Name: Sama Ayman Mokhtar Amin

Table of Contents

1 INTRODUCTION

This document aims to discuss in depth the implementation of a MIPS emulator. Nineteen instructions are supported and can be grouped into three Formats. First, the R-Format consists of the add, sub, or, nor, and, slt, sll, srl & sra instructions. Second, the I-Format consists of the addi, ori, andi, lui, slti, lw & sw instructions. Finally, the J-Format consists of the j instruction.

Starting with the emulator, the document begins by discussing the logic behind each part of the code and stating any important side notes. Then, moving on to showing the test code, which does not perform any useful functionality; however, it was chosen carefully to make sure it tests all the distinct functionalities. Finally, the output of the program is compared to the expected actual output if it was to be solved manually.

2 EMULATOR

The emulator implemented below takes the assembly code in a text file and enables the user to trace the values of registers after the execution of each line. These values are displayed in decimal so that the screen can accommodate all the 32 registers. Besides, whenever a store word (sw) instruction is encountered a table shows the address of the memory and the value that is stored in it on the right side of the window.

2.1 Emulator's Implementation with Demonstration

The Java program implemented consists of classes. For each class an explanation for the class's attributes and methods with their implementation will be demonstrated followed by the code for that class.

1.1.1. Tool class

When taking a closer look, one finds that there are common methods that will be needed in the program as converting from decimals to two's complement and vice versa. Also, a method that extends a binary number to occupy a 32 bit is useful. For these reasons, I decided to add a Tool class that groups all of the utility functions. Each of these functions is briefly explained below.

a. **twosComplementToDecimal:** is a function that takes a 32 bit two's complement string and changes it to decimal. It starts by checking the first bit if it is 0 which means the number is positive. Thus, it converts it directly using a java supported method. However, if the left most bit is 1 which means the number is negative, it gets its two's complement, changes it to decimal, and adds a negative sign.

b. **decimaltoTwosComplement:** is a function that takes a decimal number and changes it to two's complement. It start by checking if there is no negative sign. If yes, it converts it directly to binary using a java supported method. However, if the number is negative, it takes its absolute value, changes it to binary, extends it , and finally gets its twos complement.

c. **getTwosComplement:** is a function that takes a32-bit binary number and changes it to two's complement. It is a private method used by method 'a' and 'b'. It starts from the right most bit putting every zero as it is until it encounters a one where it also put it as it is. After that it flips all the coming bits till it reaches bit number 32.

d. **extendByZeros:** this method takes any binary number of less than 32 bits and extends the left most bits with zeros till the binary number reaches 32 bits.

```
package sample;

import static java.lang.Math.abs;
import static java.lang.Math.pow;

public class Tool {
    static String twosComplementToDecimal(String b32){
        String sfinal;
        if(b32.charAt(0) == '0'){
            int b = Integer.parseInt(b32,2);
            sfinal  = String.valueOf(b);
         }
         else {
             String b =getTwosComplement(b32);
             String withoutSign = b.substring(1);
             int last = Integer.parseInt(withoutSign,2);
              sfinal = String.valueOf(Integer.valueOf(last)*-1);
         }
        return sfinal;
    }
     static String decimaltoTwosComplement(String b32){
         String sfinal ;
         if(Integer.parseInt(b32) >= 0){
            String b = Integer.toBinaryString(Integer.parseInt(b32));
            sfinal  = extendByZeros(b);

         }
         else {
             String b = Integer.toBinaryString(abs(Integer.parseInt(b32)));
             b = extendByZeros(b);
             b =getTwosComplement(b);
              sfinal = b;
         }
        return sfinal;
     }
     static private String getTwosComplement (String b32){
         boolean flag = false;
         char[] arr =  new char[32];
         for (int i = 31; i > -1; i--) {
              if(b32.charAt(i) == '1' && flag == false){
                 flag = true;
                 arr[i] = '1';
             }
             else if(flag == false){
                 arr[i] = b32.charAt(i);
             }
             else {
                  if(b32.charAt(i) == '1'){
                      arr[i]= '0';
                  }
                  else
                      arr[i] = '1';
             }
         }
```

```
            return String.valueOf(arr);
        }
        static private String extendByZeros(String b){
            int length =  b.length();
            int left = 32 - b.length();
            char[] arr = new char[32];
            for (int i = 0; i < left ; i++) {
                arr[i] = '0';
            }
            for (int i = 0; i < length ; i++) {
                arr[left + i] = b.charAt(i);
            }
            return String.valueOf(arr);
        }
}
```

1.1.2. Register class (enum)

Registers are like constant in this program that has names, values, and description. Names is the way they are represented in the java code, for example s4,t0 ,…etc. Values are what the register holds and its represented as 32 binary bits. All Registers are initialized by zeros. Description is the way this registers are actually named as $s4,$t0,… etc. Enum, which is a specific type of a java class, is used to hold these 32 registers. Simple methods such as setValue and getValue are implemented in this

```
package sample;
public enum Register {
    zero("$zero"),at("$at"),v0("$v0"),v1("$v1"),a1("$a1"),a2("$a2"),
   a3("$a3"),t0("$t0"),t1("$t1"),t2("$t2"),t3("$t3"),t4("$t4"),t5("$t5"),
    t6("$t6"),t7("$t7"),s0("$S0"),s1("$S1"),s2("$S2"),s3("$S3"),s4("$S4"),
    s5("$S5"), s6("$S6"),s7("$S7"),t8("$t8"),t9("$t9"),k0("$k0"),k1("$k1"),
    gp("$gp"), sp("$sp"),fp("$fp"),ra("$ra");
    boolean isUsed;
    String value;
    String name;
    private Register(String name){
        this.name = name;
        isUsed = false;
        value = "00000000000000000000000000000000";
    }
    public String getName(){
        return name;
    }
    void setValue(String value){
        this.value = value;
        isUsed = true;
    }
    String getValue(){
        isUsed = true;
        return value;
    }
}
```

1.1.3. Operation class

This class implements all the nineteen instructions that are supported by this program. A brief comment on each method is given below. Besides, this class has a hash map data field that is used to represent memory which is needed for store and load instructions.

a. **add**: takes the value of rs and rt registers, adds them, and put the result in rd

b. **sub:** takes the value of rs and rt registers, subtracts them, and put the result in rd
c. **addi:** takes the value of rs register and an immediate, adds them, and put the result in rd
d. **and:** takes the value of rs and rt registers, changes them to binary, compares bit by bit, and according to the truth table sets rd
e. **or:** takes the value of rs and rt registers, changes them to binary, compares bit by bit, and according to the truth table sets rd
f. **andi:** takes the value of rs register and an immediate, changes them to binary, compares bit by bit, and according to the truth table sets rd
g. **ori:** takes the value of rs register and an immediate, changes them to binary, compares bit by bit, and according to the truth table sets rd
h. **nor:** takes the value of rs and rt registers, changes them to binary, compares bit by bit, and according to the truth table sets rd
i. **sll:** takes the value of rs register and shift amount, changes the shift amount to a string with that number of zeros, joins the substring of rs to the zeros string , and sets rd
j. **srl:** takes the value of rs register and shift amount, changes the shift amount to a string with that number of zeros, joins the zeros string to a substring of rs, and sets rd
k. **sra:** takes the value of rs register and shift amount, changes the shift amount to a string with that number of zeros or ones depending on the left most bit, joins the zeros string to a substring of rs, and sets rd
l. **slt**：takes the value of rs and rt registers, compares them and if rs is smaller sets rd to ‘1’
m. **slti**：takes the value of rs register and an immediate, compares them and if rs is smaller sets rd to ‘1’
n. **beq**：takes the value of rs and rt registers, compares them and if they are equal it sets the program counter with the line number where the label occurs
o. **bne**：takes the value of rs and rt registers, compares them and if they are not equal it sets the program counter with the line number where the label occurs
p. **j**：sets the program counter with the line number where the label occurs
q. **sw:** takes the value of the second register adds a constant to it, puts it as a key of a hash map, takes the value of the second register and stores it as the value of the hash map
r. **lw:** takes the value of the second register adds a constant to it, takes that memory address as the hash map’s key and setting its value by that of the first register
s. **lui:** takes the immediate value changes it to its 16 bit binary equivalent, and joins 16 zero-digits binary number to its right

```java
package sample;
import java.util.HashMap;
public class Operation {
    static HashMap<String, String> memory = new HashMap<>();

    static  void add(Register rd, Register rs,Register rt){
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        int rtNum =
Integer.parseInt(Tool.twosComplementToDecimal(rt.getValue()));
        int rdNum = rsNum + rtNum;
        rd.setValue(Tool.decimaltoTwosComplement(String.valueOf(rdNum)));
    }
    static  void sub(Register rd, Register rs,Register rt){
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        int rtNum =
Integer.parseInt(Tool.twosComplementToDecimal(rt.getValue()));
```

```
        int rdNum = rsNum - rtNum;
        rd.setValue(Tool.decimaltoTwosComplement(String.valueOf(rdNum)));
    }
    static  void addi(Register rd, Register rs, String immediate){
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        int rtNum = Integer.parseInt(immediate);
        int rdNum = rsNum + rtNum;
        rd.setValue(Tool.decimaltoTwosComplement(String.valueOf(rdNum)));
    }
    static void and(Register rd, Register rs, Register rt){
        char arr[] = new char[32];
        for (int i = 0; i < 32; i++) {
            if(rs.getValue().charAt(i) == '1' && rt.getValue().charAt(i) == '1')
                arr[i] = '1';
            else
                arr[i] = '0';
        }
        rd.setValue(String.valueOf(arr));
    }
    static void or(Register rd, Register rs, Register rt){
        char arr[] = new char[32];
        for (int i = 0; i < 32; i++) {
            if(rs.getValue().charAt(i) == '0' && rt.getValue().charAt(i) == '0')
                arr[i] = '0';
            else
                arr[i] = '1';
        }
        rd.setValue(String.valueOf(arr));
    }
     static void andi(Register rd, Register rs, String immediate){
        char arr[] = new char[32];
        for (int i = 0; i < 32; i++) {
            if(rs.getValue().charAt(i) == '1' && immediate.charAt(i) == '1')
                arr[i] = '1';
            else
                arr[i] = '0';
        }
        rd.setValue(String.valueOf(arr));
    }
    static void ori(Register rd, Register rs, String immediate){
        char arr[] = new char[32];
        for (int i = 0; i < 32; i++) {
            if(rs.getValue().charAt(i) == '0' && immediate.charAt(i) == '0')
                arr[i] = '0';
            else
                arr[i] = '1';
        }
        rd.setValue(String.valueOf(arr));
    }
     static void nor(Register rd, Register rs, Register rt){
        char arr[] = new char[32];
        for (int i = 0; i < 32; i++) {
            if(rs.getValue().charAt(i) == '0' && rt.getValue().charAt(i) == '0')
                arr[i] = '1';
            else
                arr[i] = '0';
        }
        rd.setValue(String.valueOf(arr));
    }
    static void sll(Register rd, Register rs, String shiftAmount){
        String extension = "";
        int sa = Integer.parseInt(Tool.twosComplementToDecimal(shiftAmount));
        for (int i = 0; i < sa; i++) {
            extension = extension.concat("0");
        }
        String subString = rs.getValue().substring((0+sa));
        String s = subString.concat(extension);
       // System.out.println(s);
        rd.setValue(s);
    }
    static void srl(Register rd, Register rs, String shiftAmount){
        String extension = "";
        int sa = Integer.parseInt(Tool.twosComplementToDecimal(shiftAmount));
        for (int i = 0; i < sa; i++) {
            extension = extension.concat("0");
        }
        String subString = rs.getValue().substring(0,(32 - sa));
        String s = extension.concat(subString);
        rd.setValue(s);
    }
     static void sra(Register rd, Register rs, String shiftAmount){
        String extension = "";
        boolean isPositive = (rs.getValue().charAt(0) == '0');
        int sa = Integer.parseInt(Tool.twosComplementToDecimal(shiftAmount));
        for (int i = 0; i < sa; i++) {
            if (isPositive)
            extension = extension.concat("0");
            else
            extension = extension.concat("1");
        }
        String subString = rs.getValue().substring(0,(32 - sa));
        String s = extension.concat(subString);
        rd.setValue(s);
    }
    static  void slt(Register rd, Register rs,Register rt){
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        int rtNum =
Integer.parseInt(Tool.twosComplementToDecimal(rt.getValue()));
        int rdNum = rsNum - rtNum;
        if (rdNum < 0)
            rd.setValue(Tool.decimaltoTwosComplement("1"));
        else
           rd.setValue(Tool.decimaltoTwosComplement("0"));
    }
    static  void slti(Register rd, Register rs,String immediate){
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        int rtNum = Integer.parseInt(immediate);
        int rdNum = rsNum - rtNum;
        if (rdNum < 0)
            rd.setValue(Tool.decimaltoTwosComplement("1"));
        else
           rd.setValue(Tool.decimaltoTwosComplement("0"));
    }
    static void beq(Register rd, Register rs, String label) {
        int rdNum =
Integer.parseInt(Tool.twosComplementToDecimal(rd.getValue()));
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        if (rsNum == rdNum) {
            Controller.programCounter = Controller.findLabelLine(label) ;
        }
        else
            Controller.programCounter ++;
    }
     static void bne(Register rd, Register rs, String label) {
        int rdNum =
Integer.parseInt(Tool.twosComplementToDecimal(rd.getValue()));
        int rsNum =
Integer.parseInt(Tool.twosComplementToDecimal(rs.getValue()));
        if (!(rsNum == rdNum)) {
            Controller.programCounter = Controller.findLabelLine(label) ;
        }
        else
            Controller.programCounter ++;
    }
     static void j(String label) {
            Controller.programCounter = Controller.findLabelLine(label) ;
    }

    static void sw(Register r1, String offset, Register r2){
        int offfset = Integer.parseInt(offset);
        int memoryAddress =
Integer.parseInt(Tool.twosComplementToDecimal(r2.getValue())) ;
        int sum = offfset + memoryAddress;
        memory.put( String.valueOf(sum) , r1.getValue() );
    }

    static void lw(Register r1, String offset, Register r2){
        int offfset = Integer.parseInt(offset);
        int memoryAddress =
Integer.parseInt(Tool.twosComplementToDecimal(r2.getValue())) ;
        int sum = offfset + memoryAddress;
        r1.setValue( memory.get( String.valueOf(sum) ) );
    }
    static void lui(Register rd, String immediate){
        char arr[] = new char[16];
        String immed = (Tool.decimaltoTwosComplement(immediate)).substring(16);
        for (int i = 0; i < 16; i++) {
            arr[i] = '0';
        }
        String s = String.valueOf(arr);
        String sum = immed.concat(s);
        rd.setValue(sum);
    }
}
```

