COMPUTER ORGANIZATION
EECS 2021

PROJECT 1 REPORT
MIPS SIMULATOR

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

X ____________________

Camillo John (CJ) D’Alimonte
212754396
Lab Section 2
10/21/14
Professor Aboelaze

Abstract:

This project was created to demonstrate my cumulative knowledge on Assembly Language. For this assignment, I will have to create a MIPS Simulator in a High-Level Language (Java) that is capable of reading different instructions one by one and executing them. The Execution of these instructions means I will be changing the contents of the registers and memory. Throughout the making of this assignment, I will continue to familiarize myself with important MIPS operations and their functions as well as the concept of memory/register allocation. By the end of the project, I will have used my knowledge of MIPS in order to create a simple simulator in a high level language.

Equipment Used:

This lab was done through the UNIX environment at the Lassonde Lab. The entire project was written in Java through the Eclipse IDE.

Methods/Procedures:

The first challenge of this project was converting the hexadecimal values of the text file into binary code. An example of the input file is given below.

```
0
26310005
26520007
001290c0
02329821
0271a024
```

In order to read the input from the file, I needed to create a scanner that read each line of text from the file. I had to pay special attention to the first line of the file as the value dictated how many times (and when) I would output the values of the registers. If the mode was 0, the simulator would simulate the program until the end and then print the contents of the 32 registers \$0 - \$31. If the mode was 1, the simulator would print the contents of the 32 registers after it executes each instruction. In this case, the output method would be called after each instruction is read from the file. An iterator was used to iterate through each separate line in the file. The code would be in a try-catch statement as I wanted to throw an exception if the file's input was not readable, for whatever reason.

**Assume memory is the name of an ArrayList acting as my memory and the method RegistertoString() outputs the resulting registers.

Pseudo Code	try
	inputScanner = new Scanner(new File (INPUTFILE));
	mode → 0;
	while inputScanner.hasNext()
	memory.add(inputScanner.nextInt(16));
	iterator → *memory*.iterator();
	if (iterator.hasNext())

	mode → iterator.next();
	while (iterator.hasNext())
	Convert(iterator.next());
	if ((mode → 1) && (iterator.hasNext())) *RegistertoString*();
	else *RegistertoString*();
	
	catch (Exception e)
	e.printStackTrace();

Now that I have my input, I need to convert it to binary. For this, I used Java's built in toBinaryString() method. One important thing to note is if the binary string was not 32 bits long, I would need to add the necessary number of 0's until it reached 32 bits.

Pseudo Code	StringBuffer STRING → new StringBuffer(Integer.*toBinaryString*(STRING));
	while (STRING.length() < 32)
	STRING.insert(0, 0);

Now that I had the correct binary strings to work with, I had to split them into the correct instruction formats. The first 6 digits of the string represented the opcode. If all 6 bits were 0, then the binary string represented an R-Type Register. If that was the case, the code would be divided as follows:

** Assume that "in" is the binary string

Pseudo Code	rt → Integer.*parseInt*(in.substring(11, 16), 2);
	rd → Integer.*parseInt*(in.substring(16, 21), 2);
	rs → Integer.*parseInt*(in.substring(6, 11), 2);
	shamt → Integer.*parseInt*(in.substring(21, 26), 2);
	funct → in.substring(26, 32);

Since we now know that the opcode is 000000 and that it's an R-Type Register, we must look at the value of the funct. The MIPS Green Card gave me the correct conversion values for each funct. For example, a funct value of 100100 was the AND command. I decided to use an if statement to test each funct value. If there was a match, then I would use the equivalent Java operators to calculate the correct register value for that command. For example, the AND command would be register [rd] = (int) register[rs] & (int) register[rt]. Below are all my conversions:

