CNG331 – Computer Organization

Term Project: Assembler Design

EWO ASSEMBLER by

Celal Sahir Çetiner 1755420

Osman Ömer Yıldıztugay 1921956

Introduction

We were asked to implement an assembler using one of the high level programming languages to convert any MIPS assembly program containing some of the MIPS instructions (which can be found in Fig.2.27 (Page 129) of the textbook 5th Edition) to hexadecimal machine language or object code. Apart from MIPS instructions, our design had to assemble a pseudo-instruction called “move”. This instruction can also be found in the figures mentioned above. As mentioned in the project description, we assumed that the first line of swap code is stored at MIPS memory location 0x80001000, and sort code is stored immediately after. Our assembler design had to consist of two parts:

Interactive mode, where user enters a single line of instruction from command line and gets assembled hexadecimal output.

Batch mode , where programs reads a source file with extension .src, assembles to hexadecimal and outputs the result to an object code with file extension .obj.

Although we had numerous options for implementation, we did not idle around while we were trying to decide which language to use for this project. After a short debate, we easily agreed on Java.

Why Java?

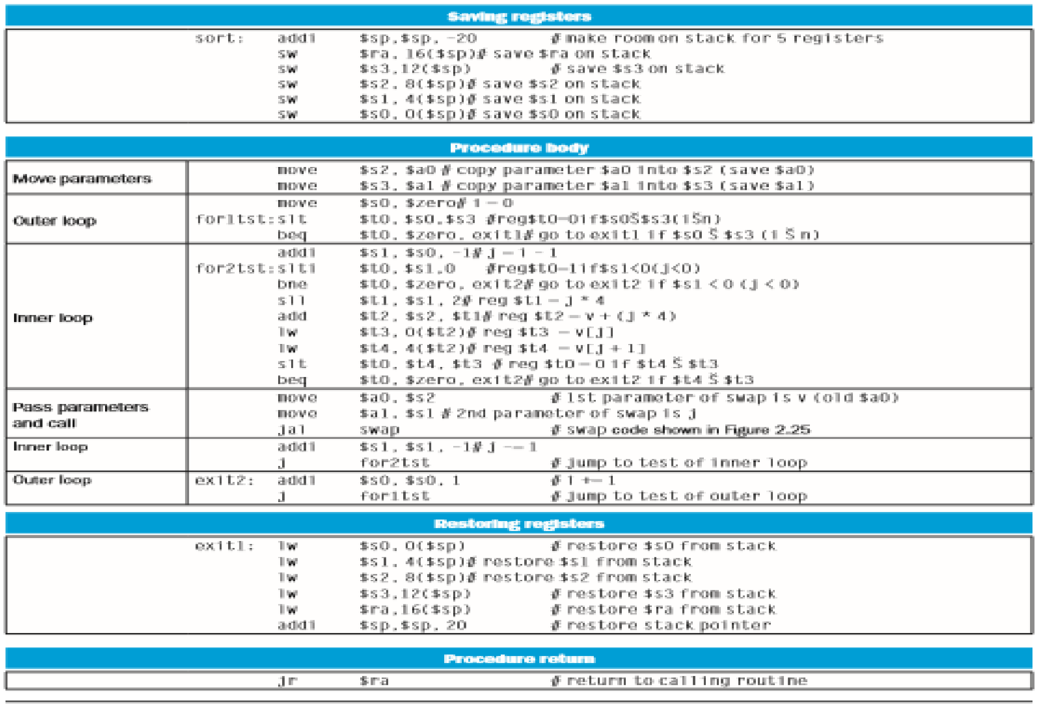
Java has always been a default choice for scientific applications. However, the main reason why we choose Java for this project was that Java provides a form of automated memory management called garbage collection. Despite the significant influence on performance caused by garbage collection, we both agreed on Java because since we had to do too many string concatenations, adjustments and parsing, other languages we practiced before could lead us to disaster. Unlike the other programming languages we practiced in previous semesters such as C, C++ and Haskell, Java is way more developer-friendly. In addition to these, we have had enough chances to practice on Java with the assignments given to us throughout this semester. We did our implementations on the environment called IntelliJ starting from scratch. There is no specific reason we can point out why we choose IntelliJ. It was already installed in both of our systems so all we needed was to start coding.

Our Compiler

We have used the IntelliJ Community addition because it is the most advanced compiler in whole industry.

Our Design

Figure below includes every single instruction that our assembler can convert to hexadecimal machine code.



Saving registers				
	sort:	addi	$sp,$sp, -20	# make room on stack for 5 registers
		sw	$ra, 16($sp)# save $ra on stack	
		sw	$s3,12($sp)	# save $s3 on stack
		sw	$s2, 8($sp)# save $s2 on stack	
		sw	$s1, 4($sp)# save $s1 on stack	
		sw	$s0, 0($sp)# save $s0 on stack	
Procedure body				
Move parameters		move	$s2, $a0 # copy parameter $a0 into $s2 (save $a0)	
		move	$s3, $a1 # copy parameter $a1 into $s3 (save $a1)	
Outer loop		move	$s0, $zero# i = 0	
	for1tst:	slt	$t0, $s0,$s3 #reg$t0=0if$s0Š$s3(iŠn)	
		beq	$t0, $zero, exit1# go to exit1 if $s0 Š $s3 (i Š n)	
Inner loop		addi	$s1, $s0, -1# j = i - 1	
	for2tst:	slti	$t0, $s1,0 #reg$t0=1if$s1<0(j<0)	
		bne	$t0, $zero, exit2# go to exit2 if $s1 < 0 (j < 0)	
		sll	$t1, $s1, 2# reg $t1 = j * 4	
		add	$t2, $s2, $t1# reg $t2 = v + (j * 4)	
		lw	$t3, 0($t2)# reg $t3 = v[j]	
		lw	$t4, 4($t2)# reg $t4 = v[j + 1]	
		slt	$t0, $t4, $t3 # reg $t0 = 0 if $t4 Š $t3	
		beq	$t0, $zero, exit2# go to exit2 if $t4 Š $t3	
Pass parameters and call		move	$a0, $s2	# 1st parameter of swap is v (old $a0)
		move	$a1, $s1 # 2nd parameter of swap is j	
		jal	swap	# swap code shown in Figure 2.25
Inner loop		addi	$s1, $s1, -1# j –= 1	
		j	for2tst	# jump to test of inner loop
Outer loop	exit2:	addi	$s0, $s0, 1	# i += 1
		j	for1tst	# jump to test of outer loop
Restoring registers				
	exit1:	lw	$s0, 0($sp)	# restore $s0 from stack
		lw	$s1, 4($sp)# restore $s1 from stack	
		lw	$s2, 8($sp)# restore $s2 from stack	
		lw	$s3,12($sp)	# restore $s3 from stack
		lw	$ra,16($sp)	# restore $ra from stack
		addi	$sp,$sp, 20	# restore stack pointer
Procedure return				
		jr	$ra	# return to calling routine

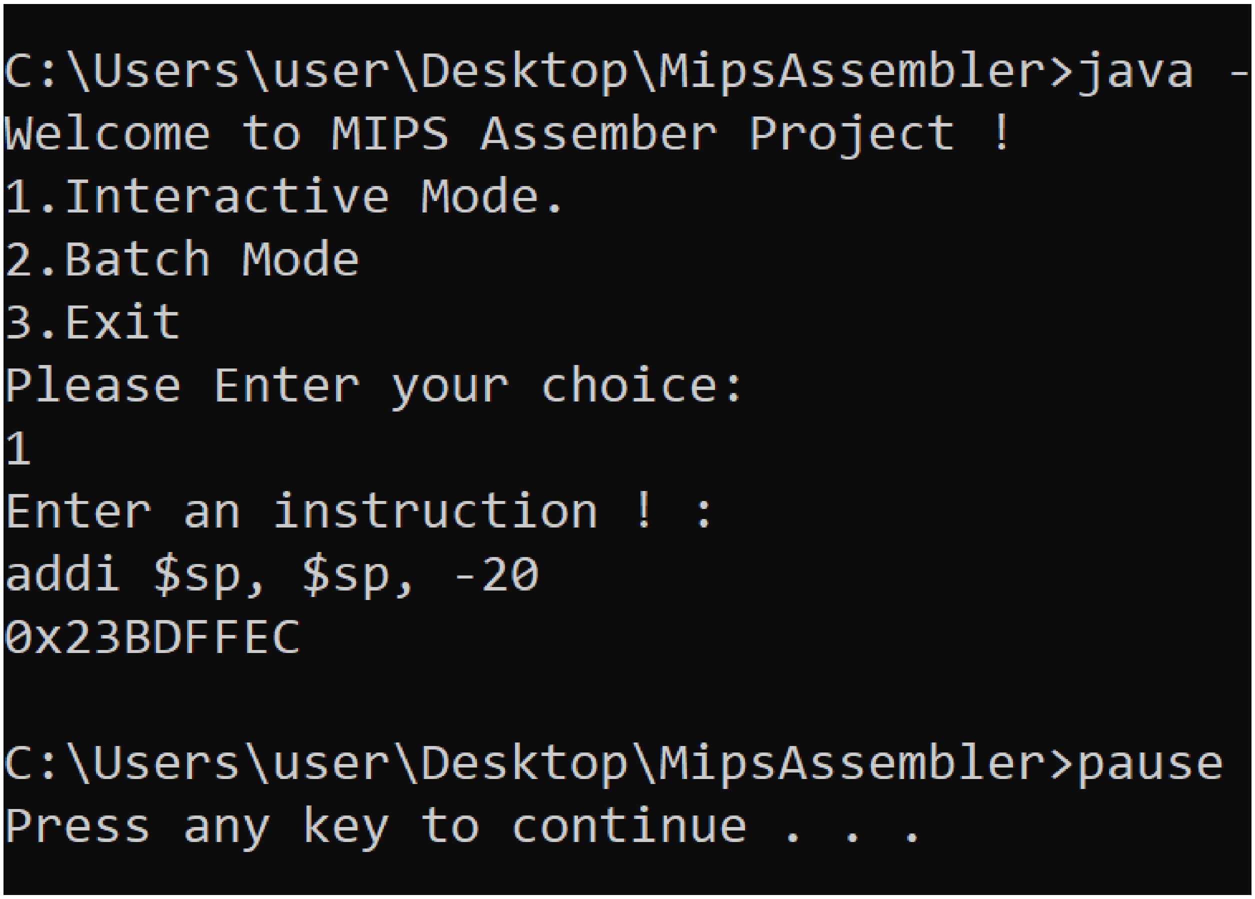
FIGURE 2.27 MIPS assembly version of procedure sort **in Figure 2.26.**

To be more descriptive, one of every instructions in the figure along with their instruction formats are listed below:

And we will show one example for each type in interactive mode;

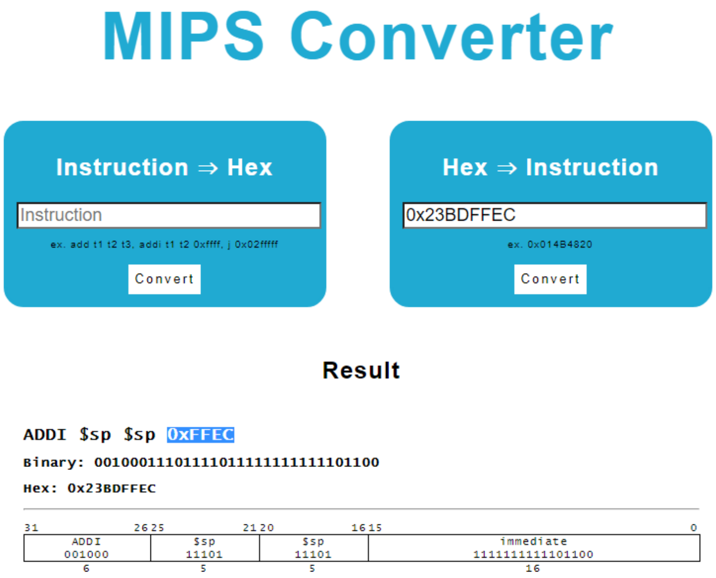
addi $sp, $sp, -20 // add -20 to the value in $sp and assign it in $sp.