1.1.4. Controller class

The function of this class is to initiate a program counter variable and monitor the flow of the program. It detects the type of an instruction, takes its parameters, and passes it to the appropriate method in the operation class. Finally it increments the value of the program counter.

a. **executeInstuction:** this method opens the file where the assembly code it stored, goes to the line that is stated by the program counter , checks if the program counter is less than or equal the number of lines in the assembly text file, if yes, it calls the method **subExecuteOperation**, and finally it calls **subExecuteOperation** again if none of the operations got executed due to a label that precedes

b. **subExecuteOperation:** this method takes a string containing an operation and according to it prepares the values of parameters needed by the **Operation class,** and finally it passes a boolean value communicating to the **excuteInstruction** whether something was executed or it was just a label

c. **findLabelLine**: this method is needed whenever a j, beq, or bne instruction is encountered. This method is passed a label and returns the line where that label is found

d. **setLineCount**: this method simply counts the number of lines in the assembly text file before and instruction get executed just to know where the program ends

e.
```
package sample;

    import java.util.Scanner;
    import java.io.File;
    import java.util.HashMap;

    public class Controller {
        static int programCounter = 1;
        static Scanner scan;
        static int lineCount;
        static HashMap<Integer,String>  InstHashMap = new HashMap<>();

        static void executeInstruction() {

            openFile();
            for (int i = 1; i < programCounter; i++) {
                scan.nextLine();
            }
            if (lineCount >= programCounter) {
                String operation = scan.next();
                boolean done = subExcuteOperation(operation);
                if( !done ){
                    if(scan.hasNext()){
                      String operation2 = scan.next();
                      subExcuteOperation(operation2);
                    }else
                    {
                        programCounter = 1 + lineCount;
                    }

                }

                //if (!scan.hasNext()) {
                  //  endOfProgram = true;
                //}

                closeFile();
            }
        }
        static private boolean subExcuteOperation(String operation){
            boolean done = true;
            if (operation.compareTo("add") == 0) {
                    Operation.add(Register.valueOf(scan.next().substring(1, 3)),
    Register.valueOf(scan.next().substring(1, 3)),
    Register.valueOf(scan.next().substring(1,3)));
                    programCounter++;
                } else if (operation.compareTo("sub") == 0) {
                    Operation.sub(Register.valueOf(scan.next().substring(1, 3)),
    Register.valueOf(scan.next().substring(1, 3)),
    Register.valueOf(scan.next().substring(1,3)));
                    programCounter++;
                } else if (operation.compareTo("addi") == 0) {
                    Operation.addi(Register.valueOf(scan.next().substring(1, 3)),
    Register.valueOf(scan.next().substring(1, 3)), scan.next());
                    programCounter++;
                } else if (operation.compareTo("and") == 0) {
```

```
                Operation.and(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1,3)));
                programCounter++;
            } else if (operation.compareTo("or") == 0) {
                Operation.or(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1,3)));
                programCounter++;
            } else if (operation.compareTo("andi") == 0) {
                Operation.andi(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Tool.decimaltoTwosComplement(scan.next()));
                programCounter++;

            } else if (operation.compareTo("ori") == 0) {
                Operation.ori(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Tool.decimaltoTwosComplement(scan.next()));
                programCounter++;
            } else if (operation.compareTo("nor") == 0) {
                Operation.nor(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1,3)));
                programCounter++;

            } else if (operation.compareTo("sll") == 0) {
                Operation.sll(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Tool.decimaltoTwosComplement(scan.next()));
                programCounter++;

            } else if (operation.compareTo("srl") == 0) {
                Operation.srl(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Tool.decimaltoTwosComplement(scan.next()));
                programCounter++;
            } else if (operation.compareTo("sra") == 0) {
                Operation.sra(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Tool.decimaltoTwosComplement(scan.next()));
                programCounter++;
            } else if (operation.compareTo("slt") == 0) {
                Operation.slt(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1,3)));
                programCounter++;
            } else if (operation.compareTo("slti") == 0) {
                Operation.slti(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)), scan.next());
                programCounter++;
            } else if (operation.compareTo("beq") == 0) {
                Operation.beq(Register.valueOf(scan.next().substring(1, 3)),
Register.valueOf(scan.next().substring(1, 3)), scan.next());
            }
            else if (operation.compareTo("bne") == 0) {
                    Operation.bne(Register.valueOf(scan.next().substring(1,
3)), Register.valueOf(scan.next().substring(1, 3)), scan.next());
            }
            else if (operation.compareTo("j") == 0) {
                    Operation.j(scan.next());
            }
            else if (operation.compareTo("sw") == 0) {
                    Register r = Register.valueOf(scan.next().substring(1,
3));
                    String connected = scan.next();
                    int size = connected.length();
                    int index = 0;
                    for (int i = 0; i < size ; i++) {
                        if(connected.charAt(i)=='('){
                           index = i ;
                        }
                    }
                    if(connected.charAt(index+2)== 'z'){
                        Operation.sw(r, connected.substring(0,index) ,
Register.valueOf(connected.substring(index+2 ,size -3)) );
                    }
                    else{
                        Operation.sw(r, connected.substring(0,index) ,
Register.valueOf(connected.substring(index+2 ,size -1)) );
                    }
                    programCounter ++ ;
            }
            else if (operation.compareTo("lw") == 0) {
                    Register r = Register.valueOf(scan.next().substring(1,
3));
                    String connected = scan.next();
                    int size = connected.length();
                    int index = 0;
                    for (int i = 0; i < size ; i++) {
                        if(connected.charAt(i)=='('){
                           index = i ;
                        }
                    }
                   // System.out.println(connected.substring(1,index)+ " " +
connected.substring(index+2,size -1) );
                    if(connected.charAt(index+2)== 'z'){
                        Operation.lw(r, connected.substring(0,index) ,
Register.valueOf(connected.substring(index+2 ,size -3)) );
                    }
                    else{
                        Operation.lw(r, connected.substring(0,index) ,
Register.valueOf(connected.substring(index+2 ,size -1)) );
                    }
                    programCounter ++;
            }
            else if (operation.compareTo("lui") == 0) {
                    Operation.lui(Register.valueOf(scan.next().substring(1,
3)), scan.next());
                    programCounter ++;
            }
            else
                done = false;
            return done;
    }
    static int findLabelLine(String label){
        int lineNum = 1;
        openFile();

        while(scan.hasNext()){
            if (scan.nextLine().startsWith(label))
                break;
            else{
                lineNum ++;
            }
        }
```

```
                closeFile();
                return lineNum;
            }
            static void setlineCount(){
                openFile();
                while(scan.hasNext()){
                        lineCount++;
                        InstHashMap.put(lineCount, scan.nextLine());

                    }
                closeFile();
            }
            static  void openFile(){
                    try{
                        File file = new File("AssemblyCode");
                        scan = new Scanner(file);
                    }
                    catch (Exception e){
                        System.out.println("File not Found!");
                    }
            }
            static void closeFile(){
                        scan.close();
            }
        }
```

1.1.5. FileWriter class

This class simple class takes the assembly code form the user and stores it in a text file.

```
package sample;
import java.io.File;
import java.io.PrintWriter;
public class FileWriter {
    static PrintWriter write ;
    static void writeFile(String s) {
        try {
            File file = new File("AssemblyCode");
            write = new PrintWriter(file);

        } catch (Exception e) {
            System.out.println("File not Found! {while writing}");
        }
        write.printf(s);
        write.close();
    }
}
```

1.1.6. Main class

```
This class mainly contains the code behind the Graphical User Interface(GUI)
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.control.ScrollPane.ScrollBarPolicy;
import java.util.Map;

public class Main extends Application {
    Boolean binary = false;
     BorderPane bp;
     GridPane middleGP = new GridPane();
     ScrollPane sp = getSp();
     HBox topHBox;
     VBox bottomVBox;
     int dynamic;
     VBox leftVBox;
     static Boolean start = false;

    @Override
    public void start(Stage primaryStage) throws Exception{
                Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));

         bp = new BorderPane();

         HBox topi = new HBox();
         Label inst = new Label("Please, Delete The Assembly Code Below, Paste Yours
Then click Continue");
        inst.setStyle("-fx-font-size: 16pt;");
        topi.getChildren().addAll(inst);
        topi.setSpacing(20);
        topi.setAlignment(Pos.CENTER);
        topi.setPadding(new Insets(20,20,20,20));
         bp.setTop(topi);

         VBox bv = new VBox();
        Button cont = new Button("Continue");
        cont.setStyle("-fx-font-size: 16pt");
        bv.getChildren().add(cont);
        bv.setAlignment(Pos.CENTER);
        bv.setPadding(new Insets(20,20,20,20));
        cont.setOnAction(e -> nextAction());
        bp.setBottom(bv);

         TextArea ta = new TextArea();
         ta.setText("slti $at, $s5, 5 \nbeq $at, $zero, Else \nadd $s6, $s5, $zero \nj
Exit \nElse: add $s6, $zero, $zero \nExit:");
         bp.setCenter(ta);
         cont.setOnAction(e-> {
             FileWriter.writeFile(ta.getText());
             addRowMiddleGP(0,0);
             middleGP.setVgap(10);
             middleGP.setHgap(10);
             middleGP.setPadding(new Insets(20,20,20,20));
             //middleGP.setGridLinesVisible(true);
            getLeftVBox();
            bp.setRight(leftVBox);
            bp.setCenter(sp);
            bp.setBottom(getBottomVBox("Next Instruction"));
            bp.setTop(getTopHBox("Click 'Next Instruction' to Start"));
            Controller.setlineCount();
        //System.out.println(Controller.InstHashMap);
        Register.s0.setValue(Tool.decimaltoTwosComplement("10"));
        Register.s1.setValue(Tool.decimaltoTwosComplement("12"));
        Register.s2.setValue(Tool.decimaltoTwosComplement("32"));
        Register.s5.setValue(Tool.decimaltoTwosComplement("500"));
```

```
         });
        Scene scene = new Scene(bp, 900, 600);
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    VBox getBottomVBox(String s){
        bottomVBox = new VBox();
        Button nextInstruction = new Button(s);
        nextInstruction.setStyle("-fx-font-size: 16pt");
        bottomVBox.getChildren().add(nextInstruction);
        bottomVBox.setAlignment(Pos.CENTER);
        bottomVBox.setPadding(new Insets(20,20,20,20));
        nextInstruction.setOnAction(e -> nextAction());
        return bottomVBox;
    }

    void nextAction(){
        if(Controller.programCounter <= Controller.lineCount){
            bp.setTop(getTopHBox(Controller.programCounter ,
Controller.InstHashMap.get(Controller.programCounter)));
            int pc = Controller.programCounter;
            Controller.executeInstruction();
            bp.setRight(getLeftVBox());
            dynamic ++;
            addRowMiddleGP(pc,dynamic);
            //System.out.println(Controller.programCounter);
            //System.out.println(Register.t0.getValue());
        }
        else{
            bp.setTop(getTopHBox( "End Of Program"));
        }
    }
    HBox getTopHBox(int num, String instruction){
        topHBox = new HBox();
        Label instCount = new Label("Line Number " + num + " ");
        instCount.setStyle("-fx-font-size: 16pt");
        Label inst = new Label(instruction);
        inst.setStyle("-fx-font-size: 16pt;");
        topHBox.getChildren().addAll(instCount,inst);
        topHBox.setSpacing(20);
        topHBox.setAlignment(Pos.CENTER);
        topHBox.setPadding(new Insets(20,20,20,20));
        return topHBox;
    }
    HBox getTopHBox( String instruction){
        topHBox = new HBox();

        Label inst = new Label(instruction);
        inst.setStyle("-fx-font-size: 16pt;");
        topHBox.getChildren().addAll(inst);
        topHBox.setSpacing(20);
        topHBox.setAlignment(Pos.CENTER);
        topHBox.setPadding(new Insets(20,20,20,20));
        return topHBox;
    }
    ScrollPane getSp(){
        ScrollPane sp = new ScrollPane();
        sp.setContent(middleGP);
        sp.setVbarPolicy(ScrollBarPolicy.ALWAYS);
        sp.setHbarPolicy(ScrollBarPolicy.ALWAYS);
        //sp.setFitToWidth(true);
        sp.setPannable(true);
        return  sp;
    }
    VBox getLeftVBox(){
        leftVBox = new VBox();
        GridPane memGp = new GridPane();
        memGp.setVgap(10);
        memGp.setHgap(10);
        memGp.setPadding(new Insets(10,10,10,10));
        VBox label = new VBox();
        Label l = new Label("Memory");
        l.setStyle("-fx-font-size: 16pt;");
        label.getChildren().add(l);
        label.setAlignment(Pos.CENTER);
        memGp.add(new Label("Address"),0,0);
        memGp.add(new Label("Value"),1,0);
        for ( Map.Entry<String, String> entry : Operation.memory.entrySet()) {
            String key = entry.getKey();
            String tab = entry.getValue();
            memGp.add(new Label(key),0,memGp.getRowCount());
            memGp.add(new Label(tab),1,memGp.getRowCount()-1);
        }
        leftVBox.getChildren().addAll(label,memGp);
        return leftVBox;
    }
    void addRowMiddleGP(int pc,int dynamic){

        if(pc == 0){
            Label f = new Label("Line #");
             f.setStyle("-fx-font-weight: bold;");
             middleGP.add(f,pc,0);

            for (Register r: Register.values()) {
                Label l = new Label(r.getName());
                l.setStyle("-fx-font-weight: bold;");
                middleGP.add(l,r.ordinal()+1,0);
            }
            middleGP.add(new Label("initially"),0,dynamic + 1);
            for (Register r: Register.values()) {
                Label l;
                if(!binary){
                   l = new Label(Tool.twosComplementToDecimal(r.getValue()));
                }
                else{
                   l = new Label(r.getValue());
                }
                middleGP.add(l,r.ordinal()+1,dynamic + 1);
            }
        }
        else{
            middleGP.add(new Label(String.valueOf(pc )),0,dynamic + 1);
           // middleGP.add(new Label(String.valueOf(pc)),0,pc-1);
            for (Register r: Register.values()) {
                Label l;
                if(!binary){
                   l = new Label(Tool.twosComplementToDecimal(r.getValue()));
                }
                else{
                   l = new Label(r.getValue());
                }
               // l.setStyle("-fx-font-weight: bold;");
                middleGP.add(l,r.ordinal()+1,dynamic + 1 );
            }
        }
    }

    public static void main(String[] args) {
                launch(args);
    }
}
```