Pseudo Code	if (funct.equals("000000"))
	register[rd] → (int) *register*[rt] << shamt;
	if (funct.equals("000010"))
	register[rd] → (int) *register*[rt] >> shamt;
	if (funct.equals("101010"))
	register[rd] → ((int) *register*[rs] < (int) *register*[rt]) ? 1: 0;
	if (funct.equals("101011"))

	register[rd] → *register*[rs] < *register*[rt] ? 1 : 0;
	if (funct.equals("100000"))
	register[rd] →(int) *register*[rs] + (int) *register*[rt];
	if (funct.equals("100001"))
	register[rd] → *register*[rs] + *register*[rt];
	if (funct.equals("100100"))
	register[rd] → (int) *register*[rs] & (int) *register*[rt];
	if (funct.equals("100111"))
	register[rd] → ~((int) *register*[rt] | (int) *register*[rs]);

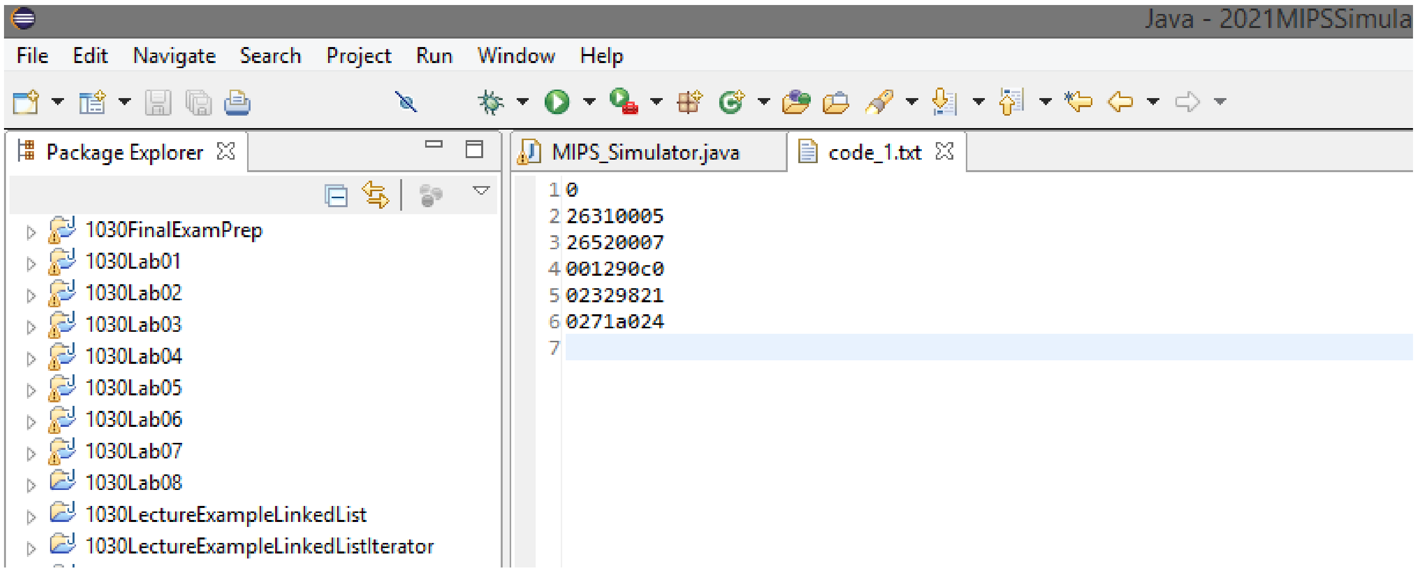
Any MIPS commands that used an immediate value would be stored in an I-Type Register. The process of converting the commands is very similar to the code mentioned above. The biggest difference is that the opcode value was no longer 000000.

Now that I have finished converting the values and storing the values into the register, I need to write a method that outputs the strings according to the lab instructions. I stored all 32 registers in a static array. I iterated through the loop to make sure that each binary value was converted to hexadecimal and that each hexadecimal value had 8 bits. If not, I would add the adequate number of 0's to the value. Once this was all complete, I had to output the value of each of the 32 registers. No pseudo code will be provided since all the code consisted of were simple print statements. 4 registers were printed on each line (4x8). 40 equal signs were outputted at the end of the file.