Our result ; FOR (I TYPE EXAMPLE)



```
C:\Users\user\Desktop\MipsAssembler>java -
Welcome to MIPS Assember Project !
1.Interactive Mode.
2.Batch Mode
3.Exit
Please Enter your choice:
1
Enter an instruction ! :
addi $sp, $sp, -20
0x23BDFFEC

C:\Users\user\Desktop\MipsAssembler>pause
Press any key to continue . . .
```

We confirm through this website ;



MIPS Converter

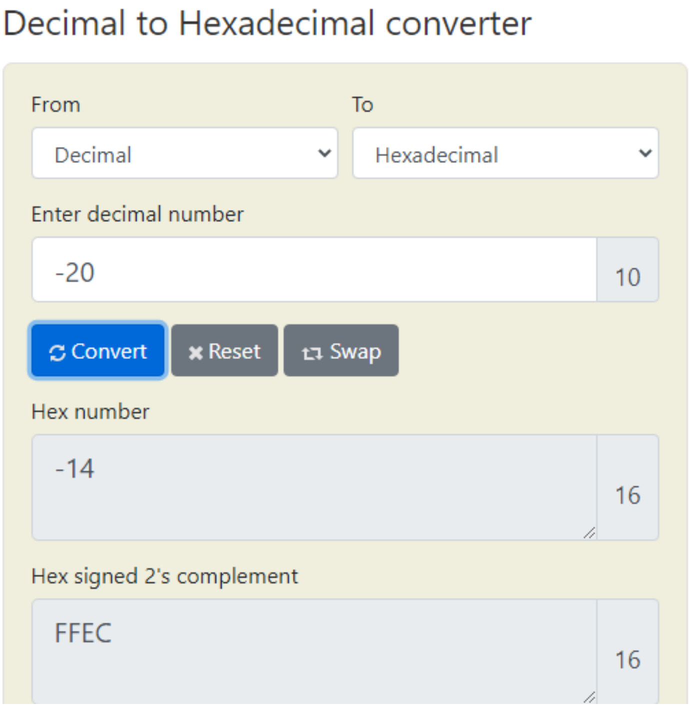
0x23BDFFEC

Result

```
ADDI $sp $sp 0xFFEC
Binary: 00100011101111011111111111101100
Hex: 0x23BDFFEC
```

31 26	25 21	20 16	15 0
ADDI 001000	$sp 11101	$sp 11101	immediate 1111111111101100
6	5	5	16

And for the immediate confirmation;



Decimal to Hexadecimal converter

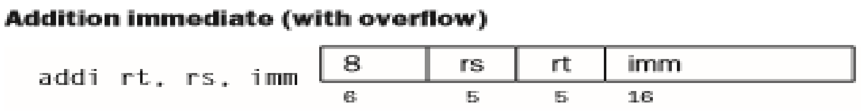
From: Decimal

To: Hexadecimal

Enter decimal number: -20 (10)

Hex number: -14 (16)

Hex signed 2's complement: FFEC (16)

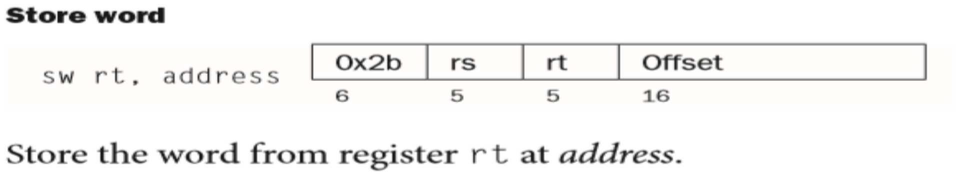


Addition immediate (with overflow)

addi rt, rs, imm

8	rs	rt	imm
6	5	5	16

sw $ra, 16($sp) // store the value in $ra to the corresponding memory.



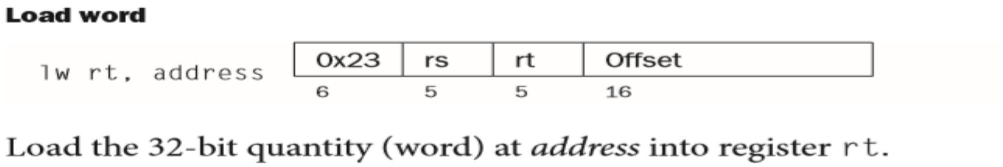
Store word

sw rt, address

0x2b	rs	rt	Offset
6	5	5	16

Store the word from register rt at *address*.

lw $s0, 0($sp) // load the value stored in the corresponding memory.



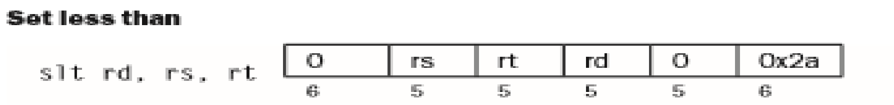
Load word

lw rt, address

0x23	rs	rt	Offset
6	5	5	16

Load the 32-bit quantity (word) at *address* into register rt.

slt $t0, $t4, $t3 // set less than



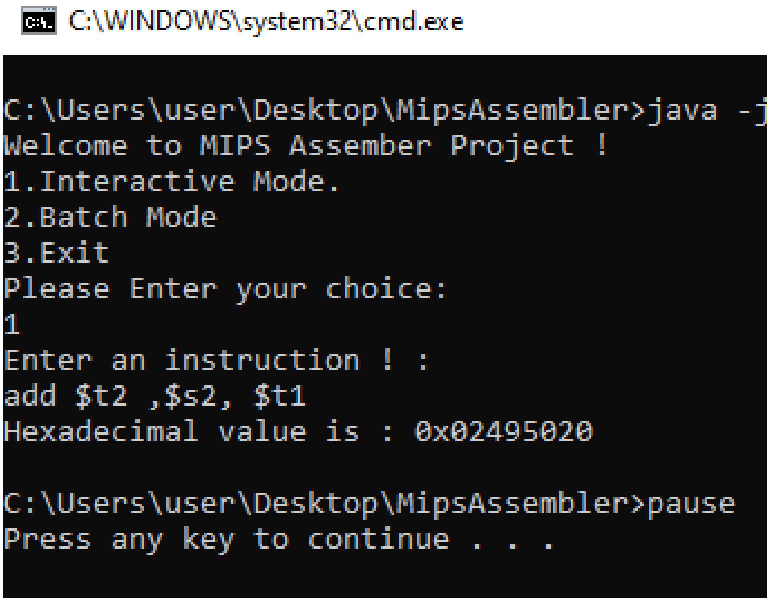
Set less than

slt rd, rs, rt

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

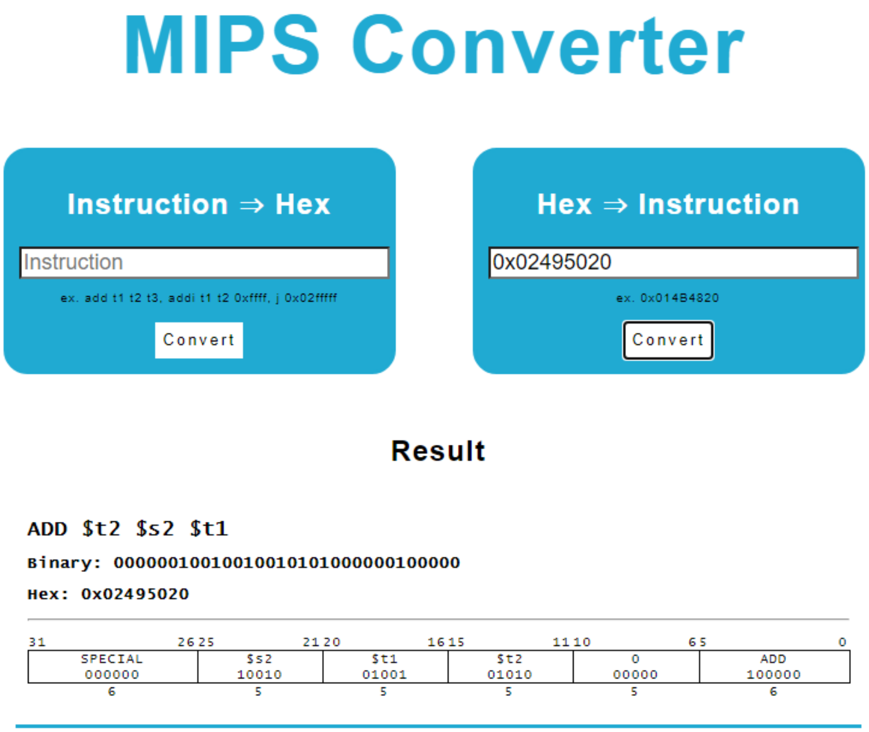
Our R TYPE EXAMPLE AS ;

add $t2 ,$s2, $t1 // sum values in $t1 and $s2, assign the result in $t2



C:\WINDOWS\system32\cmd.exe

```
C:\Users\user\Desktop\MipsAssembler>java -j
Welcome to MIPS Assember Project !
1.Interactive Mode.
2.Batch Mode
3.Exit
Please Enter your choice:
1
Enter an instruction ! :
add $t2 ,$s2, $t1
Hexadecimal value is : 0x02495020

C:\Users\user\Desktop\MipsAssembler>pause
Press any key to continue . . .
```



MIPS Converter

Instruction ⇒ Hex

Instruction

ex. add t1 t2 t3, addi t1 t2 0xffff, j 0x02ffffff

Convert

Hex ⇒ Instruction

0x02495020

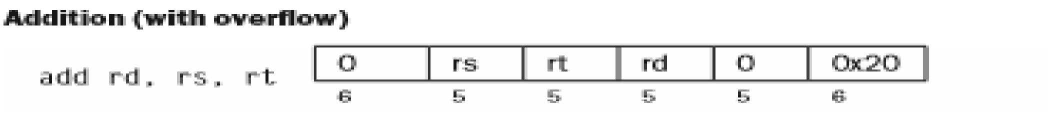
ex. 0x014B4820

Convert

Result

```
ADD $t2 $s2 $t1
Binary: 00000010010010010101000000100000
Hex: 0x02495020
```

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	$s2 10010	$t1 01001	$t2 01010	0 00000	ADD 100000
6	5	5	5	5	6

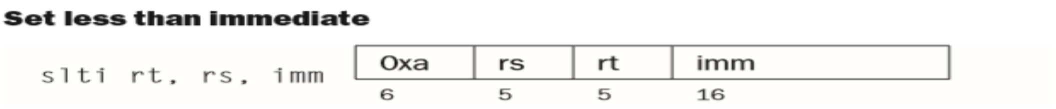


Addition (with overflow)