2.2 Test Code

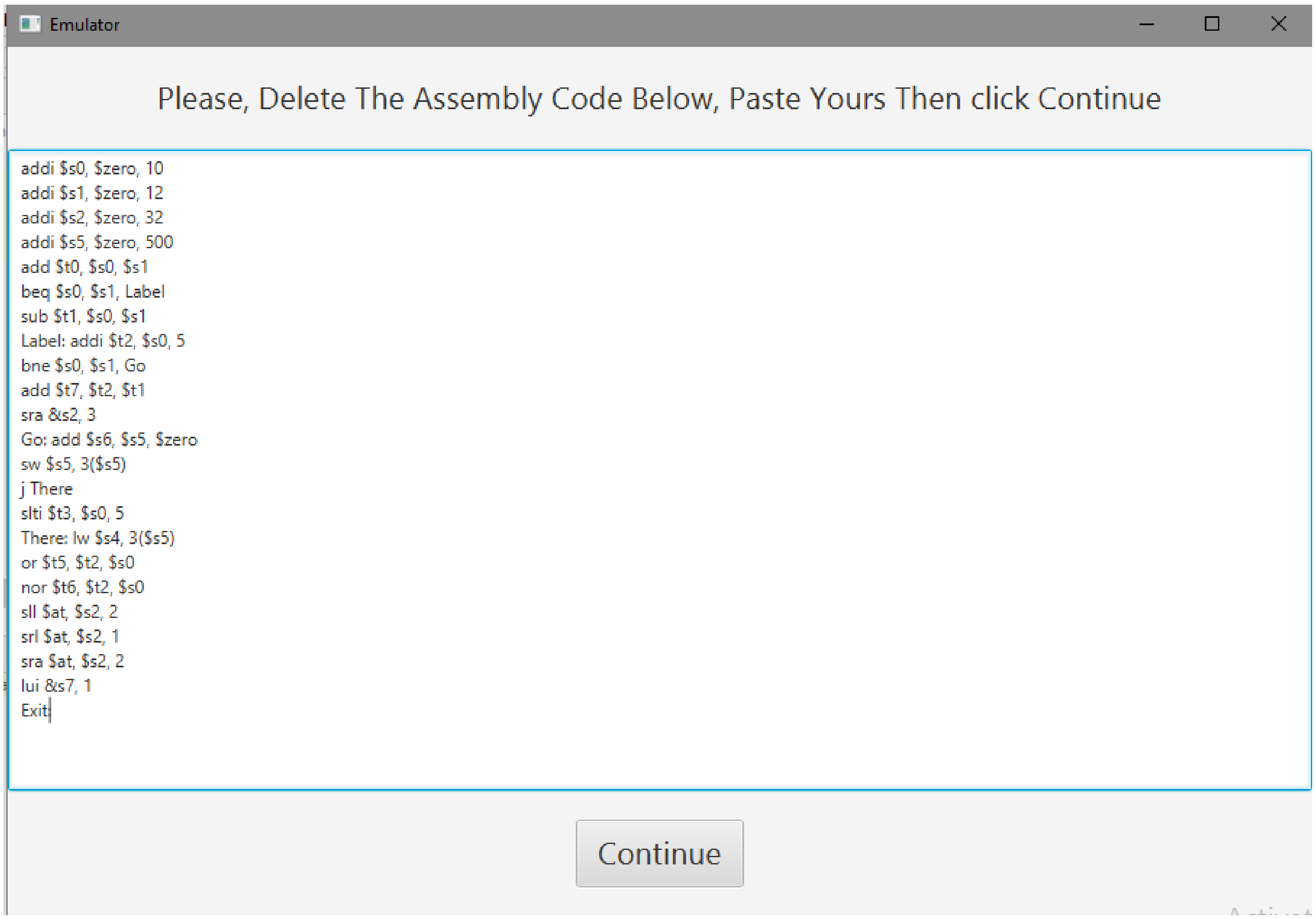
This program shall work on any code that has the nineteen instructions stated at the beginning of the report. However, the format of the Assembly code must follow strict, standard writing ways as that used to write the test code given below. For example, the name of each instruction should be in lower case letters, registers shall start with dollar sign '$', and a comma shall stick to what precedes it leaving spaces before what follows it.

The test code given below contains all the distinct instructions that need to be tested. However, the code does not reflect the implementation of any useful functionality.

```
addi $s0, $zero, 10
addi $s1, $zero, 12
addi $s2, $zero, 32
addi $s5, $zero, 500
add $t0, $s0, $s1
beq $s0, $s1, Label
sub $t1, $s0, $s1
Label: addi $t2, $s0, 5
bne $s0, $s1, Go
add $t7, $t2, $t1
sra $s2, $s1, 3
Go: add $s6, $s5, $zero
sw $s5, 3($s5)
j There
slti $t3, $s0, 5
There: lw $s4, 3($s5)
or $t5, $t2, $s0
nor $t6, $t2, $s0
sll $at, $s2, 2
srl $at, $s2, 1
sra $at, $s2, 2
lui $s7, 1
Exit:
```

2.3 Screen Shots of Line by Line Output

- First, this window appears where the user pasts the assembly code they want to test.



Please, Delete The Assembly Code Below, Paste Yours Then click Continue

```
addi $s0, $zero, 10
addi $s1, $zero, 12
addi $s2, $zero, 32
addi $s5, $zero, 500
add $t0, $s0, $s1
beq $s0, $s1, Label
sub $t1, $s0, $s1
Label: addi $t2, $s0, 5
bne $s0, $s1, Go
add $t7, $t2, $t1
sra &s2, 3
Go: add $s6, $s5, $zero
sw $s5, 3($s5)
j There
slti $t3, $s0, 5
There: lw $s4, 3($s5)
or $t5, $t2, $s0
nor $t6, $t2, $s0
sll $at, $s2, 2
srl $at, $s2, 1
sra $at, $s2, 2
lui &s7, 1
Exit:
```

- After clicking continue, the following window appears

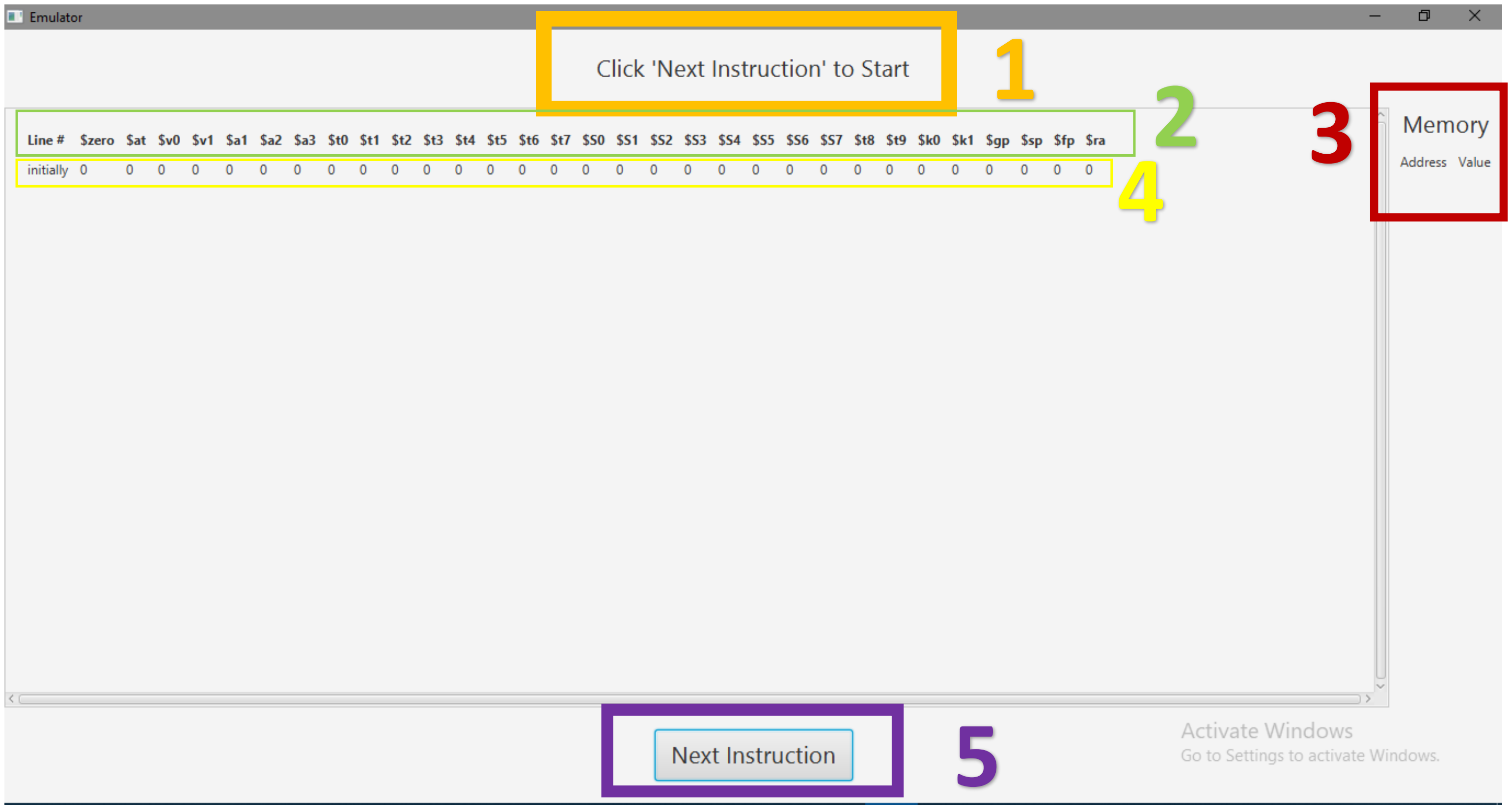
  Number 1 points to where the assembly instruction that is executed is displayed

  Number 2 points to the names of registers that are tracked

  Number 3 points to the memory that sw uses to store words

  Number 4 points to the initial values of registers (all set to zeros)

