }

    @Override
    public byte getOpcode() {
        return 13;
    }

    @Override
    public boolean usesConstants() {
        return false;
    }

    @Override
    public boolean hasOneSource() {
        return true;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory() {
        return false;
    }
    @Override
    public Integer action() {
        return source1;
    }

    @Override
    public boolean isBounce() {
        return true;
    }

    public int getBounceLocation(){
        return source1;
    }

}

package hw2.Operands;

/**
 *
 * @author cahn
 */
public abstract class Operand<T> {

    private byte opcode;
    protected T source1;
    protected T source2;

    public T getSource1() {
        return source1;
    }

    public T getSource2() {
        return source2;
    }

    public void setSource1(T source1) {
        this.source1 = source1;
    }

    public void setSource2(T source2) {
        this.source2 = source2;
    }

    public abstract T action();

    public abstract byte getOpcode();
    public abstract boolean usesConstants();

    public abstract boolean hasOneSource();

    public abstract boolean isWriteOperation();

    public abstract boolean loadsMemory();

    public abstract boolean isBounce();

}

package hw2.Operands;

public class Load extends Operand<Integer> {

    Load(String source1) {
        super.setSource1(Integer.parseInt(source1));
    }

    @Override
    public Integer action() {
        return super.source1;
    }

    @Override
    public byte getOpcode() {
        return 3;
    }

    @Override
    public boolean usesConstants() {
        return true;
    }

    @Override
    public boolean hasOneSource() {
        return true;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory(){
       return false;
    }
    @Override
    public boolean isBounce() {
        return false;
    }

}

package hw2.Operands;

public class LoadByte extends Operand<Integer> {

    public LoadByte(String source1) {
        super.setSource1(Integer.parseInt(source1,2));
    }

    @Override
    public byte getOpcode() {
        return 7;
    }

    @Override
    public Integer action() {
        return source1;
    }

    @Override
    public boolean usesConstants() {
        return true;
    }

    @Override
    public boolean hasOneSource() {
        return true;
    }

    @Override
    public boolean isWriteOperation() {
        return false;
    }

    @Override
    public boolean loadsMemory() {
        return false;
    }

    @Override
    public boolean isBounce() {
        return false;
    }

}

package hw2.Operands;

import hw2.RegisterLookup;

/**
 *
 * @author cahn
 */
public class Store extends Operand<Byte> {

    protected int registerNumberToRead = 0;
    protected short offset = 0;
    protected int valueInRegister = 0;

    public Store(String source1) {
        String parts[] = source1.split("[(]");
        for (int i = 0; i < parts.length; i++) {
            parts[i] = parts[i].replace("(", "");
            parts[i] = parts[i].replace(")", "");
        }
        this.registerNumberToRead = new RegisterLookup(parts[1]).getRegisterNumber();
        this.offset = twosComplement(parts[0]);

    }

    private short twosComplement(String number){
       return (short) (Short.parseShort(number));
    }

    public int getRegisterNumberToRead() {
        return registerNumberToRead;
    }

    public short getOffset() {
        return offset;
    }

    @Override
    public Byte action() {
        return source1;
    }

    @Override
    public byte getOpcode() {
        return 6;
    }

    @Override
    public boolean usesConstants() {
        return true;
    }

    @Override
    public boolean hasOneSource() {
        return true;
    }

    @Override
    public boolean isWriteOperation() {
        return true;
    }

   @Override
   public boolean loadsMemory() {
     return false;
   }

    @Override
    public boolean isBounce() {
        return false;
    }
}

package hw2.Operands;

public class StoreRegister extends Store {

   public StoreRegister(String source1) {
     super(source1);
   }

   @Override
   public Byte action() {
     return source1;
   }

   @Override
   public boolean usesConstants() {
     return false;
   }

}

package hw2.Operands;
```

```
public enum INSTRUCTION_TYPE {
    R, I, J
};
```