Results:

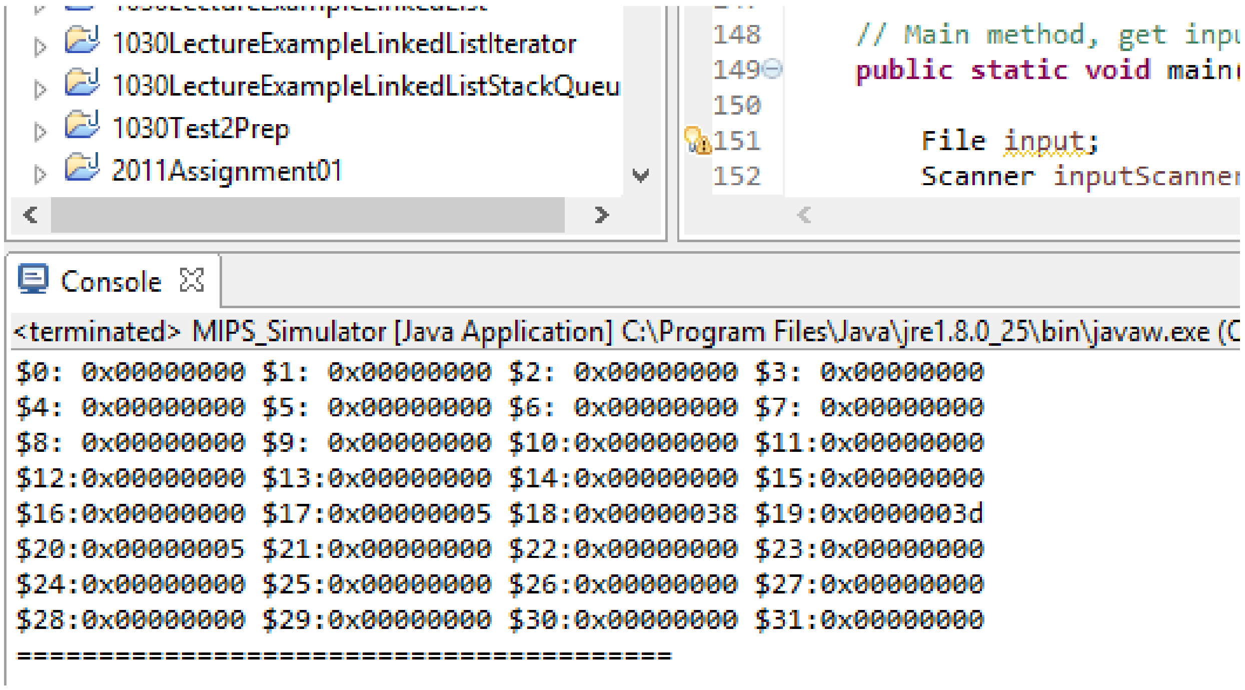
The following screenshots describe the various results I recorded from my test case of modes 0 and 1.