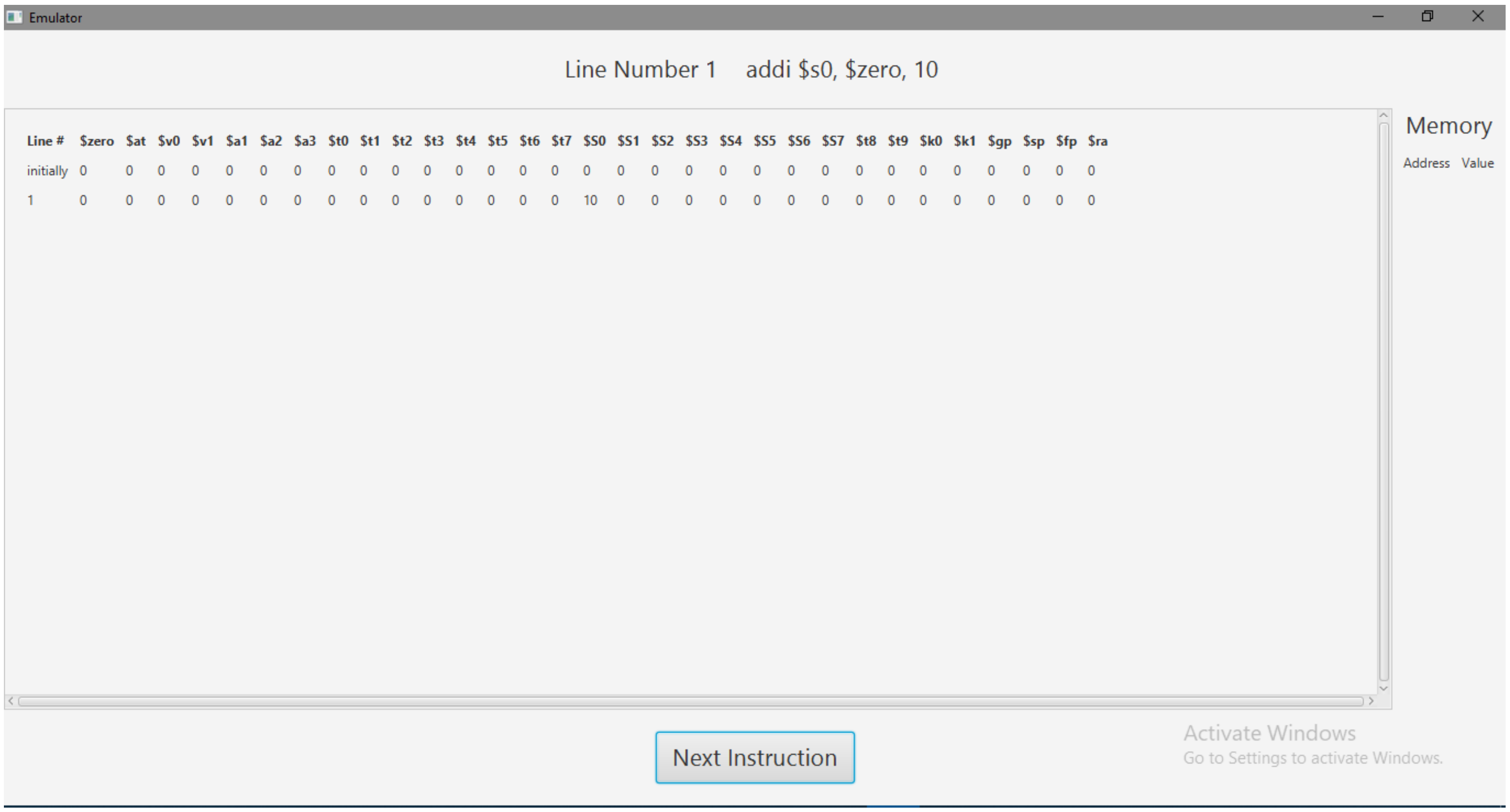
  Number 5 points to the button used to show the next instruction



Click 'Next Instruction' to Start

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

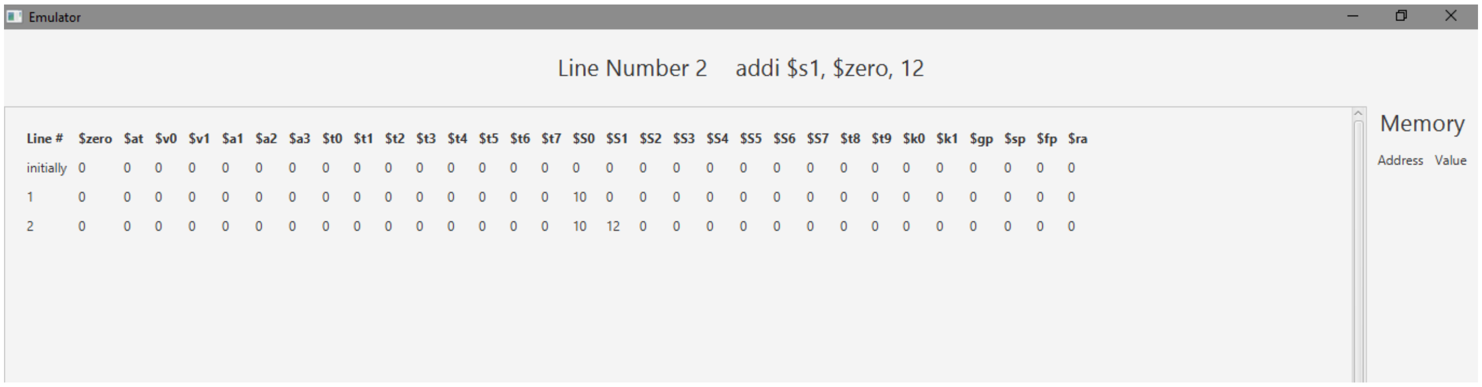
- After clicking next, the first instruction is displayed at the top of the window and a line is added showing the values of all registers after this step. The first 3 instructions are addi just to initiate registers
- Instruction 1 (line 1): addi $s0, $zero, 10
  Register $s0 is set to 10



Line Number 1 addi $s0, $zero, 10

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

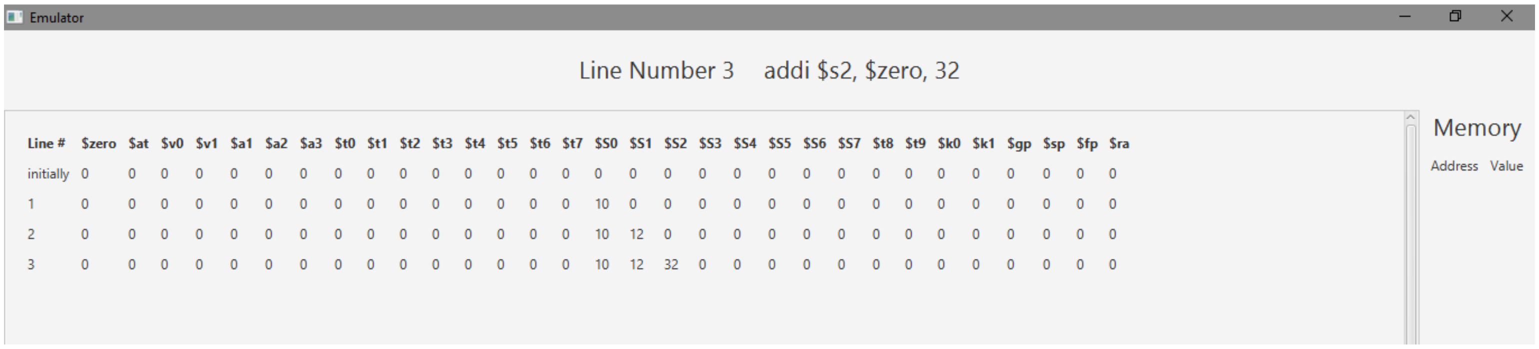
- Instruction 2 (line 2): addi $s1, $zero, 12
  Register $s1 is set to 12



Line Number 2 addi $s1, $zero, 12

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Instruction 3 (line 3): addi $s2, $zero, 32
  Register $s2 is set to 32



Emulator

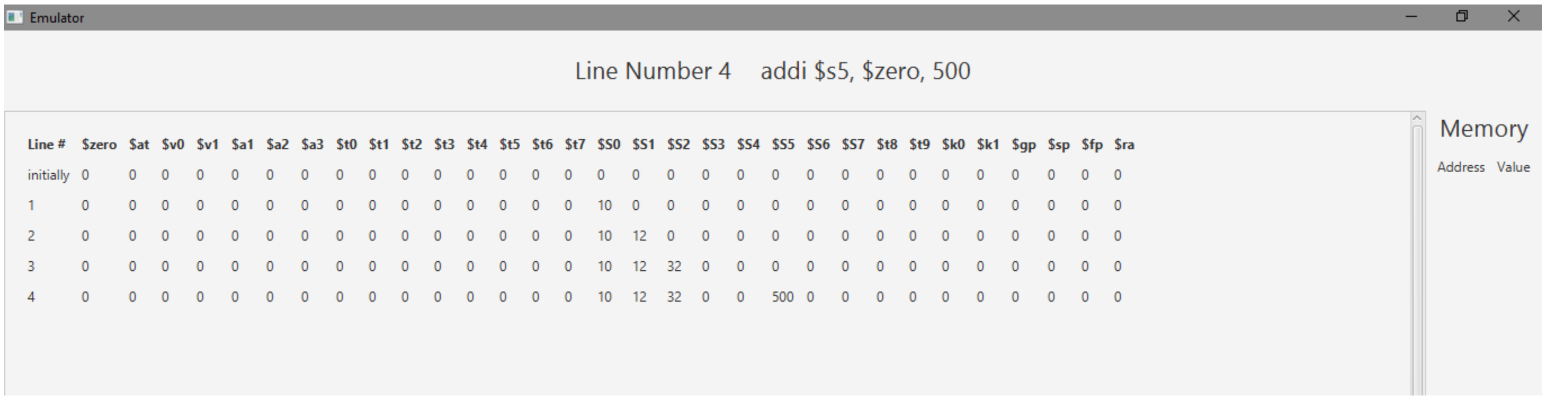
Line Number 3 addi $s2, $zero, 32

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0

Memory

Address	Value

- Instruction 4 (line 4): addi $s5, $zero, 500
  Register $s5 is set to 500



Emulator

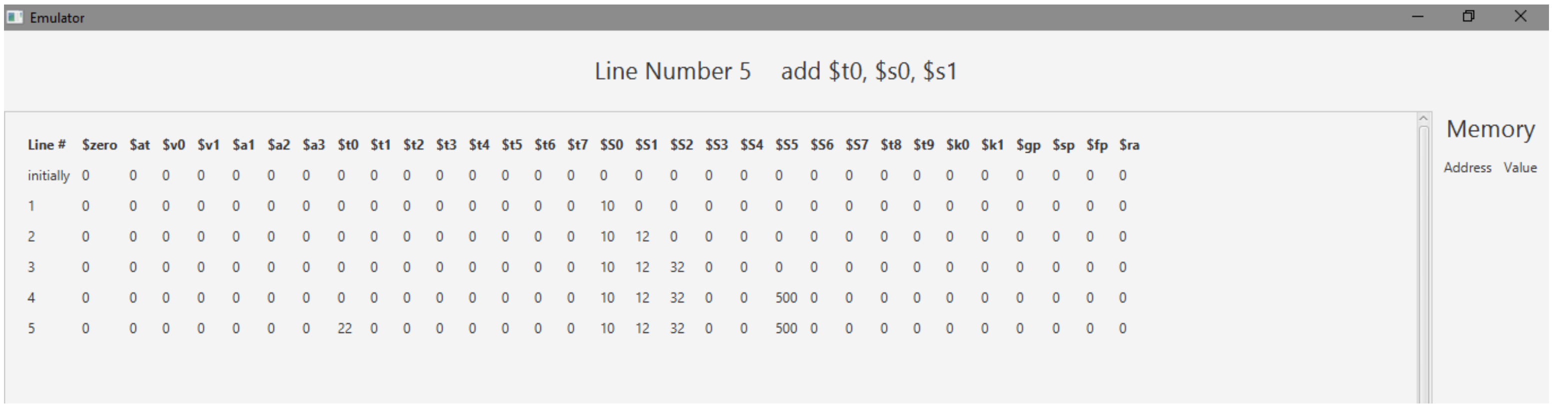
Line Number 4 addi $s5, $zero, 500

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

Memory

Address	Value

- Instruction 5 (line 5): add $t0, $s0, $s1
  Register $t0 = $s0 + $s1 = 10 + 12 = 22



Emulator

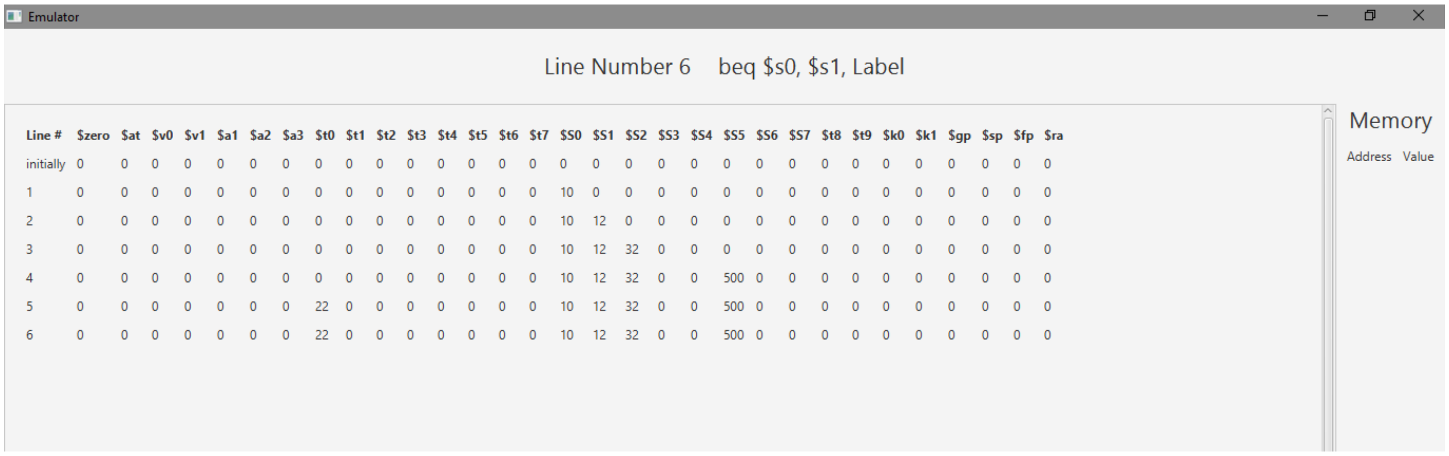
Line Number 5 add $t0, $s0, $s1

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

Memory

Address	Value

- Instruction 6 (line 6): beq $s0, $s1, Label
  Register $s1 = 12; $s0 = 10 since they are not equal the program will not branch



