Take the given jar file and the two MIPS program sections you want to input.

```
PS C:\Users\zklod\UWM\Fall 2024\Computer Architecture\Program2\Prog2Milestone2\src> java -jar Prog2Milestone2.jar EvenOr
Odd.text EvenOrOdd.data
Enter your integer:
```

This MIPS program requests you to enter a number, and it will determine if the number is even or odd.

```
Your integer is EVEN!
```

```
Your integer is ODD!
```

Below is the source code that will read the given files and break down the logic into register and memory values and perform mnemonic keywords from there.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    static int pc = 0x00400000; // Program Counter starts here
    static boolean TERMINATE = false;
    static HashMap<Integer, String> textSegment = new HashMap<>();
    static HashMap<Integer, String> dataSegment = new HashMap<>();
    private static final Map<String, Integer> registers = new HashMap<>();

    // Map for register numbers (e.g., 2 -> $v0, 4 -> $a0)
    private static final Map<Integer, String> registerNumbers = new HashMap<>();

    static {
        // Initialize registers with default values
        registers.put("$zero", 0);
        registers.put("$at", 0);
        registers.put("$v0", 0);
        registers.put("$v1", 0);
        registers.put("$a0", 0);
        registers.put("$a1", 0);
        registers.put("$a2", 0);
        registers.put("$a3", 0);
        registers.put("$t0", 0);
        registers.put("$t1", 0);
        registers.put("$t2", 0);
        registers.put("$t3", 0);
        registers.put("$t4", 0);
        registers.put("$t5", 0);
        registers.put("$t6", 0);
        registers.put("$t7", 0);
        registers.put("$s0", 0);
        registers.put("$s1", 0);
        registers.put("$s2", 0);
        registers.put("$s3", 0);
        registers.put("$s4", 0);
        registers.put("$s5", 0);
        registers.put("$s6", 0);
        registers.put("$s7", 0);
        registers.put("$t8", 0);
        registers.put("$t9", 0);
        registers.put("$k0", 0);
        registers.put("$k1", 0);
        registers.put("$gp", 0);
        registers.put("$sp", 0);
```

```
        registers.put("$fp", 0);
        registers.put("$ra", 0);

        // Initialize the register-number-to-register-name mapping
        registerNumbers.put(0x0, "$zero");
        registerNumbers.put(0x1, "$at");
        registerNumbers.put(0x2, "$v0");
        registerNumbers.put(0x3, "$v1");
        registerNumbers.put(0x4, "$a0");
        registerNumbers.put(0x5, "$a1");
        registerNumbers.put(0x6, "$a2");
        registerNumbers.put(0x7, "$a3");
        registerNumbers.put(0x8, "$t0");
        registerNumbers.put(0x9, "$t1");
        registerNumbers.put(0xA, "$t2");
        registerNumbers.put(0xB, "$t3");
        registerNumbers.put(0xC, "$t4");
        registerNumbers.put(0xD, "$t5");
        registerNumbers.put(0xE, "$t6");
        registerNumbers.put(0xF, "$t7");
        registerNumbers.put(0x10, "$s0");
        registerNumbers.put(0x11, "$s1");
        registerNumbers.put(0x12, "$s2");
        registerNumbers.put(0x13, "$s3");
        registerNumbers.put(0x14, "$s4");
        registerNumbers.put(0x15, "$s5");
        registerNumbers.put(0x16, "$s6");
        registerNumbers.put(0x17, "$s7");
        registerNumbers.put(0x18, "$t8");
        registerNumbers.put(0x19, "$t9");
        registerNumbers.put(0x1A, "$k0");
        registerNumbers.put(0x1B, "$k1");
        registerNumbers.put(0x1C, "$gp");
        registerNumbers.put(0x1D, "$sp");
        registerNumbers.put(0x1E, "$fp");
        registerNumbers.put(0x1F, "$ra");
    }
    // R-type instructions (use funct codes)
    private static final Map<String, String> rTypeInstructions = new HashMap<>();
    static {
        rTypeInstructions.put("100000", "add");
        rTypeInstructions.put("100100", "and");
        rTypeInstructions.put("100101", "or");
        rTypeInstructions.put("101010", "slt");
        rTypeInstructions.put("100010", "sub");
        rTypeInstructions.put("001100", "syscall");
    }