Input File:



```
0
26310005
26520007
001290c0
02329821
0271a024
```

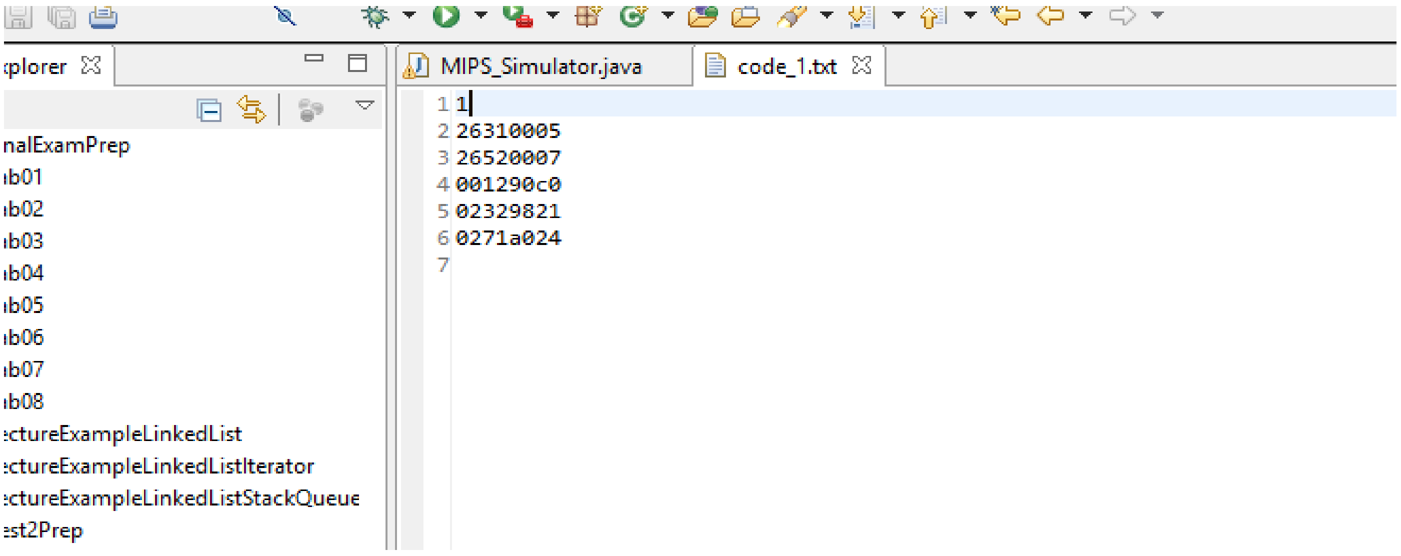
Output:


1030LectureExampleLinkedListIterator
1030LectureExampleLinkedListStackQueu
1030Test2Prep
2011Assignment01
Console

```
    // Main method, get inpu
    public static void main(
    
        File input;
        Scanner inputScanner
```

```
<terminated> MIPS_Simulator [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (C
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000038 $19:0x0000003d
$20:0x00000005 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
```