Emulator

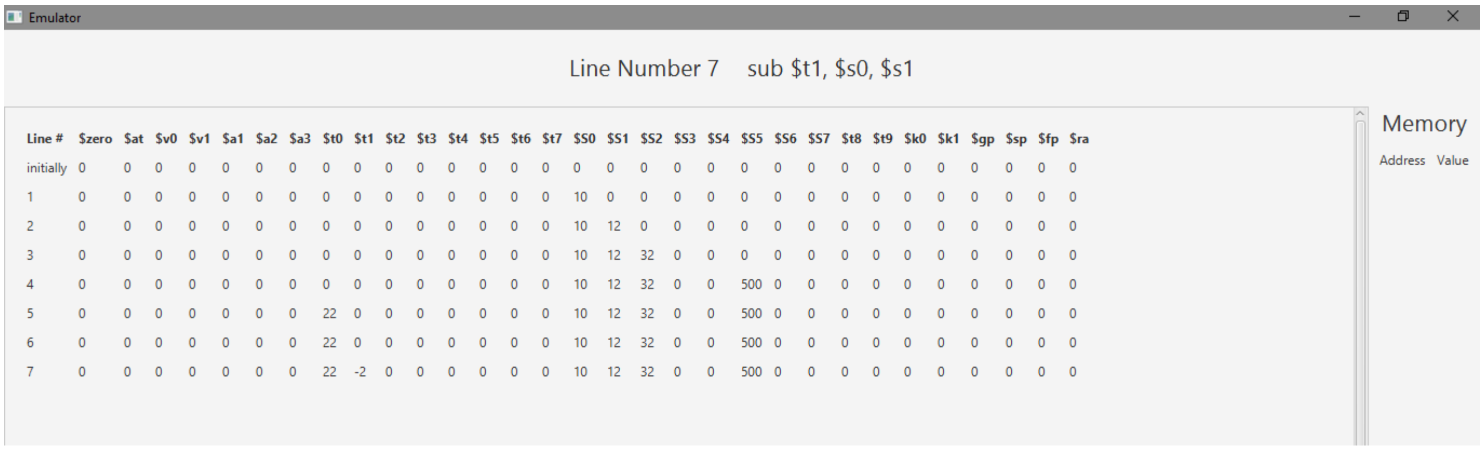
Line Number 6   beq $s0, $s1, Label

Memory

Address Value

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

- Instruction 7 (line 7): sub $t1, $s0, $s1
  Register $t1 = $s0 - $s1= 10 – 12 = -2



Emulator

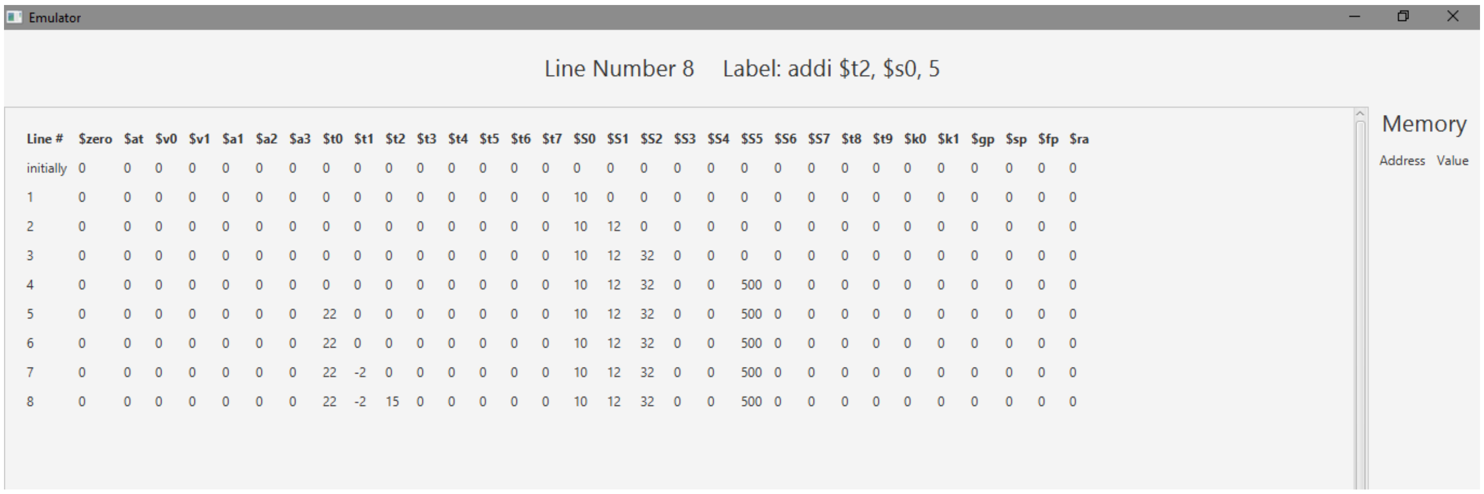
Line Number 7   sub $t1, $s0, $s1

Memory

Address Value

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

- Instruction 8 (line 8): Label: addi $t2, $s0, 5
  Register $t2 = $s0 + 5 = 10 + 5 = 15



Emulator

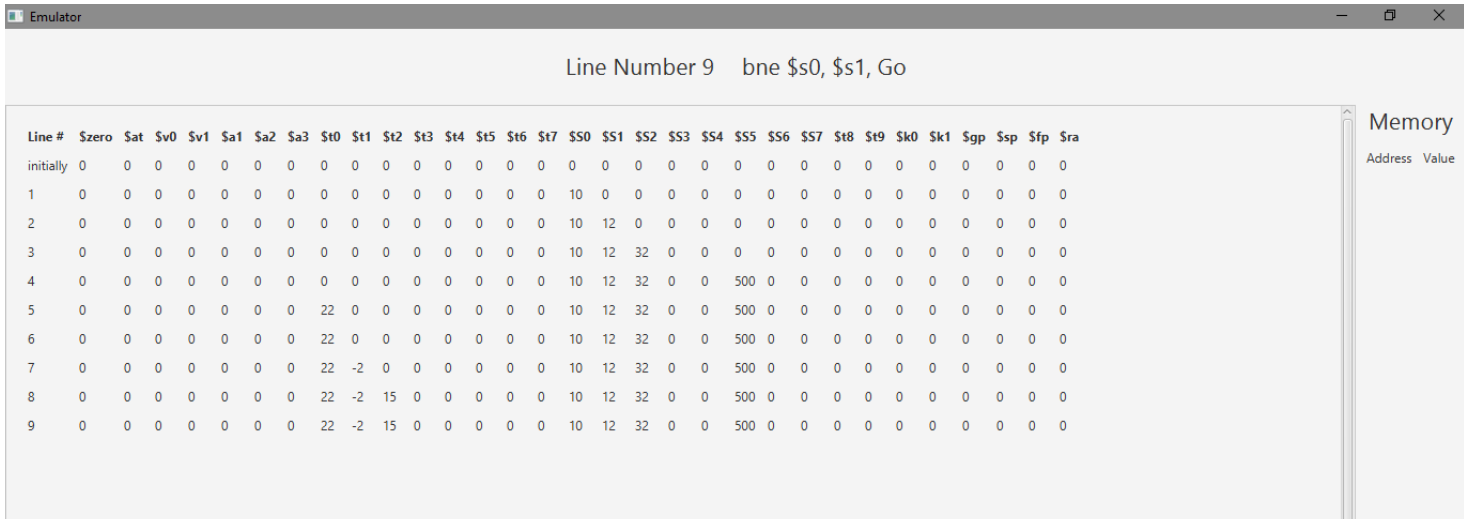
Line Number 8   Label: addi $t2, $s0, 5

Memory

Address Value

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

- Instruction 9 (line 9): bne $s0, $s1, Go
  Register $s0 = 10, $s1 = 12 since they are not equal the program branches to label Go on line 12



Emulator

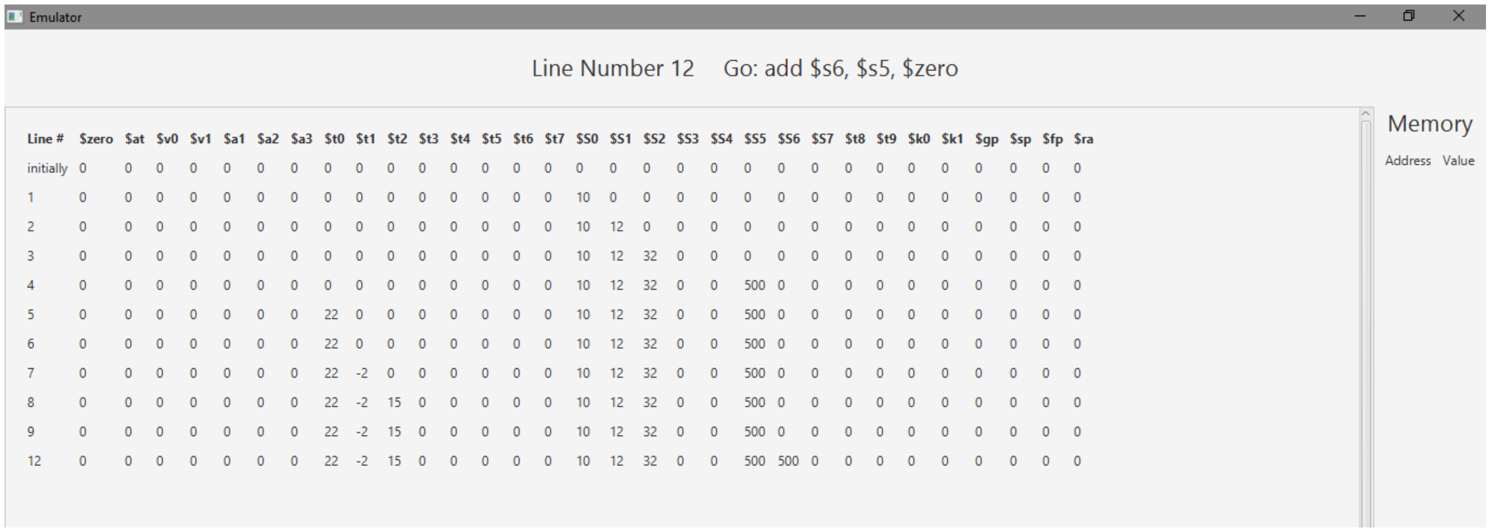
Line Number 9 bne $s0, $s1, Go

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0

Memory

Address	Value

- Instruction 10 (line 12): Go: add $s6, $s5, $zero
  Register $s6 = $s5 + $zero = 500 + 0 = 500



Emulator

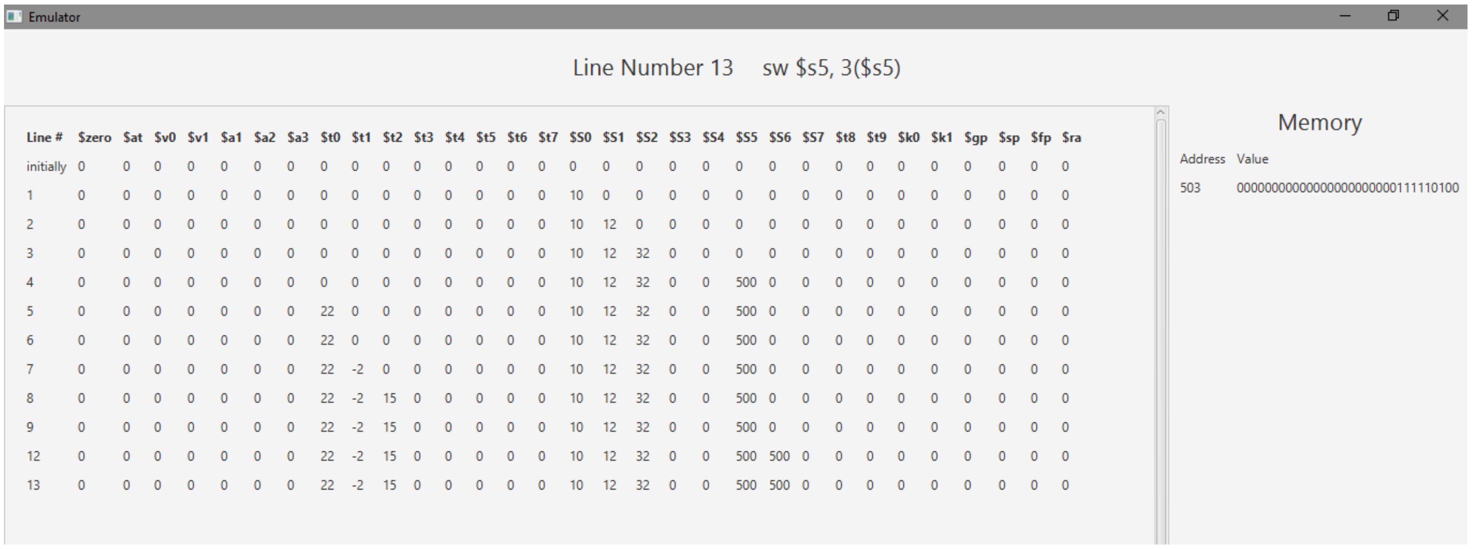
Line Number 12 Go: add $s6, $s5, $zero

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value

- Instruction 11 (line 13): sw $s5, 3($s5)
  Address = 3 + $s5 = 3 + 500 = 503; so, at address 503 the value of $s5 which is 500 is stored in memory