    // I-type instructions (use opcodes)
```

```
    private static final Map<String, String> iTypeInstructions = new HashMap<>();
    static {
        iTypeInstructions.put("001001", "addiu");
        iTypeInstructions.put("001100", "andi");
        iTypeInstructions.put("000100", "beq");
        iTypeInstructions.put("000101", "bne");
        iTypeInstructions.put("001111", "lui");
        iTypeInstructions.put("100011", "lw");
        iTypeInstructions.put("101011", "sw");
        iTypeInstructions.put("001101", "ori");
    }

    // J-type instructions (use opcodes)
    private static final Map<String, String> jTypeInstructions = new HashMap<>();
    static {
        jTypeInstructions.put("000010", "j");
    }









    public static void main(String[] args) {
        String textFilePath = args[0];
        String dataFilePath = args[1];



        int textStartAddress = 0x00400000; // Starting address for instructions
        int dataStartAddress = 0x10010000;
        try {

            // Read the file and populate the textSegment
            BufferedReader reader = new BufferedReader(new FileReader(textFilePath));
            String line;
            int address = textStartAddress;

            while ((line = reader.readLine()) != null  && !line.equals("00000000")) {
                textSegment.put(address, line.trim());
                address += 4; // Increment address by 4 bytes for each instruction
            }



            reader.close();
        } catch (IOException e) {
            System.err.println("Error reading the file: " + e.getMessage());
        }
```

```
        try {
            // Read the file and populate the textSegment
            BufferedReader reader = new BufferedReader(new FileReader(dataFilePath));
            String line;
            int address = dataStartAddress;
            while ((line = reader.readLine()) != null) {
                String trimmed = line.trim();
                String string = "";
                for (int i = 0; i<8; i+=2){
                    string = trimmed.substring(i, i+2);
                    dataSegment.put(address, string);
                    address += 1;
                }

            }



            reader.close();
        } catch (IOException e) {
            System.err.println("Error reading the file: " + e.getMessage());
        }



        // Simulate instruction execution



        while (textSegment.containsKey(pc) && !TERMINATE) {
            String instruction = textSegment.get(pc);
            if(instruction.equals("00000000")){
                System.out.println("-- Program has finished running (dropped off bottom)
--");
                break;
            }
            String hex = args[0];
            long decimal = Long.parseUnsignedLong(instruction, 16);
            String bin = String.format("%32s", Long.toBinaryString(decimal)).replace(' ',
'0');
            String opcode = bin.substring(0, 6);
            String mnemonic;
            if (opcode.equals("000000")) {
                String funct = bin.substring(26);
                mnemonic = (String)rTypeInstructions.getOrDefault(funct, "unknown");
                rType(mnemonic, bin, funct);

            } else if (jTypeInstructions.containsKey(opcode)) {
                mnemonic = (String)jTypeInstructions.getOrDefault(opcode, "unknown");
                jType(mnemonic, bin, opcode);
```

```
            } else {
                mnemonic = (String)iTypeInstructions.getOrDefault(opcode, "unknown");
                iType(mnemonic, bin, opcode);
            }
            pc += 4; // Move to the next instruction
        }
        if(TERMINATE){
            System.out.println("-- Program has finished running --");
        }