`add rd, rs, rt`

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

slti $t0, $s1, 0 // set less than immediate

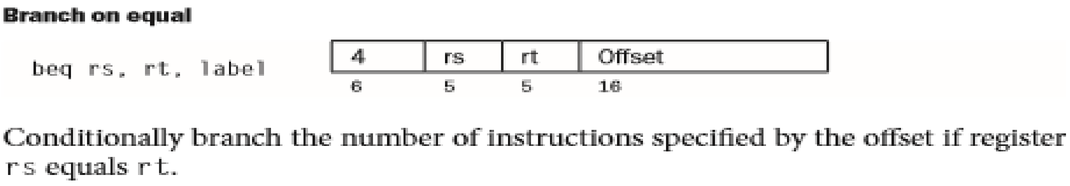


Set less than immediate

`slti rt, rs, imm`

0xa	rs	rt	imm
6	5	5	16

beq $t0, $zero, exit1 // branch equal



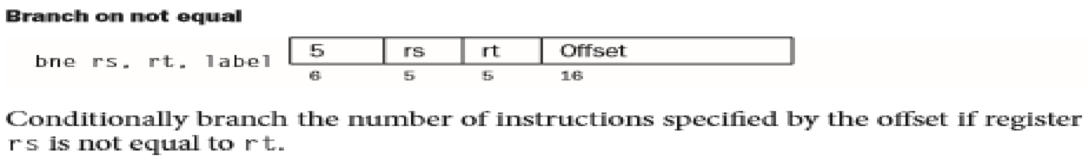
Branch on equal

`beq rs, rt, label`

4	rs	rt	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` equals `rt`.

bne $t0, $zero, $exit2 // branch not equal



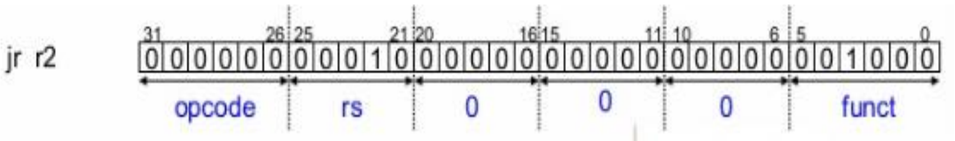
Branch on not equal

bne rs, rt, label

5	rs	rt	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register rs is not equal to rt.

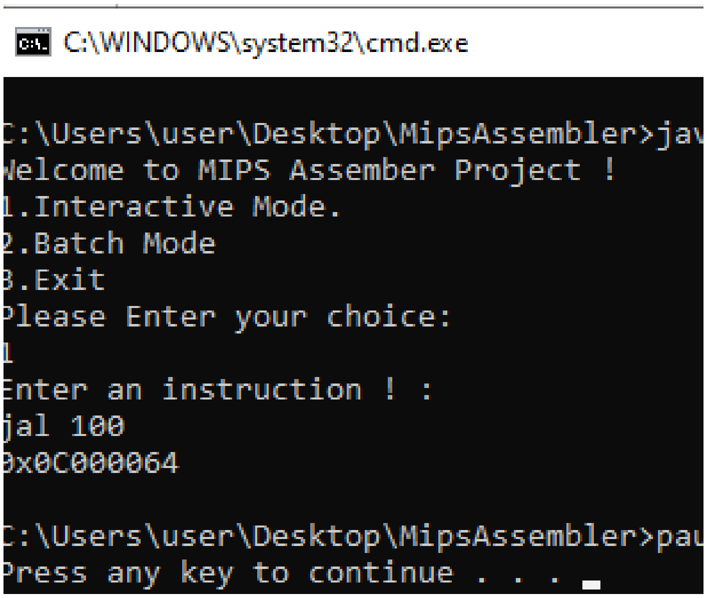
jr $ra



Ou J-TYPE EXAMPLE AS ;

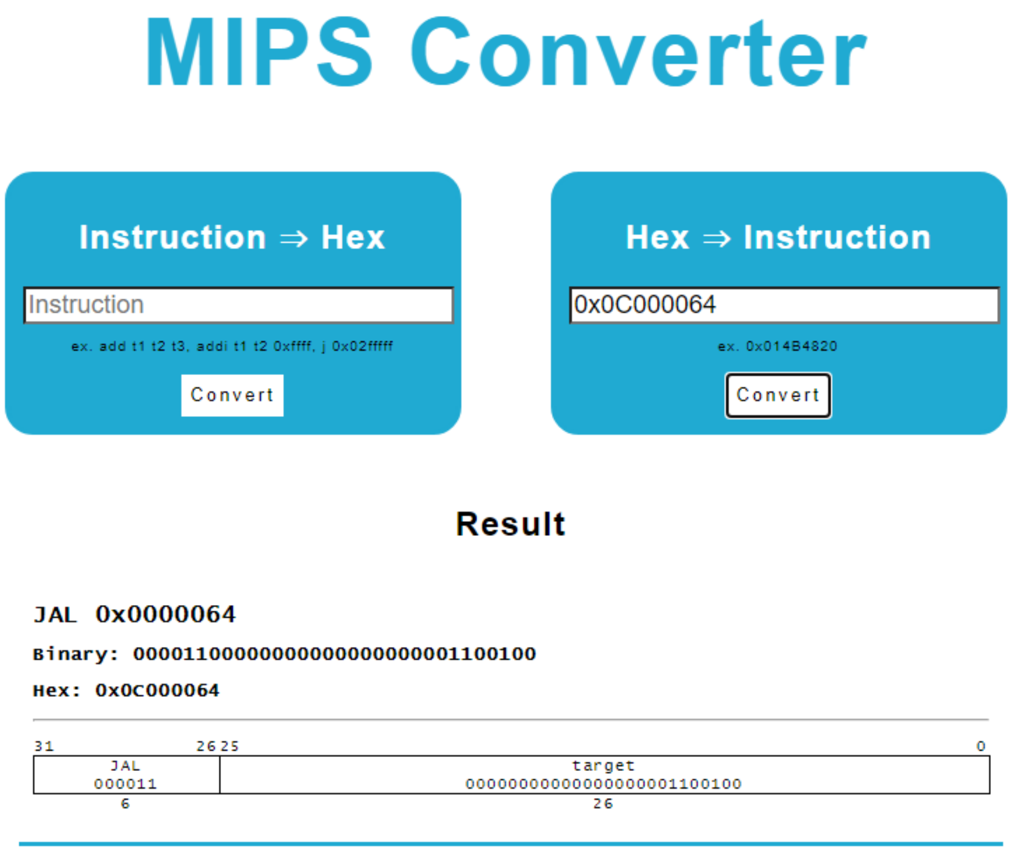
jal 100 // jump and link

in interactive mode we cant specify the Label name because we don’t know the address, we just enter an address as a value : 100



```
:\Users\user\Desktop\MipsAssembler>jav
elcome to MIPS Assember Project !
.Interactive Mode.
.Batch Mode
.Exit
lease Enter your choice:

nter an instruction ! :
al 100
x0C000064

:\Users\user\Desktop\MipsAssembler>pau
ress any key to continue . . .
```

Confirm the instruction



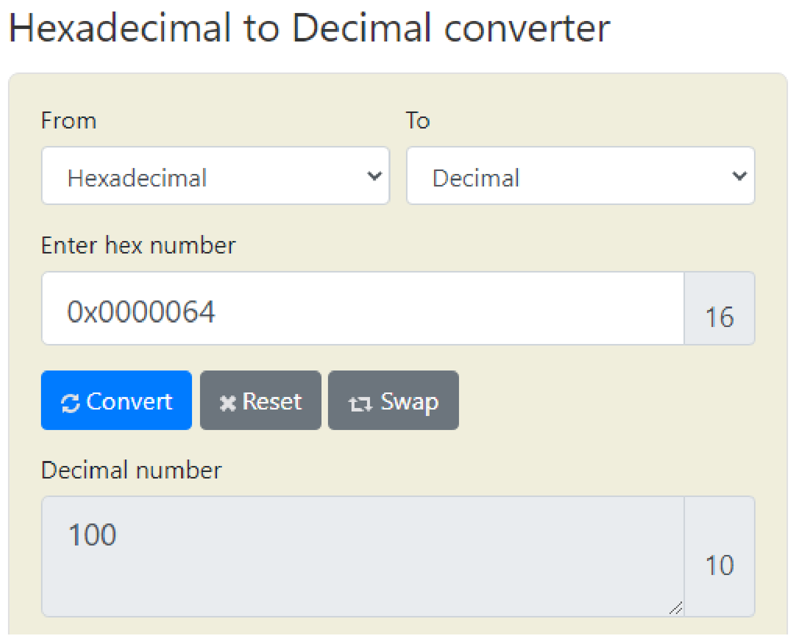
0x0C000064

Result

```
JAL 0x0000064
Binary: 00001100000000000000000001100100
Hex: 0x0C000064
```

```
31          26 25                                   0
|   JAL       |              target                  |
|  000011     |   00000000000000000001100100         |
      6                          26
```

Confirm the hexa target part ;



From: Hexadecimal

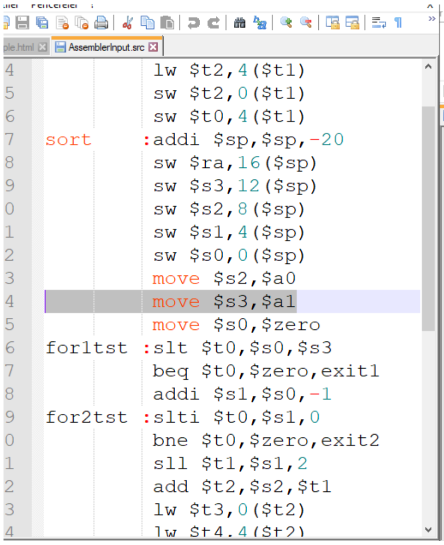
To: Decimal

Enter hex number: 0x0000064 (16)

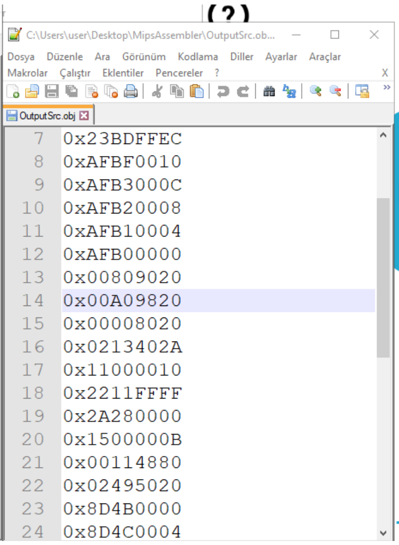
Decimal number: 100 (10)

move $s3 , $a1 // pseudo instruction which adds 0 to the value of $s2 and assign it to $a0