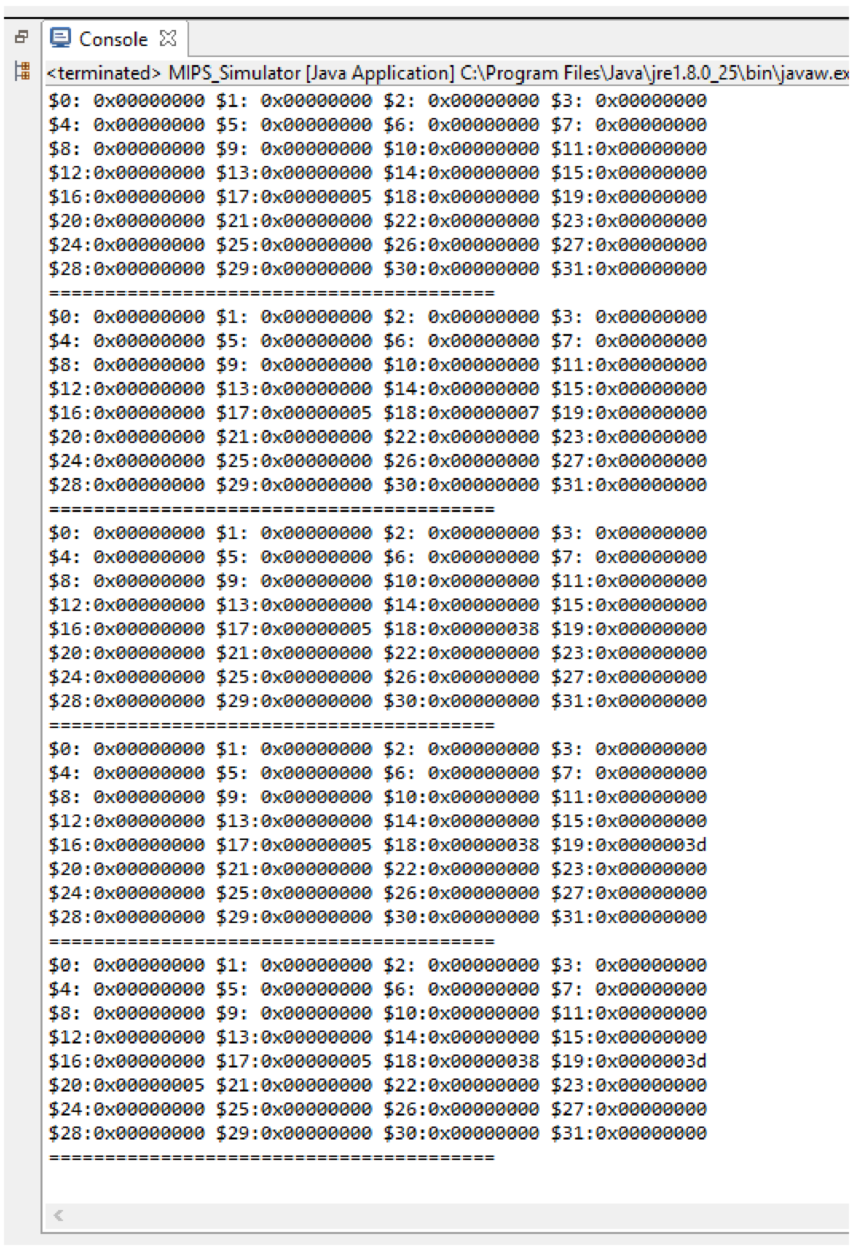
Input File:



```
1
26310005
26520007
001290c0
02329821
0271a024

```

Ouput:



```
<terminated> MIPS_Simulator [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.ex
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000000 $19:0x00000000
$20:0x00000000 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000007 $19:0x00000000
$20:0x00000000 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000038 $19:0x00000000
$20:0x00000000 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000038 $19:0x0000003d
$20:0x00000000 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
$0: 0x00000000 $1: 0x00000000 $2: 0x00000000 $3: 0x00000000
$4: 0x00000000 $5: 0x00000000 $6: 0x00000000 $7: 0x00000000
$8: 0x00000000 $9: 0x00000000 $10:0x00000000 $11:0x00000000
$12:0x00000000 $13:0x00000000 $14:0x00000000 $15:0x00000000
$16:0x00000000 $17:0x00000005 $18:0x00000038 $19:0x0000003d
$20:0x00000005 $21:0x00000000 $22:0x00000000 $23:0x00000000
$24:0x00000000 $25:0x00000000 $26:0x00000000 $27:0x00000000
$28:0x00000000 $29:0x00000000 $30:0x00000000 $31:0x00000000
========================================
```

Discussion:

For this lab project, I needed to create a MIPS Simulator that would execute a set of Assembly commands and store the resulting values in 32 registers. This project was independent of all the previous labs I have done but it relied on my knowledge thus far in the course. I divided the project into numerous small tasks. Such divisions are listed below:

- Read input from file, convert from hexadecimal to binary
- Add adequate number of 0's to set each binary value to 32 bits
- Split strings into appropriate sections (R & I Type Registers)

- Use the encoding info (MIPS Green Card) to match commands with correct opcode
- Use equivalent Java operators (ie >>, <, +) to manipulate registers
- Note: Any commands with an immediate needs to be stored in an I-Type Register
- Once binary values are calculated, convert back to hex (add adequate 0's)
- Output 32 register values to the console

Overall, this project was a little difficult, especially at the beginning. Although I knew how to do the project theoretically, I struggled actually writing it in code. I ran into some trouble converting the hexadecimal values of the file into binary but the biggest problem was just arranging the different values with their subsequent commands.