Emulator

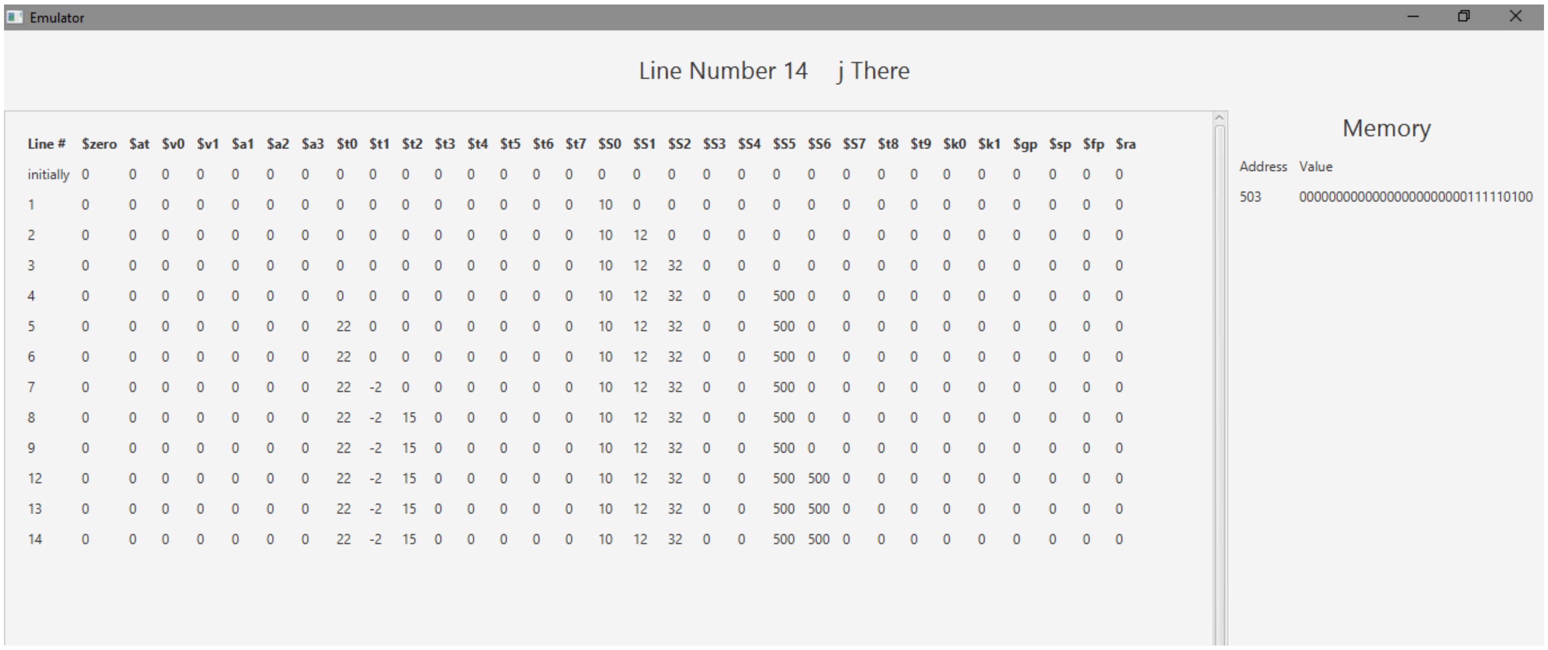
Line Number 13 sw $s5, 3($s5)

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 12 (line 14): j There
  The program branches to label There on line 16



Emulator

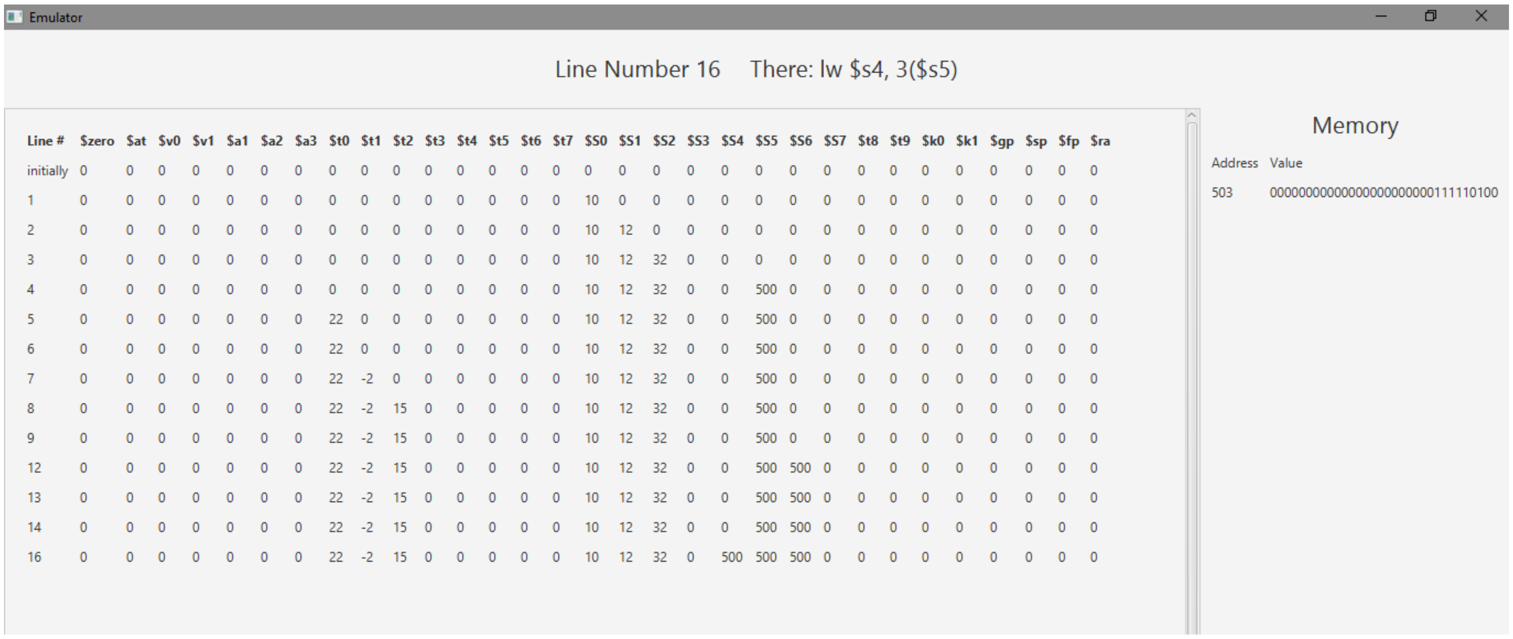
Line Number 14 j There

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 13 (line 16): There: lw $s4, 3($s5)
  Address = 3 + $s5 = 3 + 500 = 503; so, the value at address 503 which is 500 is brought back and stored at $s4



Emulator

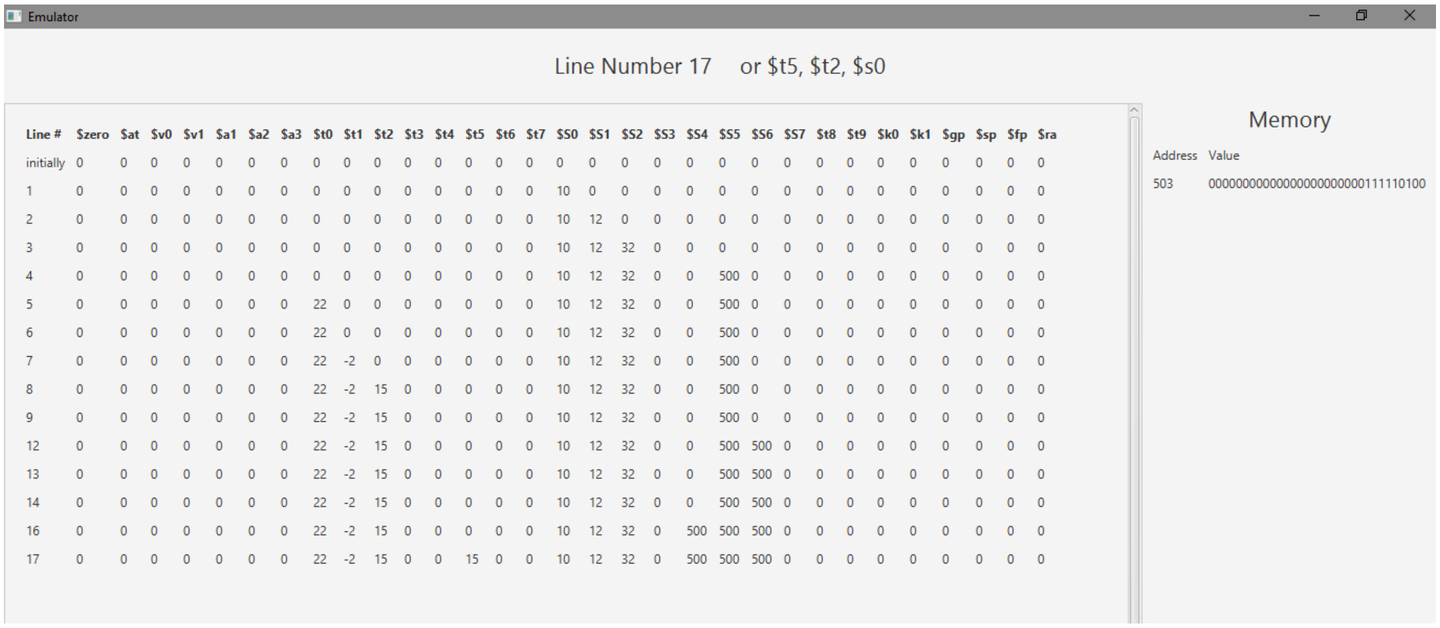
Line Number 16 There: lw $s4, 3($s5)

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 14 (line 17): or $t5, $t2, $s0
  $s0 = 10 = 00000000000000000000000000001010; or
  $t2 = 15 = 00000000000000000000000000001111;
  $t5 = 15 = 00000000000000000000000000001111;



Emulator

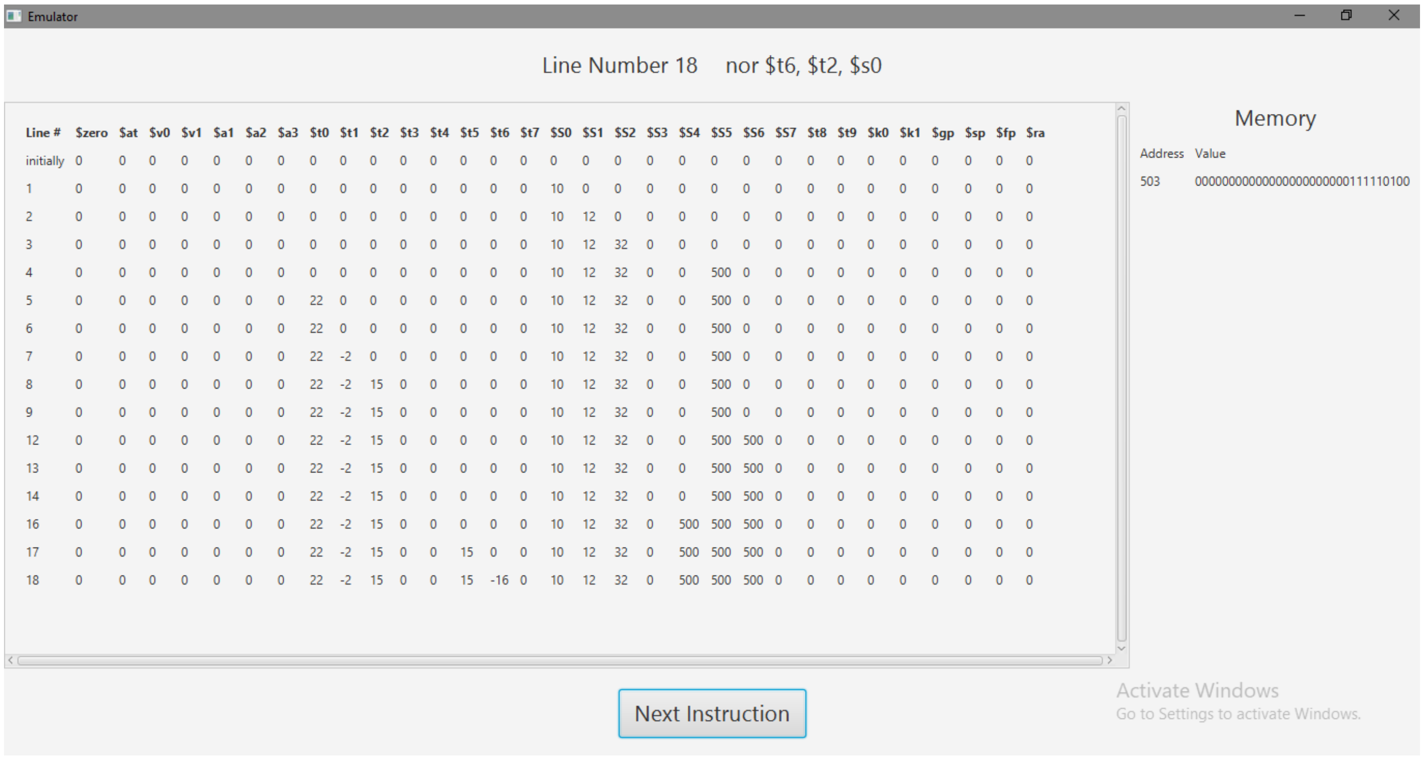
Line Number 17   or $t5, $t2, $s0

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 15 (line 18): nor $t6, $t2, $s0
  $s0 = 10 = 00000000000000000000000000001010; nor
  $t2 = 15 = 00000000000000000000000000001111;
  $t6 = -16 = 11111111111111111111111111110000;