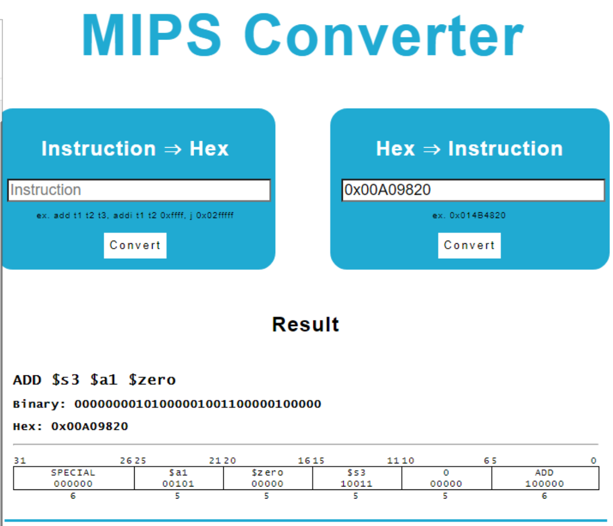
so it acts like = add   add $s3 $a1 $zero which is like moving copying



```
                lw $t2,4($t1)
                sw $t2,0($t1)
                sw $t0,4($t1)
sort           :addi $sp,$sp,-20
                sw $ra,16($sp)
                sw $s3,12($sp)
                sw $s2,8($sp)
                sw $s1,4($sp)
                sw $s0,0($sp)
                move $s2,$a0
                move $s3,$a1
                move $s0,$zero
for1tst        :slt $t0,$s0,$s3
                beq $t0,$zero,exit1
                addi $s1,$s0,-1
for2tst        :slti $t0,$s1,0
                bne $t0,$zero,exit2
                sll $t1,$s1,2
                add $t2,$s2,$t1
                lw $t3,0($t2)
                lw $t4,4($t2)
```



```
0x23BDFFEC
0xAFBF0010
0xAFB3000C
0xAFB20008
0xAFB10004
0xAFB00000
0x00809020
0x00A09820
0x00008020
0x0213402A
0x11000010
0x2211FFFF
0x2A280000
0x1500000B
0x00114880
0x02495020
0x8D4B0000
0x8D4C0004
```



0x00A09820

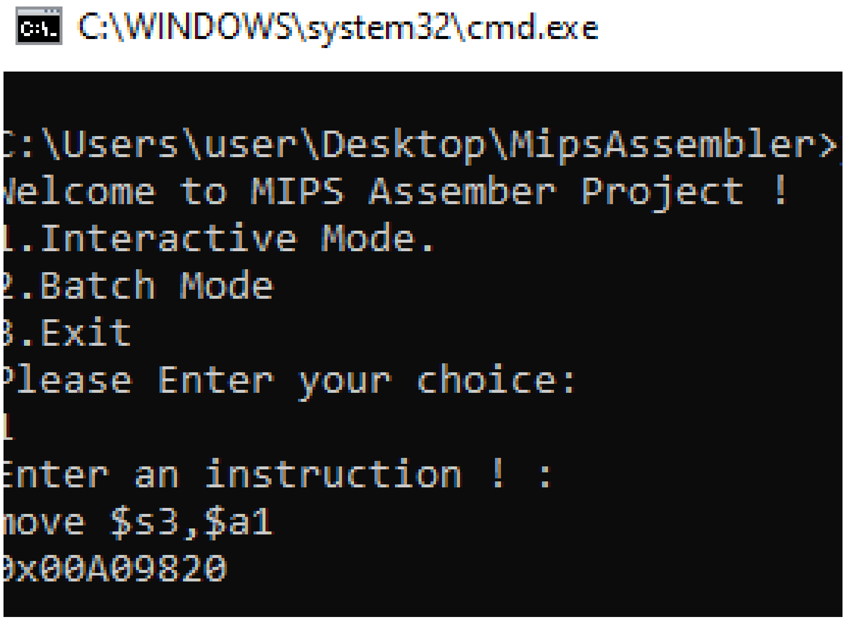
Result

ADD $s3 $a1 $zero

Binary: 00000000101000001001100000100000

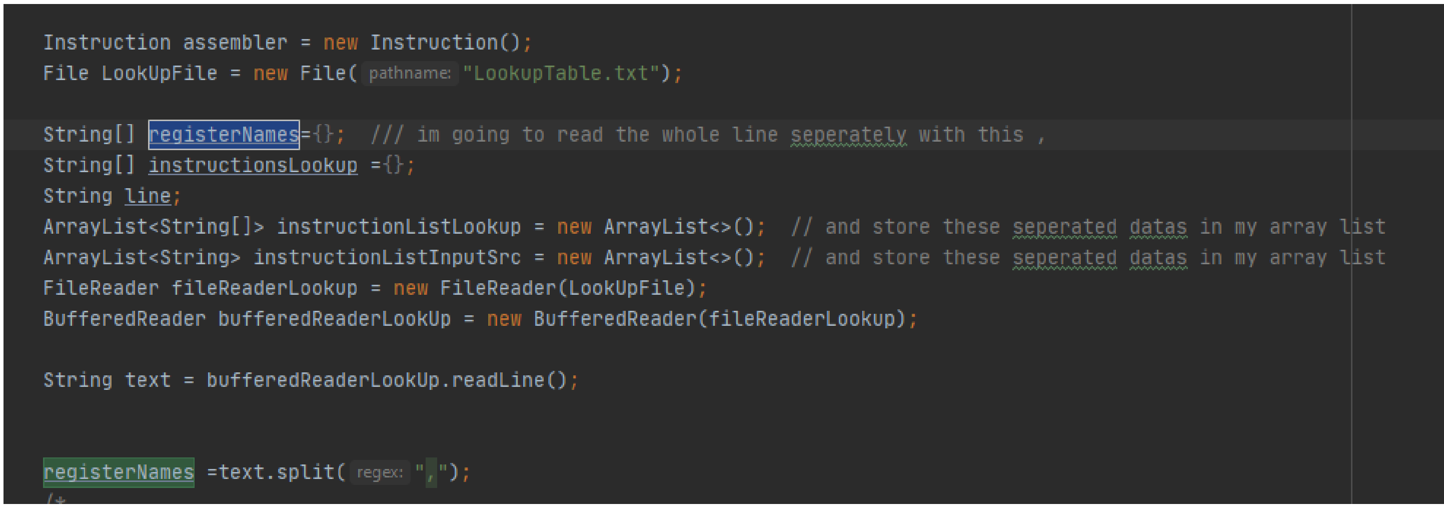
Hex: 0x00A09820

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	$a1 00101	$zero 00000	$s3 10011	0 00000	ADD 100000
6	5	5	5	5	6



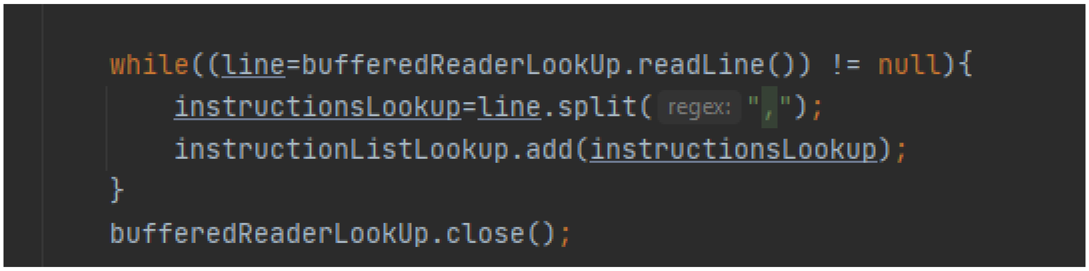
```
:\Users\user\Desktop\MipsAssembler>
elcome to MIPS Assember Project !
.Interactive Mode.
.Batch Mode
.Exit
lease Enter your choice:

nter an instruction ! :
ove $s3,$a1
x00A09820
```

Registers and all instructions listed above are saved in an external file named **lookupTable.txt.** This file has a crucial role in our design. It is kind of “How to do” manual for our program. lookupTable.txt is designed in a way that our program can read the data properly and do the required operations. Content of the lookupTable is listed in the figure below:



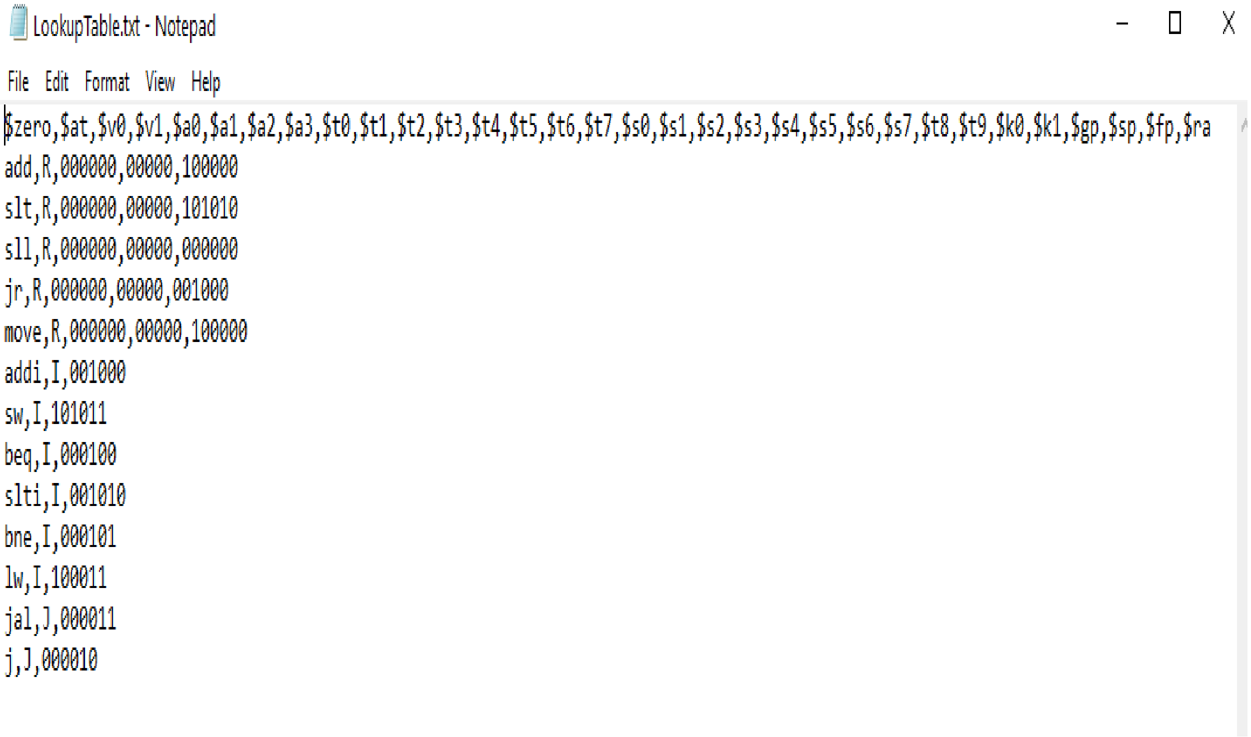
```
Instruction assembler = new Instruction();
File LookUpFile = new File( pathname: "LookupTable.txt");

String[] registerNames={};  /// im going to read the whole line seperately with this ,
String[] instructionsLookup ={};
String line;
ArrayList<String[]> instructionListLookup = new ArrayList<>();  // and store these seperated datas in my array list
ArrayList<String> instructionListInputSrc = new ArrayList<>();  // and store these seperated datas in my array list
FileReader fileReaderLookup = new FileReader(LookUpFile);
BufferedReader bufferedReaderLookUp = new BufferedReader(fileReaderLookup);

String text = bufferedReaderLookUp.readLine();


registerNames =text.split( regex: ",");
/*
```

As shown here, we are detailly splitting the register names and saved in an arraylist to use after