Correctly using the Java operators, such as the shift operators, proved to be a challenge. I ran into some issues remembering how to use the shift operators and understanding how they actually worked. Converting the values back into hexadecimal and outputting the values proved to be fairly simple. This was the first time I was using the MIPS Green Card which proved to be extremely useful in helping me convert the funct/opcode values into specific binary commands.

Although I ran into some difficulties, I did learn a lot from this project and it did help me cement my knowledge in MIPS and how memory/registers work.

Conclusion:

This project allowed me the opportunity to demonstrate my cumulative knowledge in both MIPS and Java. In building this MIPS Simulator, I had to use both my knowledge of high and low level programming techniques in order to create high level software that was able to simulate the behaviour of a low level application. The project involved many of the different MIPS commands that I have learned over the last few labs, including arithmetic commands (add, addi) and shift operators (sll, srl, etc.). All in all, this project gave me an opportunity to demonstrate all the skills I have learned in MIPS and to truly understand how binary values are stored in memory/registers.

References:

Some external references were used to complete this lab. The links below reference the MIPS Green Card and a MIPS encoding chart.

https://www.student.cs.uwaterloo.ca/~isg/res/mips/opcodes

http://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf

Appendix:

The source code for my Java program is below:

```
package SimulatorMIPS;

import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

/* MAIN OBJECTIVES:
 *
 * Read input from file, convert from hexadecimal to binary
 * Add adequate number of 0's to set each binary value to 32 bits
 * Split strings into appropriate sections (R vs I Type Registers)
 * Use the encoding info (MIPS Green Card) to match commands with correct opcode
 */

public class MIPS_Simulator {

	private static ArrayList<Integer> memory = new ArrayList<Integer>();

	private static int[] register = new int[] { 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, 0, };
		// array of 32 ints for register


	private static void Convert(Integer i) {

		StringBuffer in = new StringBuffer(Integer.toBinaryString(i));
		while (in.length() < 32) { // add adequate number of 0's
			in.insert(0, 0);
		}
		String opcode = in.substring(0, 6);
		int shamt, immediate, rt, rs, rd = 0;
		String funct, imm;

		switch (opcode) {
		case "000000":
			rt = Integer.parseInt(in.substring(11, 16), 2);
			rd = Integer.parseInt(in.substring(16, 21), 2);
			rs = Integer.parseInt(in.substring(6, 11), 2);
			shamt = Integer.parseInt(in.substring(21, 26), 2);
			funct = in.substring(26, 32);


			/*
			 *                         R-Type Register
			 *   -----------------------------------------------------------
			 *   | opcode (0,6) --> rs (6,11) --> rt (11,16) --> rd (16,21) -
			 -> shamt (21,26) --> funct (26,32)   |

			 *   -----------------------------------------------------------
			 */


			if (funct.equals("000000")) {   // shift left logically
				register[rd] = (int) register[rt] << shamt;
			}
			if (funct.equals("000010")) {   // shift right logically
				register[rd] = (int) register[rt] >> shamt;
```

```
                        }

                        if (funct.equals("101010")) {   // set less than
                                register[rd] = ((int) register[rs] < (int) register[rt]) ?
1
                                                : 0;
                        }

                        if (funct.equals("101011")) {   // set less than unsigned
                                register[rd] = register[rs] < register[rt] ? 1 : 0;
                        }

                        if (funct.equals("100000")) {   // add
                                register[rd] = (int) register[rs] + (int) register[rt];
                        }
                        if (funct.equals("100001")) {    // add unsigned
                                register[rd] = register[rs] + register[rt];
                        }

                        if (funct.equals("100100")) {    // and