Emulator

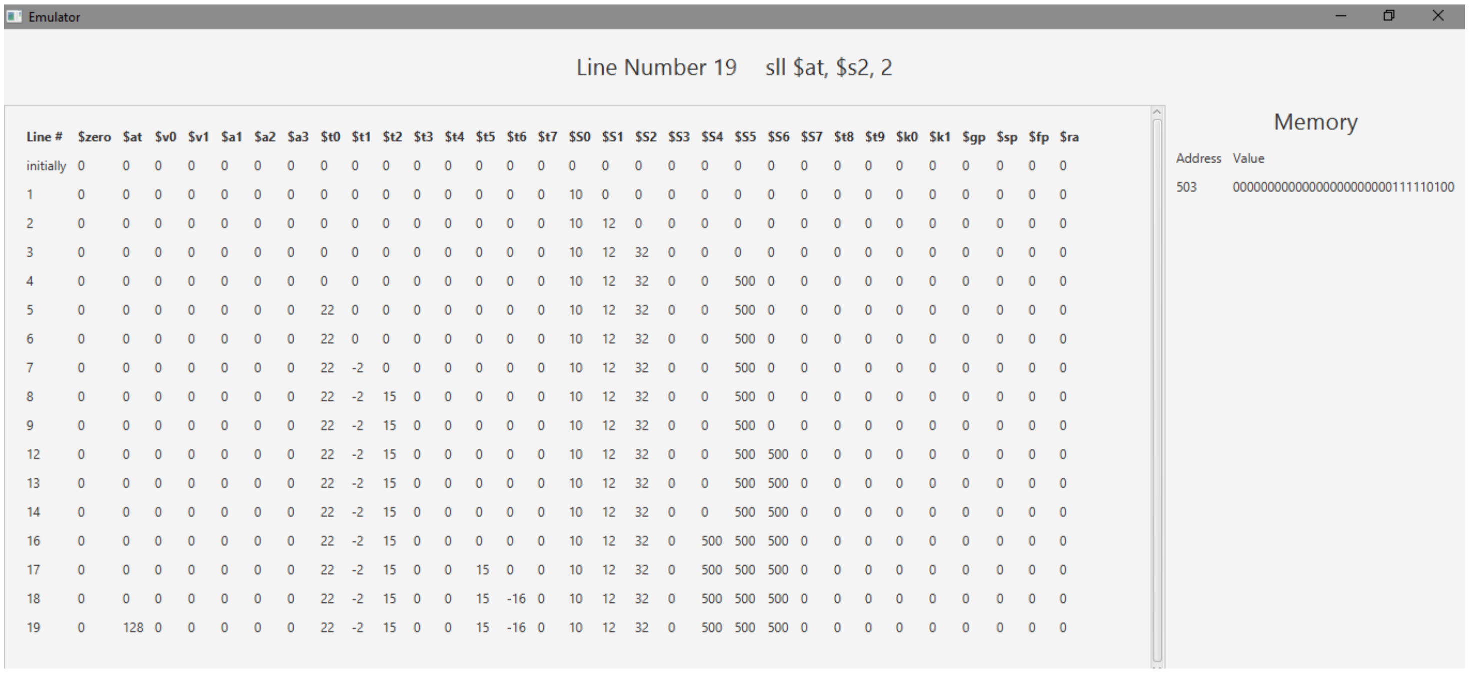
Line Number 18   nor $t6, $t2, $s0

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 16 (line 19): sll $at, $s2, 2
  $s2 = 10 = 00000000000000000000000000100000;
  $at = 128 = 00000000000000000000000010000000;


Emulator

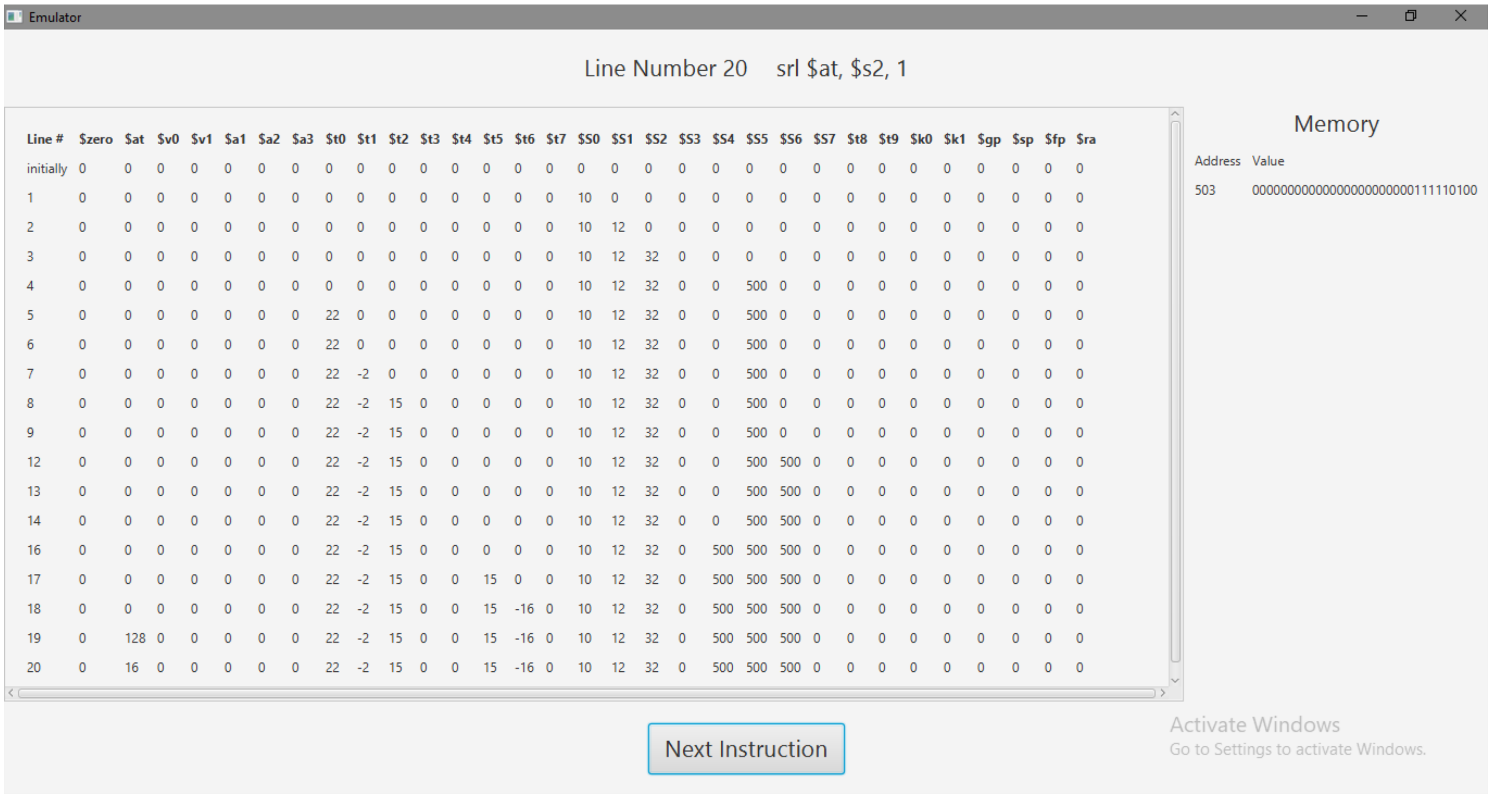
Line Number 19 sll $at, $s2, 2

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
19	0	128	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 17 (line 20): srl $at, $s2, 1
  $s2 = 10 = 00000000000000000000000000100000;
  $at = 16 = 00000000000000000000000000010000;


Emulator

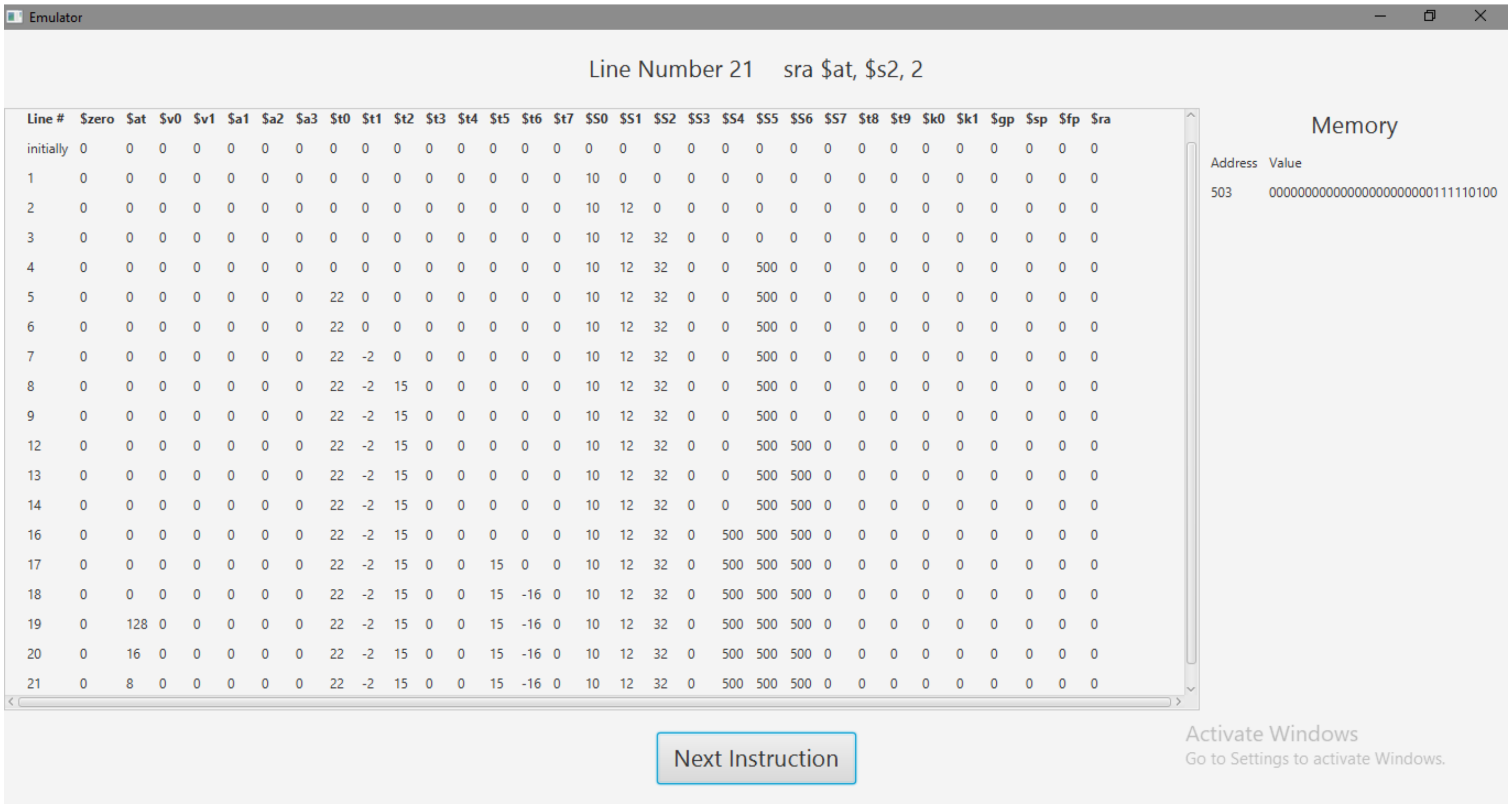
Line Number 20 srl $at, $s2, 1

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
19	0	128	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
20	0	16	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 18 (line 21): sra $at, $s2, 2
  $s2 = 10 = 00000000000000000000000000100000;
  $at = 8 = 00000000000000000000000000001000;