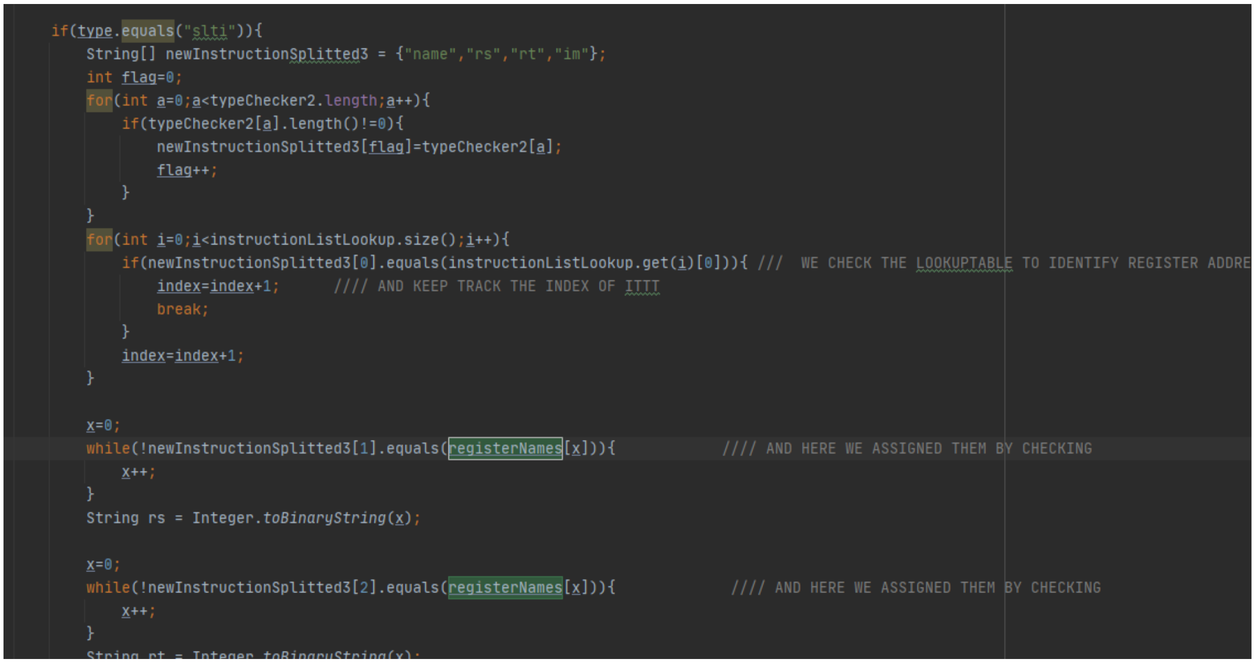
```
while((line=bufferedReaderLookUp.readLine()) != null){
    instructionsLookup=line.split( regex: ",");
    instructionListLookup.add(instructionsLookup);
}
bufferedReaderLookUp.close();
```

OUR LOOKUP TABLE TXT



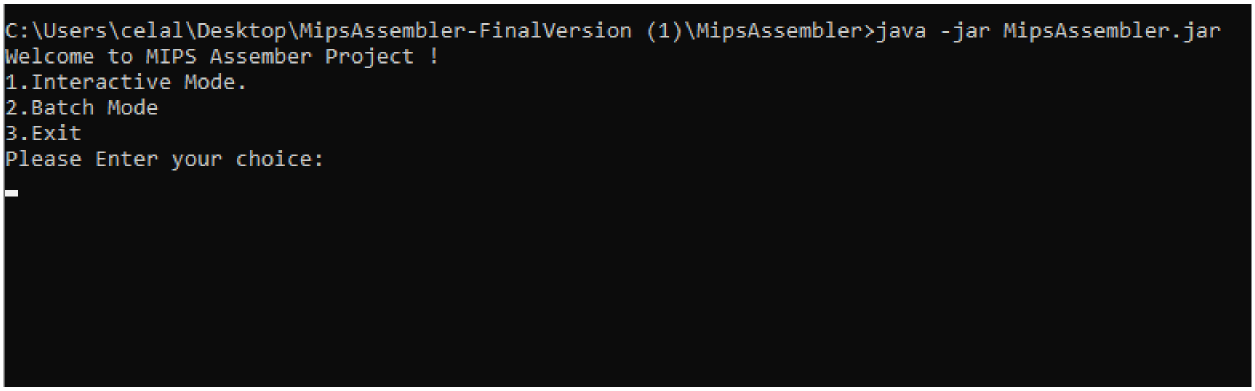
```
$zero,$at,$v0,$v1,$a0,$a1,$a2,$a3,$t0,$t1,$t2,$t3,$t4,$t5,$t6,$t7,$s0,$s1,$s2,$s3,$s4,$s5,$s6,$s7,$t8,$t9,$k0,$k1,$gp,$sp,$fp,$ra
add,R,000000,00000,100000
slt,R,000000,00000,101010
sll,R,000000,00000,000000
jr,R,000000,00000,001000
move,R,000000,00000,100000
addi,I,001000
sw,I,101011
beq,I,000100
slti,I,001010
bne,I,000101
lw,I,100011
jal,J,000011
j,J,000010
```

Based on the data retrieved from lookup table, our program can operate on both Interactive and Batch mode efficiently. For example;



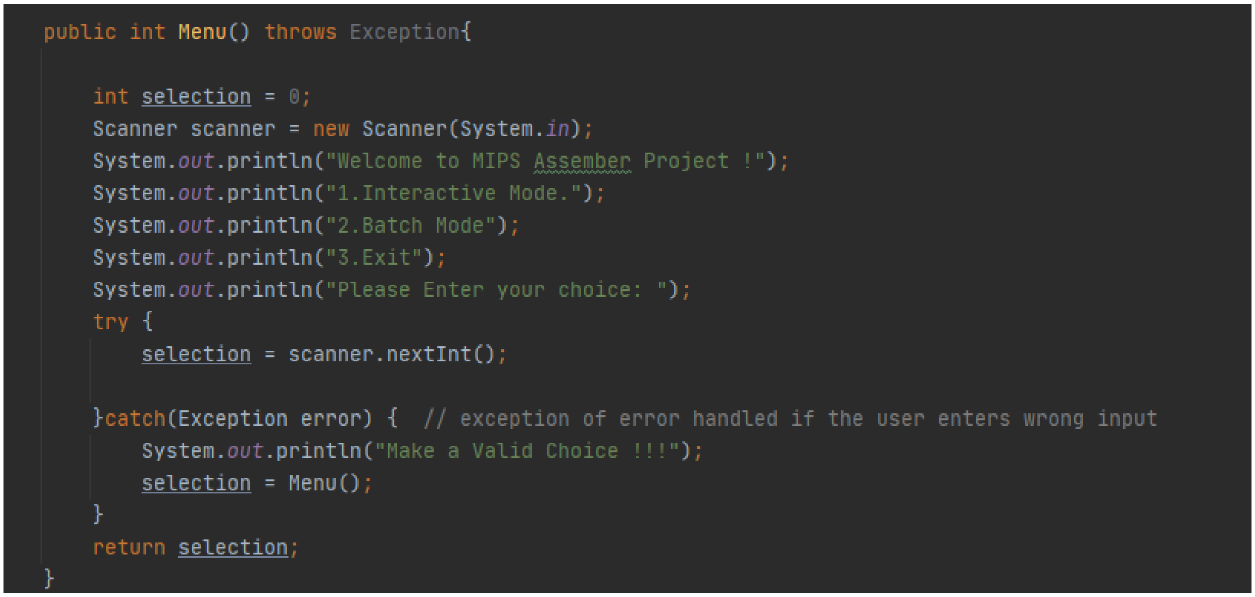
```
if(type.equals("slti")){
    String[] newInstructionSplitted3 = {"name","rs","rt","im"};
    int flag=0;
    for(int a=0;a<typeChecker2.length;a++){
        if(typeChecker2[a].length()!=0){
            newInstructionSplitted3[flag]=typeChecker2[a];
            flag++;
        }
    }
    for(int i=0;i<instructionListLookup.size();i++){
        if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){ ///  WE CHECK THE LOOKUPTABLE TO IDENTIFY REGISTER ADDRE
            index=index+1;      //// AND KEEP TRACK THE INDEX OF ITTT
            break;
        }
        index=index+1;
    }

    x=0;
    while(!newInstructionSplitted3[1].equals(registerNames[x])){            //// AND HERE WE ASSIGNED THEM BY CHECKING
        x++;
    }
    String rs = Integer.toBinaryString(x);

    x=0;
    while(!newInstructionSplitted3[2].equals(registerNames[x])){             //// AND HERE WE ASSIGNED THEM BY CHECKING
        x++;
    }
    String rt = Integer.toBinaryString(x);
```

Our assembler is designed as a simple command line application. Main menu is shown in the figure below:



```
C:\Users\celal\Desktop\MipsAssembler-FinalVersion (1)\MipsAssembler>java -jar MipsAssembler.jar
Welcome to MIPS Assember Project !
1.Interactive Mode.
2.Batch Mode
3.Exit
Please Enter your choice:
```

We set the menu as this and handled the errors



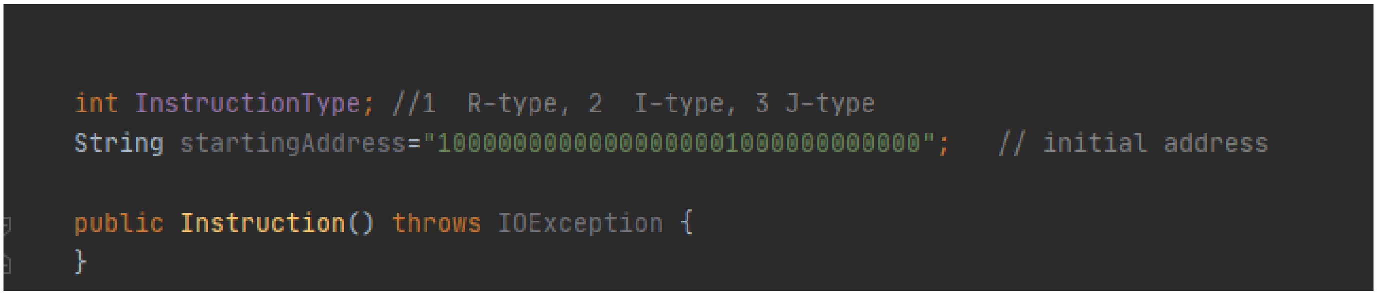
```java
public int Menu() throws Exception{

    int selection = 0;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Welcome to MIPS Assember Project !");
    System.out.println("1.Interactive Mode.");
    System.out.println("2.Batch Mode");
    System.out.println("3.Exit");
    System.out.println("Please Enter your choice: ");
    try {
        selection = scanner.nextInt();

    }catch(Exception error) {  // exception of error handled if the user enters wrong input
        System.out.println("Make a Valid Choice !!!");
        selection = Menu();
    }
    return selection;
}
```