    }

    public static void rType(String mnemonic, String bin, String funct) {
        String opcode = "00";
        String rs = Integer.toHexString(Integer.parseInt(bin.substring(6, 11), 2));
        String rt = Integer.toHexString(Integer.parseInt(bin.substring(11, 16), 2));
        String rd = Integer.toHexString(Integer.parseInt(bin.substring(16, 21), 2));
        String shmt = Integer.toHexString(Integer.parseInt(bin.substring(21, 25), 2));
        funct = Integer.toHexString(Integer.parseInt(funct, 2));
        rs = String.format("%2s", rs).replace(' ', '0');
        rt = String.format("%2s", rt).replace(' ', '0');
        rd = String.format("%2s", rd).replace(' ', '0');
        shmt = String.format("%2s", shmt).replace(' ', '0');
        funct = String.format("%2s", funct).replace(' ', '0');

        switch (mnemonic) {
            case "add": {
                // Get the register values by calling the helper method
                int[] values = getRegisterValues(rs, rt, rd);
                int rsValue = values[0];
                int rtValue = values[1];

                // Perform the addition
                int result = rsValue + rtValue;

                // Store the result in the destination register
                registers.put(registerNumbers.get(Integer.parseInt(rd, 16)), result);
                break;
            }
            case "and": {
                // Get the register values by calling the helper method
                int[] values = getRegisterValues(rs, rt, rd);
                int rsValue = values[0];
                int rtValue = values[1];


                int result = rsValue & rtValue;
```

```
            registers.put(registerNumbers.get(Integer.parseInt(rd, 16)), result);
            break;
        }
        case "or": {
            // Get the register values by calling the helper method
            int[] values = getRegisterValues(rs, rt, rd);
            int rsValue = values[0];
            int rtValue = values[1];


            int result = rsValue | rtValue;


            registers.put(registerNumbers.get(Integer.parseInt(rd, 16)), result);
            break;
        }
        case "slt": {
            int[] values = getRegisterValues(rs, rt, rd);
            int rsValue = values[0];
            int rtValue = values[1];

            // Perform the Set Less Than (slt) operation
            int result = (rsValue < rtValue) ? 1 : 0;

            // Store the result in the destination register
            registers.put(registerNumbers.get(Integer.parseInt(rd, 16)), result);
            break;
        }
        case "sub": {
            int[] values = getRegisterValues(rs, rt, rd);
            int rsValue = values[0];
            int rtValue = values[1];


            int result = rsValue - rtValue;


            registers.put(registerNumbers.get(Integer.parseInt(rd, 16)), result);
            break;
        }
        case "syscall": {
            int regValue = registers.get("$v0");
            if (regValue == 4) {
                int address = registers.get("$a0") + 3;
```

```
                int builder = address;
                String word = dataSegment.get(address);

                String output = "";
                while (!word.equals("00")) {
                    String string = "";
                    for (int i = 0; i < 4; i++) {
                        string += word;
                        builder -= 1;
                        word = dataSegment.get(builder);
                    }
                    address += 4;
                    builder = address;
                    output += string;
                    word = dataSegment.get(address);

                }
                StringBuilder text = new StringBuilder("");
                for (int i = 0; i < output.length(); i += 2) {
                    String str = output.substring(i, i + 2);
                    text.append((char) Integer.parseInt(str, 16));
                }
                System.out.println(text);
            } else if (regValue == 1) {
                System.out.println(registers.get("$a0"));
            } else if (regValue == 5) {
                Scanner scanner = new Scanner(System.in);
                registers.put(registerNumbers.get(2), scanner.nextInt());
            } else if (regValue == 10) {
                TERMINATE = true;
            }
            break;
        }
    }


}

public static void iType (String mnemonic, String bin, String opcode){
    opcode = Integer.toHexString(Integer.parseInt(opcode, 2));
    String rs = Integer.toHexString(Integer.parseInt(bin.substring(6, 11), 2));
    String rt = Integer.toHexString(Integer.parseInt(bin.substring(11, 16), 2));
    String imm = Integer.toHexString(Integer.parseInt(bin.substring(16), 2));
    opcode = String.format("%2s", opcode).replace(' ', '0');
    rs = String.format("%2s", rs).replace(' ', '0');
    rt = String.format("%2s", rt).replace(' ', '0');
    imm = String.format("%4s", imm).replace(' ', '0');
```