Emulator

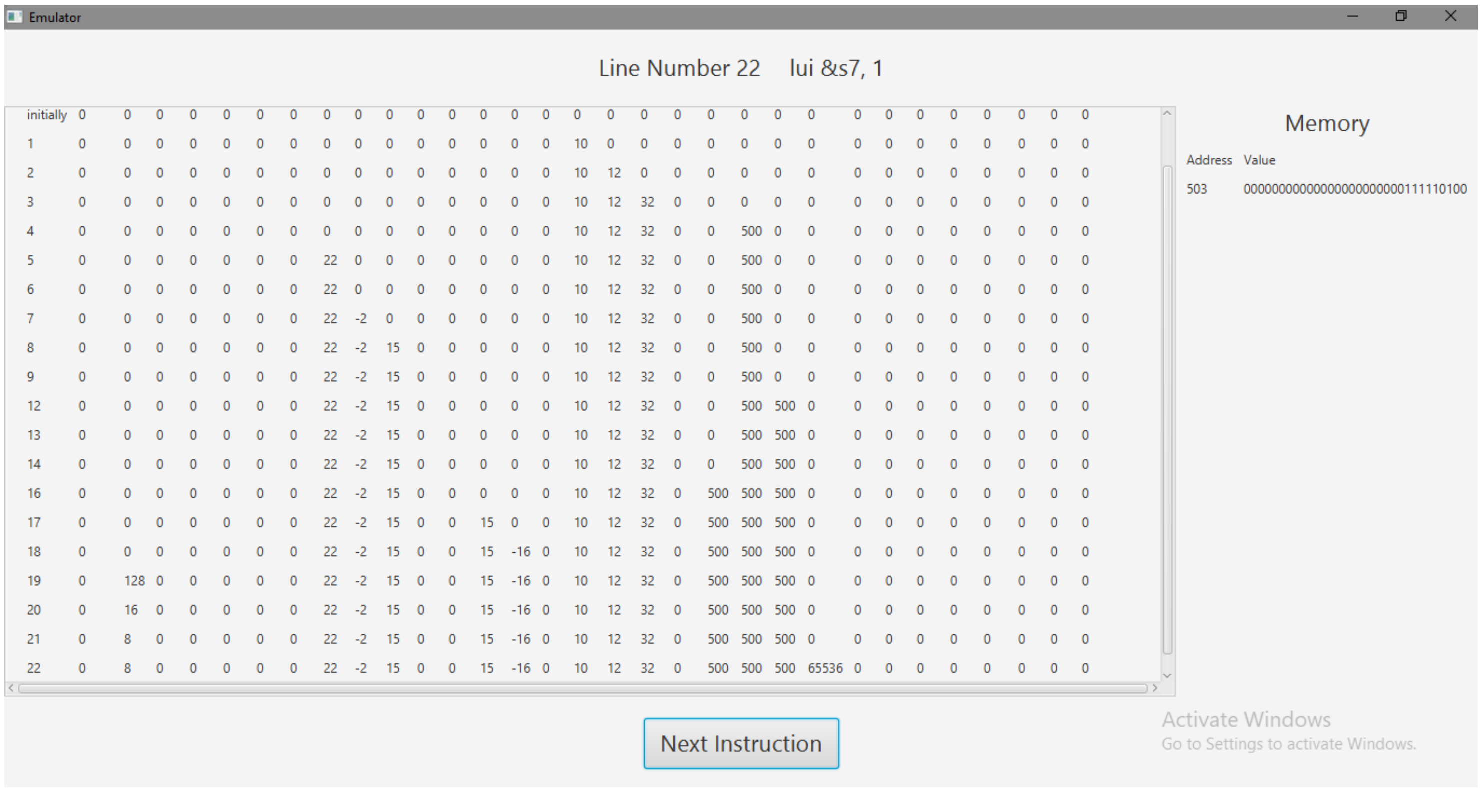
Line Number 21 sra $at, $s2, 2

Line #	$zero	$at	$v0	$v1	$a1	$a2	$a3	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$S0	$S1	$S2	$S3	$S4	$S5	$S6	$S7	$t8	$t9	$k0	$k1	$gp	$sp	$fp	$ra
initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
19	0	128	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
20	0	16	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
21	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 19 (line 22): lui &s7, 1
  $s7 = 65536 = 00000000000000010000000000000000;



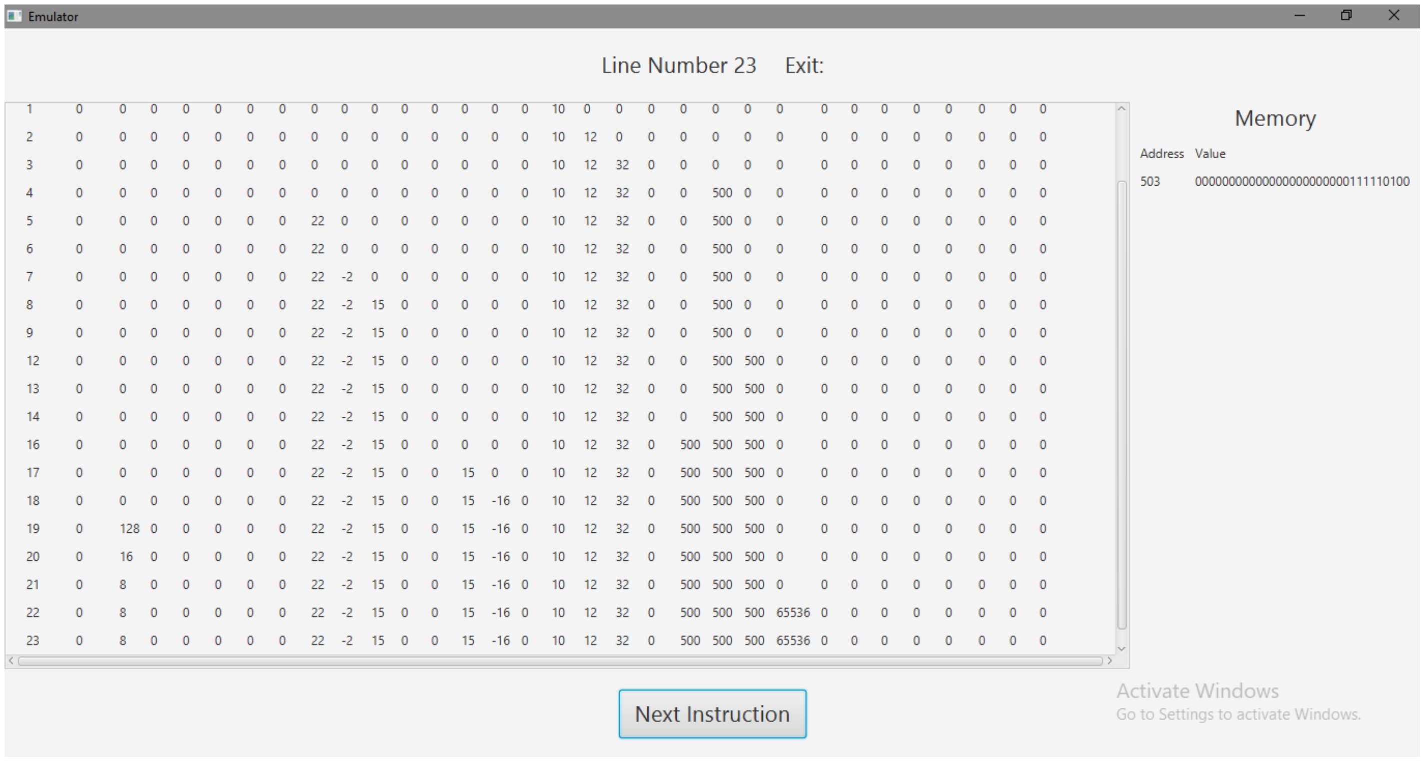
Line Number 22 lui &s7, 1

initially	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
19	0	128	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
20	0	16	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
21	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
22	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	65536	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

- Instruction 20 (line 23): Exit:
  Here nothing happens



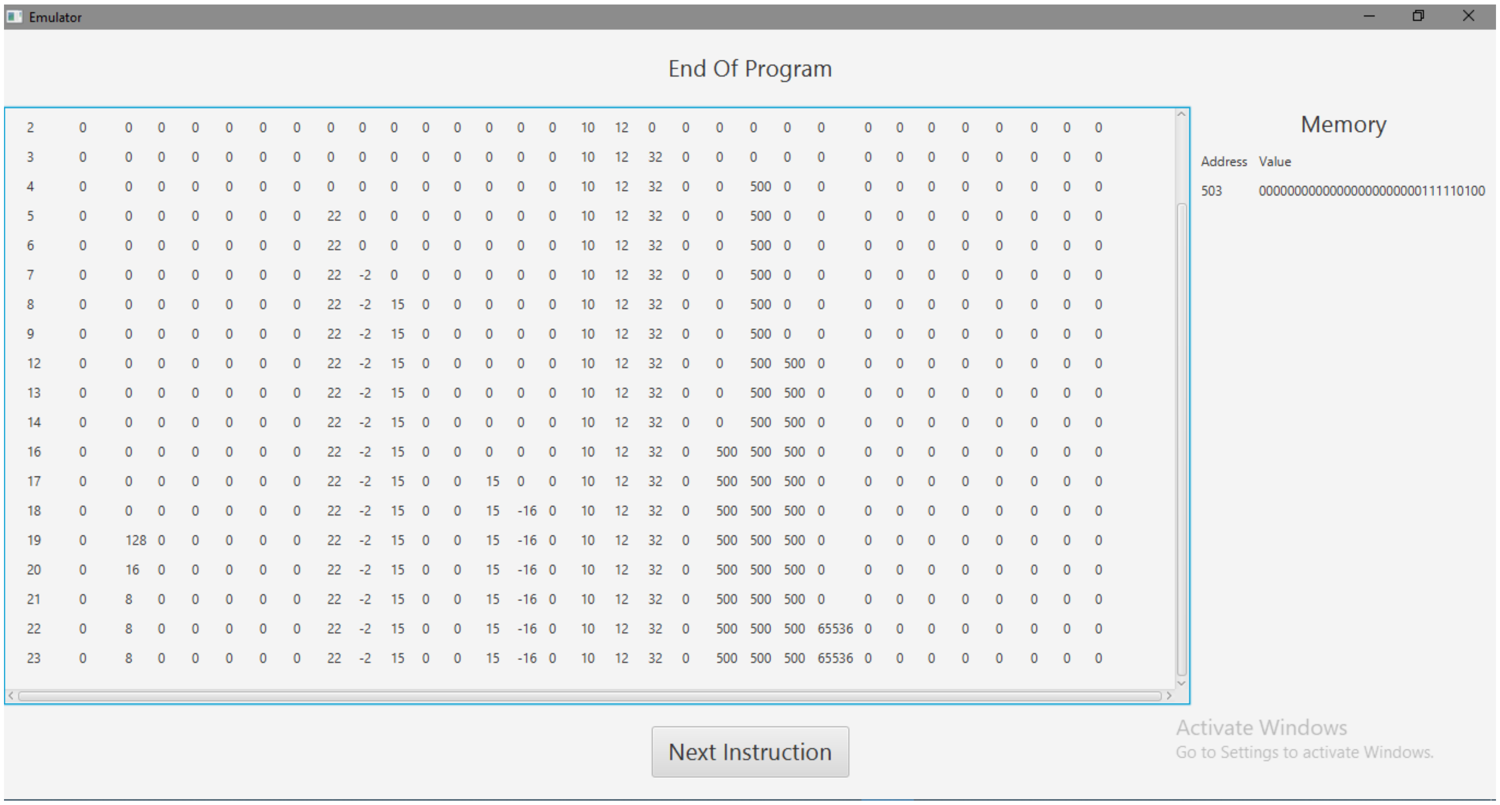
Line Number 23 Exit:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	22	-2	0	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	0	500	500	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	22	-2	15	0	0	0	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	22	-2	15	0	0	15	0	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
19	0	128	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
20	0	16	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
21	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	0	0	0	0	0	0	0	0	0
22	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	65536	0	0	0	0	0	0	0	0
23	0	8	0	0	0	0	0	22	-2	15	0	0	15	-16	0	10	12	32	0	500	500	500	65536	0	0	0	0	0	0	0	0

Memory

Address	Value
503	00000000000000000000000111110100

Finally, the program ends



End Of Program

```
2    0  0    0  0  0  0  0  0   0   0   0  0  0   0   0  10  12  0   0  0    0    0    0      0  0  0  0  0  0  0  0
3    0  0    0  0  0  0  0  0   0   0   0  0  0   0   0  10  12  32  0  0    0    0    0      0  0  0  0  0  0  0  0
4    0  0    0  0  0  0  0  0   0   0   0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
5    0  0    0  0  0  0  0  22  0   0   0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
6    0  0    0  0  0  0  0  22  0   0   0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
7    0  0    0  0  0  0  0  22  -2  0   0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
8    0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
9    0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  0    500  0    0      0  0  0  0  0  0  0  0
12   0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  0    500  500  0      0  0  0  0  0  0  0  0
13   0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  0    500  500  0      0  0  0  0  0  0  0  0
14   0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  0    500  500  0      0  0  0  0  0  0  0  0
16   0  0    0  0  0  0  0  22  -2  15  0  0  0   0   0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
17   0  0    0  0  0  0  0  22  -2  15  0  0  15  0   0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
18   0  0    0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
19   0  128  0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
20   0  16   0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
21   0  8    0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  0      0  0  0  0  0  0  0  0
22   0  8    0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  65536  0  0  0  0  0  0  0  0
23   0  8    0  0  0  0  0  22  -2  15  0  0  15  -16 0  10  12  32  0  500  500  500  65536  0  0  0  0  0  0  0  0
```

Memory

Address	Value
503	00000000000000000000000111110100