How Interactive Mode works

YOU SHOULD ENTER THE TYPES WITH ONE SPACES AFTER THE INSTRUCTION AND PUT COMMA BETWEEN REGISTER LIKE IN REAL LIFE

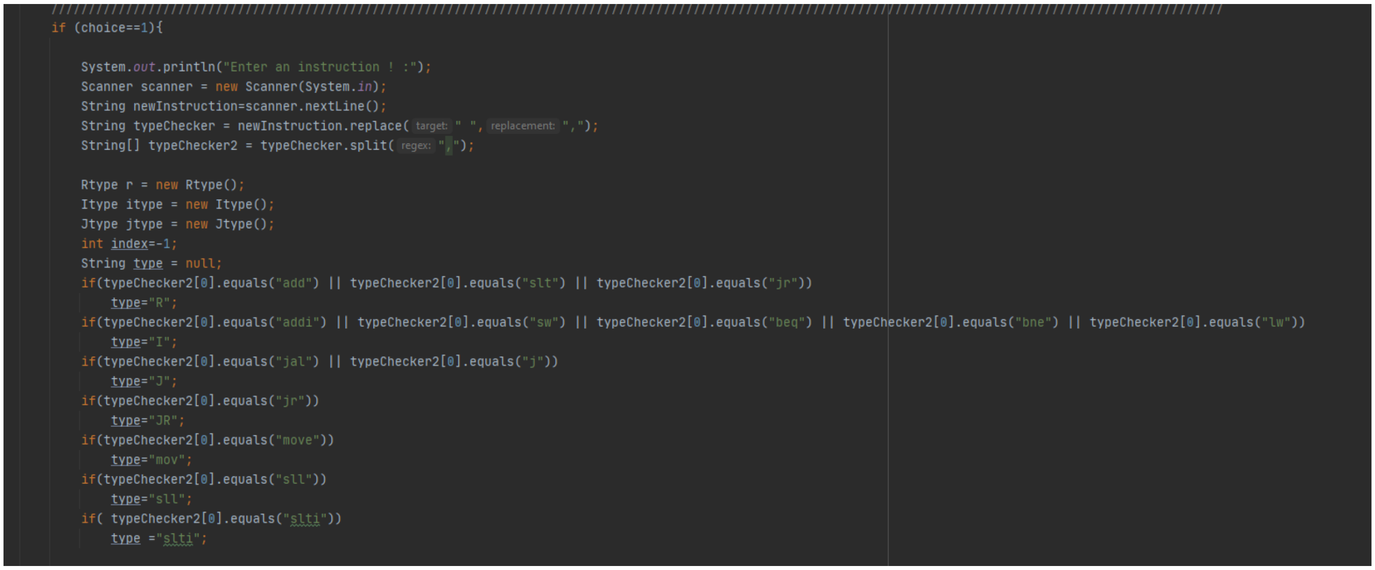
ADD $S1,$S2,$S3

We set the initial address as this ;



```java
int InstructionType; //1  R-type, 2  I-type, 3 J-type
String startingAddress="10000000000000000000010000000000000";   // initial address

public Instruction() throws IOException {
}
```

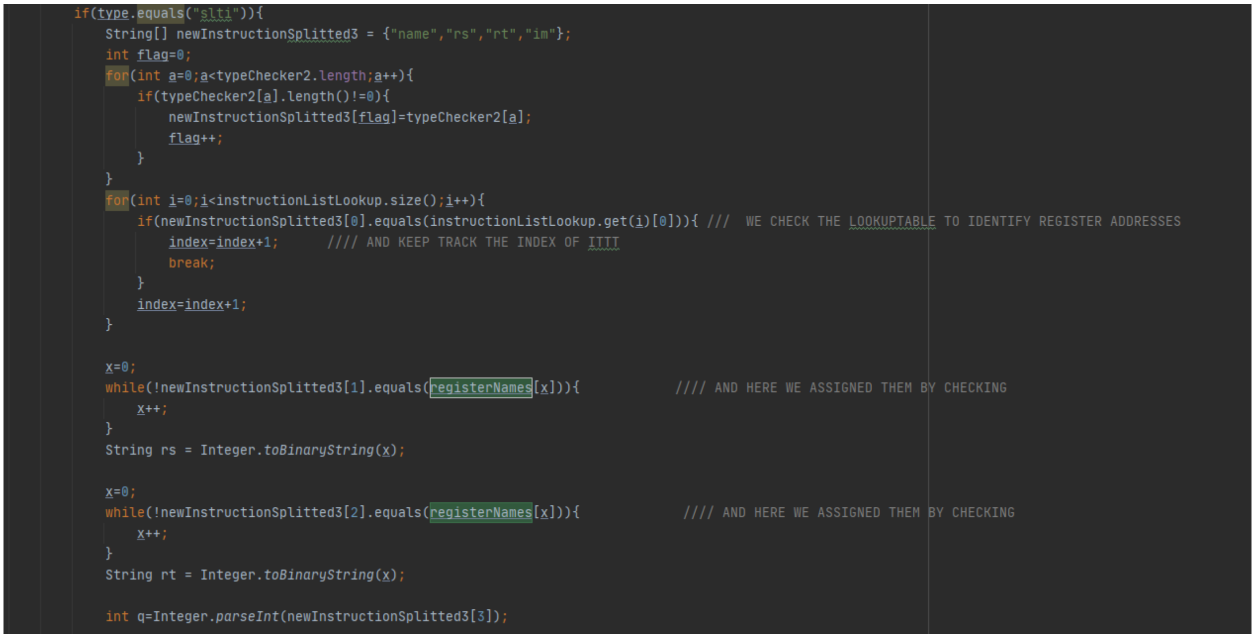
Our ıntereactive mode works through the checking types as;



```java
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
if (choice==1){

    System.out.println("Enter an instruction ! :");
    Scanner scanner = new Scanner(System.in);
    String newInstruction=scanner.nextLine();
    String typeChecker = newInstruction.replace( target: " ", replacement: ",");
    String[] typeChecker2 = typeChecker.split( regex: ",");

    Rtype r = new Rtype();
    Itype itype = new Itype();
    Jtype jtype = new Jtype();
    int index=-1;
    String type = null;
    if(typeChecker2[0].equals("add") || typeChecker2[0].equals("slt") || typeChecker2[0].equals("jr"))
        type="R";
    if(typeChecker2[0].equals("addi") || typeChecker2[0].equals("sw") || typeChecker2[0].equals("beq") || typeChecker2[0].equals("bne") || typeChecker2[0].equals("lw"))
        type="I";
    if(typeChecker2[0].equals("jal") || typeChecker2[0].equals("j"))
        type="J";
    if(typeChecker2[0].equals("jr"))
        type="JR";
    if(typeChecker2[0].equals("move"))
        type="mov";
    if(typeChecker2[0].equals("sll"))
        type="sll";
    if( typeChecker2[0].equals("slti"))
        type ="slti";
```

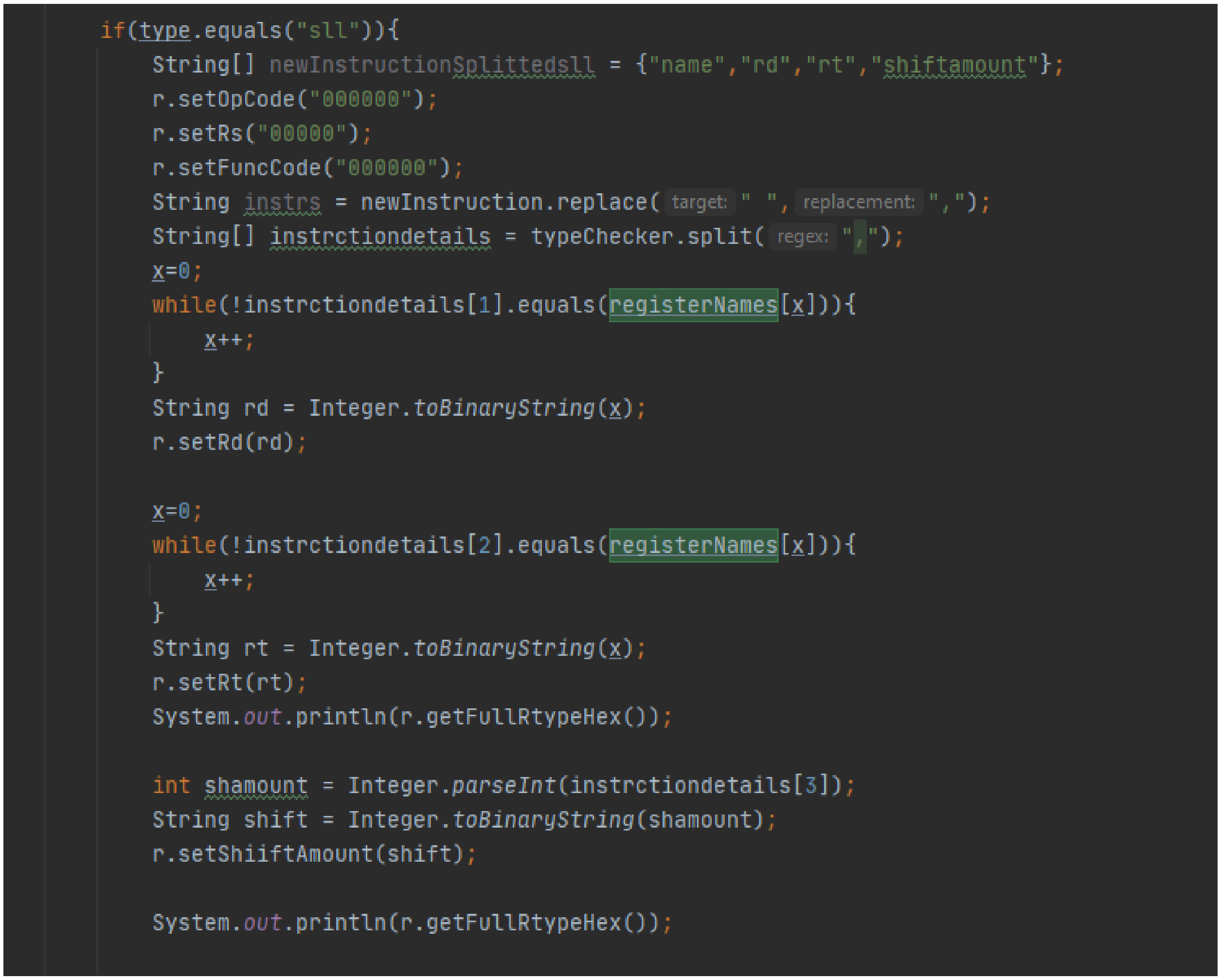
For the special working types of some R,I,J , we set the their types specially besides;

For example ; slti not behaves like general I types.