                                register[rd] = (int) register[rs] & (int) register[rt];
                        }
                        if (funct.equals("100111")) {    // nor (not or)
                                register[rd] = ~((int) register[rt] | (int) register[rs]);
                        }

                        if (funct.equals("001100")) {    // syscall
                                System.out.println("Syscall - Program Ended");
                                return;
                        }
                        break;

                case "001001": // add immediate unsigned

                        imm = in.substring(16, 32);    // I TYPE REGISTER
                        immediate = Integer.parseInt(imm, 2);
                        rt = Integer.parseInt(in.substring(11, 16), 2);
                        rs = Integer.parseInt(in.substring(6, 11), 2);
                        register[rt] = register[rs] + immediate;
                        break;

                case "001010": // set less than immediate

                        imm = in.substring(16, 32);   // I TYPE REGISTER
                        if (imm.charAt(0) == 1)// sign extension for immediate
                                imm = "1111111111111111" + imm;
                        immediate = Integer.parseInt(imm, 2);
                        rt = Integer.parseInt(in.substring(11, 16), 2);
                        rs = Integer.parseInt(in.substring(6, 11), 2);
                        register[rt] = ((int) register[rs] < immediate) ? 1 : 0; // check
if less than

                        break;

                case "001111": // lui
                        immediate = Integer.parseInt(in.substring(16, 32), 2);
```

```
				rt = Integer.parseInt(in.substring(11, 16), 2);
				register[rt] = immediate << 16;  // shift
				break;
			}

		}

		private static void RegistertoString() {


			String[] reg = new String[32]; // array of 32 registers
			for (int i = 0; i < 32; i++) {
				reg[i] = Integer.toHexString((int) register[i]); // convert to
hexadecimal
				while (reg[i].length() < 8) {
					reg[i] = "0" + reg[i];   // add adequate 0's until 8 bits
				}
				reg[i] = "0x" + reg[i];
			}


			System.out.println("$0: " + reg[0] + " " + "$1: " + reg[1] + " "
					+ "$2: " + reg[2] + " " + "$3: " + reg[3]);
			System.out.println("$4: " + reg[4] + " " + "$5: " + reg[5] + " "   // 4
registers per line
					+ "$6: " + reg[6] + " " + "$7: " + reg[7]);
			System.out.println("$8: " + reg[8] + " " + "$9: " + reg[9] + " "
					+ "$10:" + reg[10] + " " + "$11:" + reg[11]);
			System.out.println("$12:" + reg[12] + " " + "$13:" + reg[13] + " "
					+ "$14:" + reg[14] + " " + "$15:" + reg[15]);
			System.out.println("$16:" + reg[16] + " " + "$17:" + reg[17] + " "
					+ "$18:" + reg[18] + " " + "$19:" + reg[19]);
			System.out.println("$20:" + reg[20] + " " + "$21:" + reg[21] + " "
					+ "$22:" + reg[22] + " " + "$23:" + reg[23]);
			System.out.println("$24:" + reg[24] + " " + "$25:" + reg[25] + " "
					+ "$26:" + reg[26] + " " + "$27:" + reg[27]);
			System.out.println("$28:" + reg[28] + " " + "$29:" + reg[29] + " "
					+ "$30:" + reg[30] + " " + "$31:" + reg[31]);
			System.out.println("========================================"); // 40
='s

		}

		// Main method, get input from file
		public static void main(String[] args) {

			File input;
			Scanner inputScanner;
			int mode;

			try {
				inputScanner = new Scanner(new File ("code_1.txt"));
				mode = 0;
				while (inputScanner.hasNext()) {
					memory.add(inputScanner.nextInt(16));
```

```
                    }

                    Iterator<Integer> iterator = memory.iterator();
                    if (iterator.hasNext())
                          mode = iterator.next();    // first line of file is mode
(0 or 1)
                    while (iterator.hasNext()) {
                          Convert(iterator.next());
                          if ((mode == 1) && (iterator.hasNext()))  // if mode = 1,
then print registers for every step/input
                                RegistertoString();  // print registers
                    }
                    RegistertoString();

              } catch (Exception e) {
                    e.printStackTrace();
              }
        }
}
```