```
        switch(mnemonic){
            case "addiu" : {
                int[] values = getRegisterValues(rs, rt, null);
                int rsValue = values[0]; // Source register value



                int immediateValue = Integer.parseInt(imm, 16);



                int result = rsValue + immediateValue;



                registers.put(registerNumbers.get(Integer.parseInt(rt, 16)), result);
                break;
            }
            case "andi" : {
                int[] values = getRegisterValues(rs, rt, null);
                int rsValue = values[0]; // Source register value



                int immediateValue = Integer.parseInt(imm, 16);



                int result = rsValue & immediateValue;



                registers.put(registerNumbers.get(Integer.parseInt(rt, 16)), result);
                break;
            }
            case "beq" : {
                // Get the register values for rs and rt
                int[] values = getRegisterValues(rs, rt, null);
                int rsValue = values[0]; // Value in source register rs
               int rtValue = values[1]; // Value in source register rt

                 if (rsValue == rtValue) {
                     int offset = Integer.parseInt(imm, 16); // Parse the immediate
(offset)

                     offset = offset << 2; // Multiply offset by 4 (left shift by 2)

                    // Update the program counter (PC)
```

```
                pc = pc + offset;
                break;
            }
        }
        case "bne" : {

            int[] values = getRegisterValues(rs, rt, null);
            int rsValue = values[0];
            int rtValue = values[1];

            if (rsValue != rtValue) {
                int offset = Integer.parseInt(imm, 16);

                offset = offset << 2;



                pc = pc + offset;
                break;
            }
        }
        case "lui" : {



            int immediateValue = Integer.parseInt(imm, 16);



            int upperValue = immediateValue << 16;



            registers.put(registerNumbers.get(Integer.parseInt(rt, 16)), upperValue);
            break;
        }
        case "lw" : {
            int baseAddress = registers.get(rs);
            int memoryAddress = baseAddress + Integer.parseInt(imm);



            if (dataSegment.containsKey(memoryAddress)) {
                int value = Integer.parseInt(dataSegment.get(memoryAddress));
                registers.put(rt, value);
             }
            break;
        }
        case "sw" : {
            int baseAddress = registers.get(rs);
            int memoryAddress = baseAddress + Integer.parseInt(imm);
```

```
            // Retrieve the value to store
            if (registers.containsKey(rt)) {
                int value = registers.get(rt);
                dataSegment.put(memoryAddress, String.valueOf(value));
             }
            break;
        }
        case "ori" : {
            int[] values = getRegisterValues(rs, rt, null);
            int rsValue = values[0]; // Source register value



            int immediateValue = Integer.parseInt(imm, 16);



            int result = rsValue | immediateValue;



            registers.put(registerNumbers.get(Integer.parseInt(rt, 16)), result);
            break;
        }
    }
}

public static void jType (String mnemonic, String bin, String opcode){
    opcode = Integer.toHexString(Integer.parseInt(opcode, 2));
    String shift = bin.substring(6) + "00";
    String index = Integer.toHexString(Integer.parseInt(shift, 2));
    opcode = String.format("%2s", opcode).replace(' ', '0');
    index = String.format("%7s", index).replace(' ', '0');



    pc = Integer.parseInt(index, 16) - 4;
}

private static int[] getRegisterValues(String rs, String rt, String rd) {
    // Convert the hexadecimal strings to register numbers
    int rsRegisterNumber = Integer.parseInt(rs, 16);
    int rtRegisterNumber = Integer.parseInt(rt, 16);




    // Retrieve the corresponding register names from the registerNumbers map
    String rsRegister = registerNumbers.get(rsRegisterNumber);
```

```
        String rtRegister = registerNumbers.get(rtRegisterNumber);



        // Retrieve the values of rs, rt, and rd from the registers map
        int rsValue = registers.get(rsRegister);
        int rtValue = registers.get(rtRegister);

        if(!(rd == null)){
            int rdRegisterNumber = Integer.parseInt(rd, 16);
            String rdRegister = registerNumbers.get(rdRegisterNumber);
            int rdValue = registers.get(rdRegister);
            return new int[] {rsValue, rtValue, rdValue};
        }else{
            return new int[] {rsValue, rtValue};
        }



    }
}
```