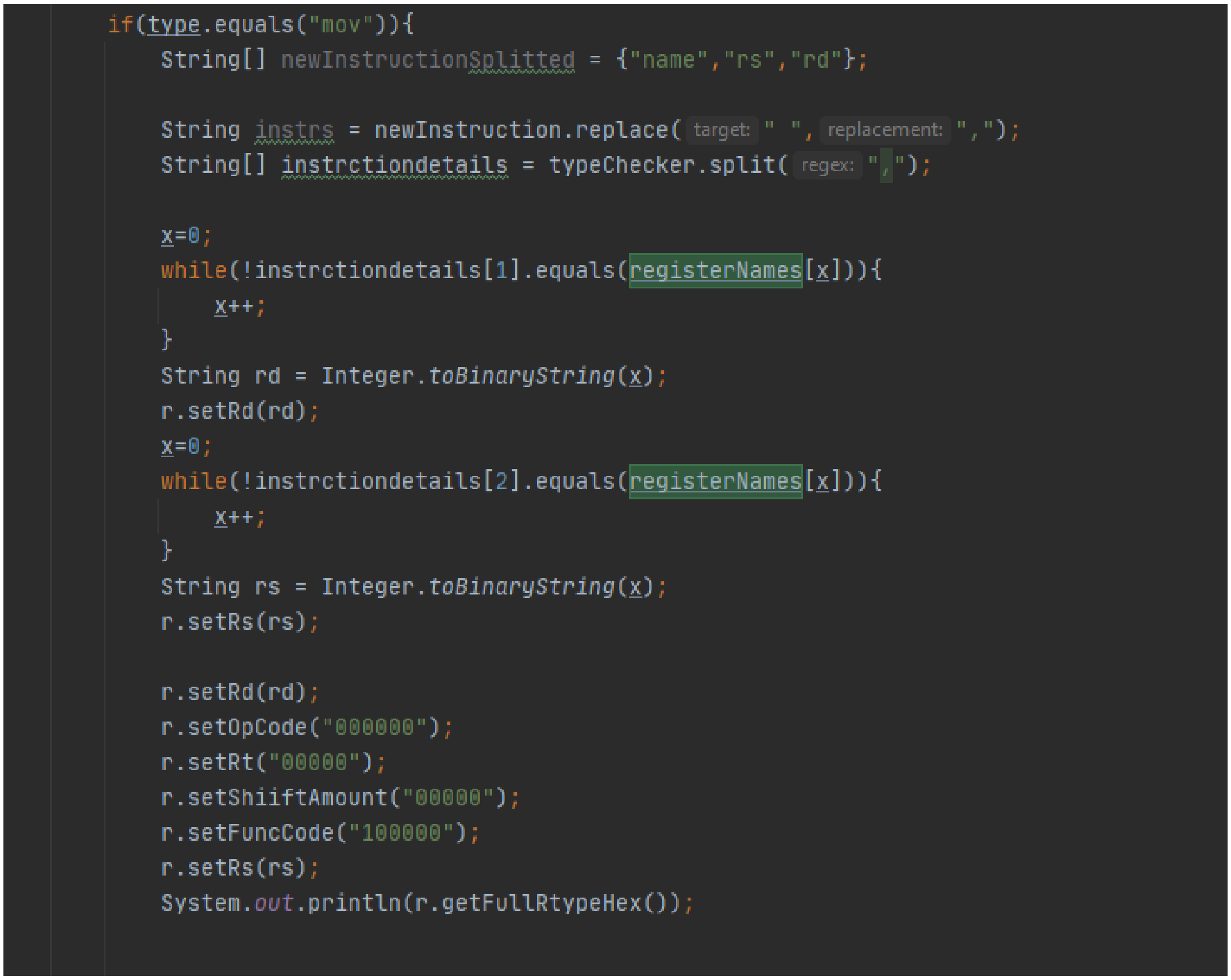
```
if(type.equals("slti")){
    String[] newInstructionSplitted3 = {"name","rs","rt","im"};
    int flag=0;
    for(int a=0;a<typeChecker2.length;a++){
        if(typeChecker2[a].length()!=0){
            newInstructionSplitted3[flag]=typeChecker2[a];
            flag++;
        }
    }
    for(int i=0;i<instructionListLookup.size();i++){
        if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){ ///  WE CHECK THE LOOKUPTABLE TO IDENTIFY REGISTER ADDRESSES
            index=index+1;      //// AND KEEP TRACK THE INDEX OF ITTT
            break;
        }
        index=index+1;
    }

    x=0;
    while(!newInstructionSplitted3[1].equals(registerNames[x])){            //// AND HERE WE ASSIGNED THEM BY CHECKING
        x++;
    }
    String rs = Integer.toBinaryString(x);

    x=0;
    while(!newInstructionSplitted3[2].equals(registerNames[x])){             //// AND HERE WE ASSIGNED THEM BY CHECKING
        x++;
    }
    String rt = Integer.toBinaryString(x);

    int q=Integer.parseInt(newInstructionSplitted3[3]);
```

For example ; sll not behaves like general R types.



```
if(type.equals("sll")){
    String[] newInstructionSplittedsll = {"name","rd","rt","shiftamount"};
    r.setOpCode("000000");
    r.setRs("00000");
    r.setFuncCode("000000");
    String instrs = newInstruction.replace( target: " ", replacement: ",");
    String[] instrctiondetails = typeChecker.split( regex: ",");
    x=0;
    while(!instrctiondetails[1].equals(registerNames[x])){
        x++;
    }
    String rd = Integer.toBinaryString(x);
    r.setRd(rd);

    x=0;
    while(!instrctiondetails[2].equals(registerNames[x])){
        x++;
    }
    String rt = Integer.toBinaryString(x);
    r.setRt(rt);
    System.out.println(r.getFullRtypeHex());

    int shamount = Integer.parseInt(instrctiondetails[3]);
    String shift = Integer.toBinaryString(shamount);
    r.setShiiftAmount(shift);

    System.out.println(r.getFullRtypeHex());
```

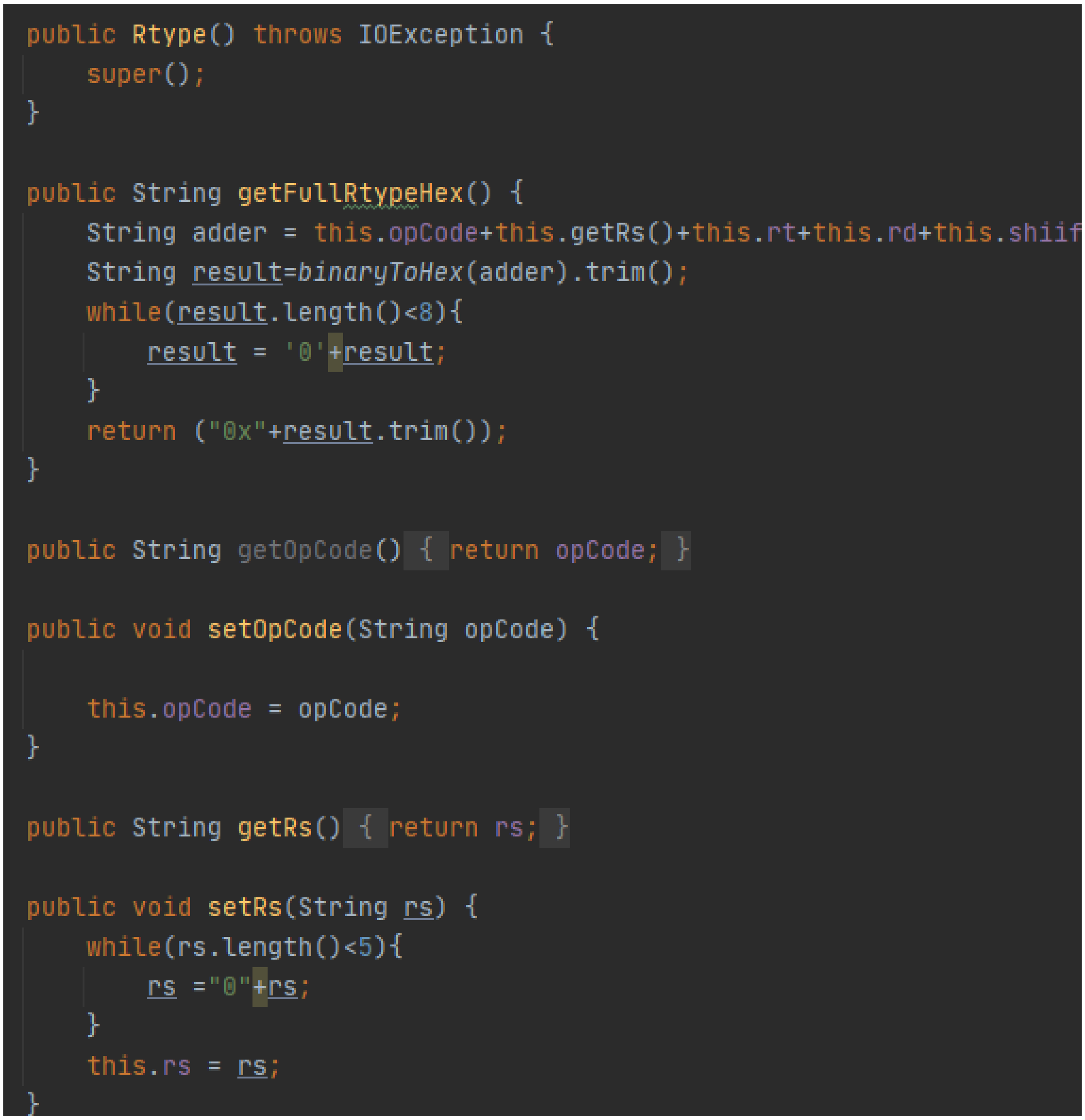
And for some special pseuodo codes like “move”;



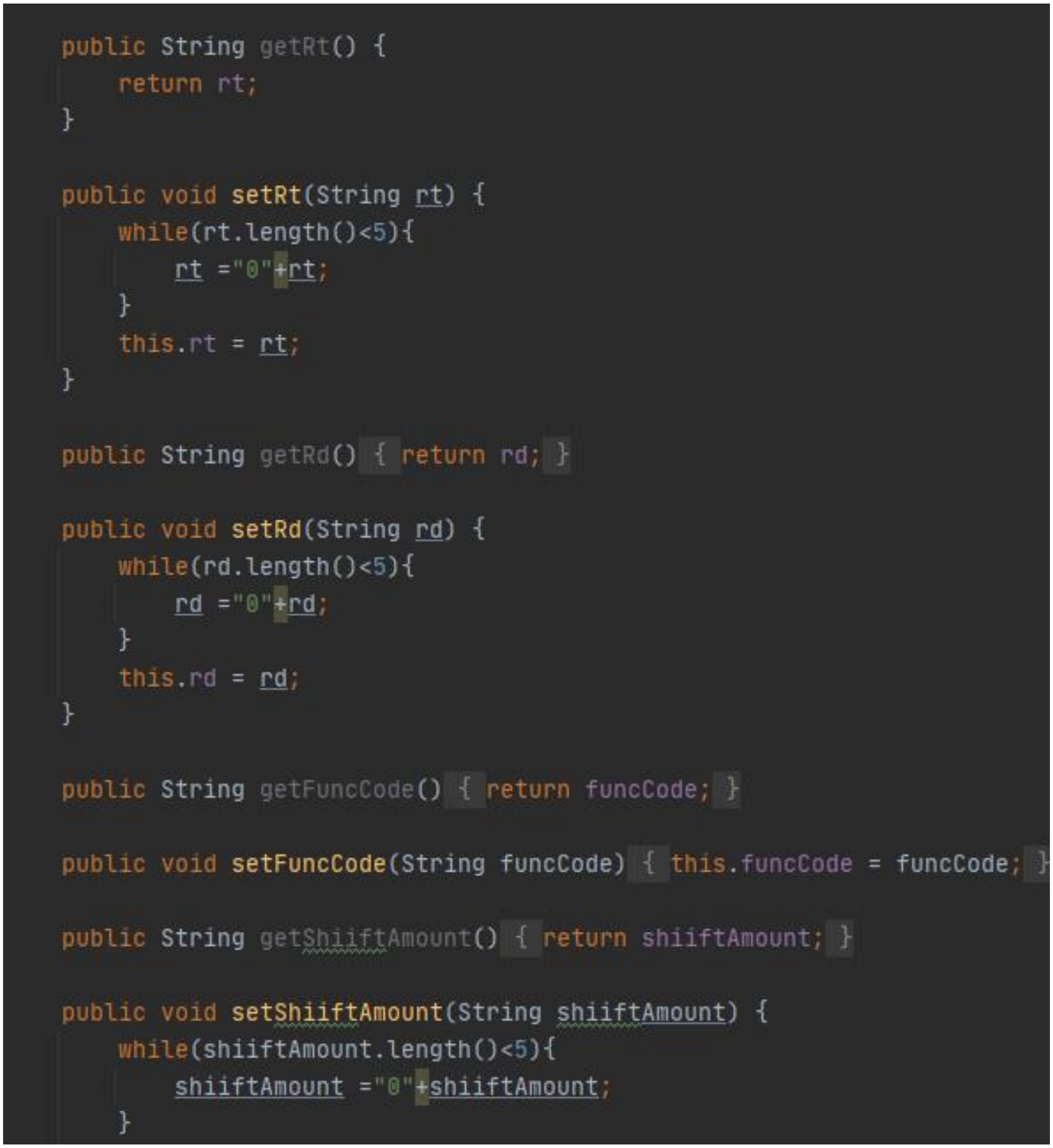
```
if(type.equals("mov")){
    String[] newInstructionSplitted = {"name","rs","rd"};

    String instrs = newInstruction.replace( target: " ", replacement: ",");
    String[] instrctiondetails = typeChecker.split( regex: ",");

    x=0;
    while(!instrctiondetails[1].equals(registerNames[x])){
        x++;
    }
    String rd = Integer.toBinaryString(x);
    r.setRd(rd);
    x=0;
    while(!instrctiondetails[2].equals(registerNames[x])){
        x++;
    }
    String rs = Integer.toBinaryString(x);
    r.setRs(rs);

    r.setRd(rd);
    r.setOpCode("000000");
    r.setRt("00000");
    r.setShiiftAmount("00000");
    r.setFuncCode("100000");
    r.setRs(rs);
    System.out.println(r.getFullRtypeHex());
```

For general R- TYPE :

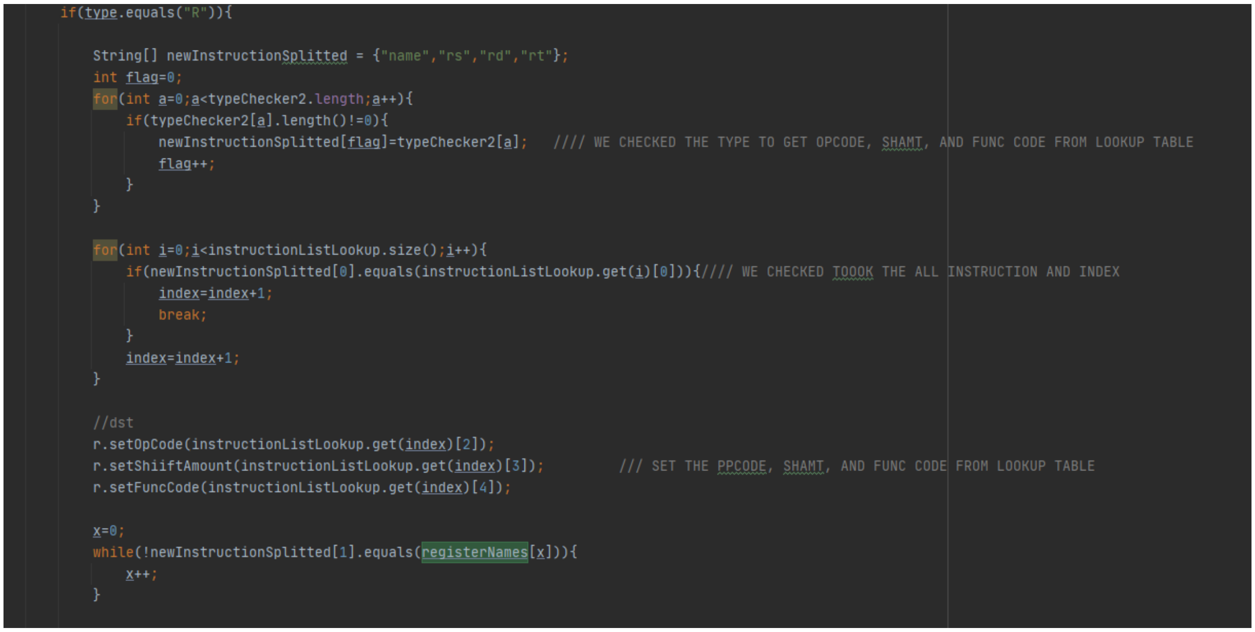
Here is our Object oriented R TYPE



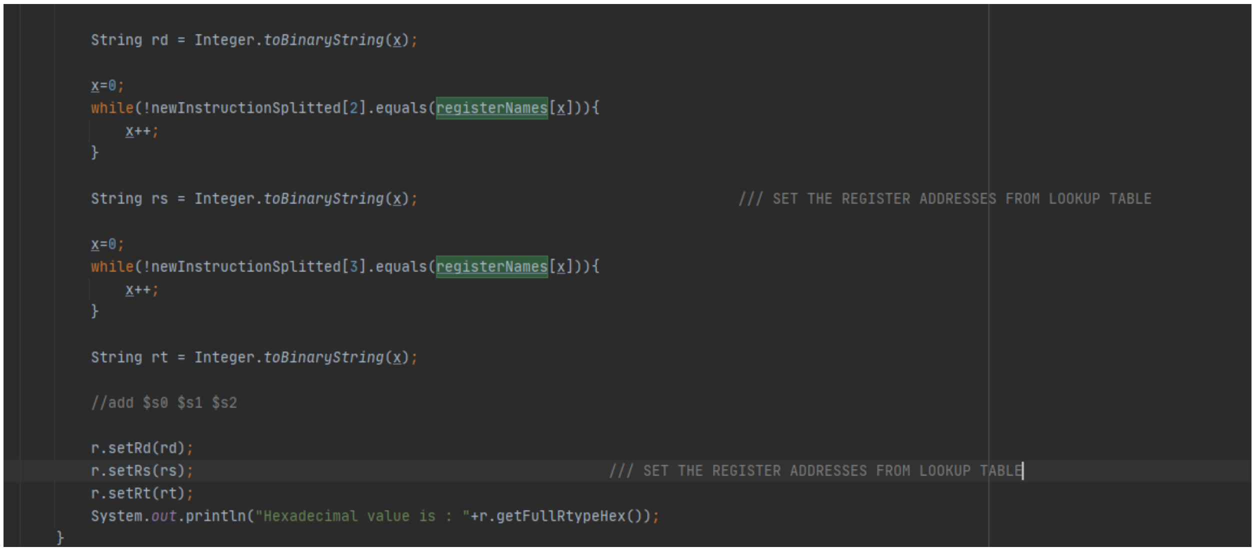
```
public Rtype() throws IOException {
    super();
}

public String getFullRtypeHex() {
    String adder = this.opCode+this.getRs()+this.rt+this.rd+this.shiif
    String result=binaryToHex(adder).trim();
    while(result.length()<8){
        result = '0'+result;
    }
    return ("0x"+result.trim());
}

public String getOpCode() { return opCode; }

public void setOpCode(String opCode) {

    this.opCode = opCode;
}

public String getRs() { return rs; }

public void setRs(String rs) {
    while(rs.length()<5){
        rs ="0"+rs;
    }
    this.rs = rs;
}
```



```
public String getRt() {
    return rt;
}

public void setRt(String rt) {
    while(rt.length()<5){
        rt ="0"+rt;
    }
    this.rt = rt;
}

public String getRd() { return rd; }

public void setRd(String rd) {
    while(rd.length()<5){
        rd ="0"+rd;
    }
    this.rd = rd;
}

public String getFuncCode() { return funcCode; }

public void setFuncCode(String funcCode) { this.funcCode = funcCode; }

public String getShiiftAmount() { return shiiftAmount; }

public void setShiiftAmount(String shiiftAmount) {
    while(shiiftAmount.length()<5){
        shiiftAmount ="0"+shiiftAmount;
    }
```



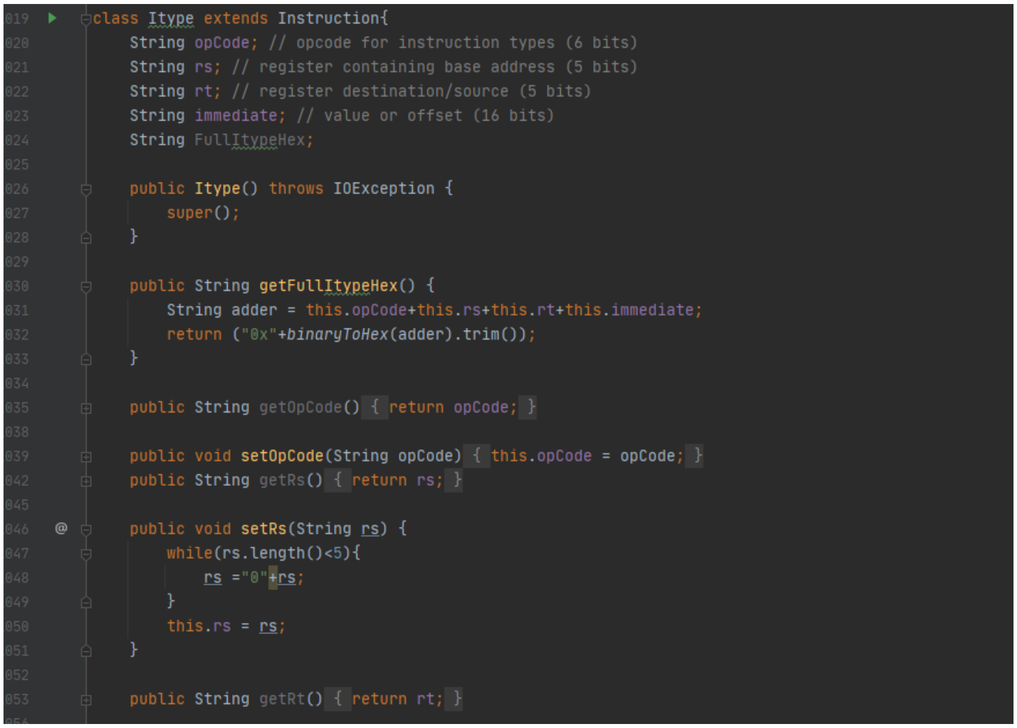
```
if(type.equals("R")){

    String[] newInstructionSplitted = {"name","rs","rd","rt"};
    int flag=0;
    for(int a=0;a<typeChecker2.length;a++){
        if(typeChecker2[a].length()!=0){
            newInstructionSplitted[flag]=typeChecker2[a];   //// WE CHECKED THE TYPE TO GET OPCODE, SHAMT, AND FUNC CODE FROM LOOKUP TABLE
            flag++;
        }
    }

    for(int i=0;i<instructionListLookup.size();i++){
        if(newInstructionSplitted[0].equals(instructionListLookup.get(i)[0])){//// WE CHECKED TOOOK THE ALL INSTRUCTION AND INDEX
            index=index+1;
            break;
        }
        index=index+1;
    }

    //dst
    r.setOpCode(instructionListLookup.get(index)[2]);
    r.setShiiftAmount(instructionListLookup.get(index)[3]);         /// SET THE PPCODE, SHAMT, AND FUNC CODE FROM LOOKUP TABLE
    r.setFuncCode(instructionListLookup.get(index)[4]);

    x=0;
    while(!newInstructionSplitted[1].equals(registerNames[x])){
        x++;
    }
```



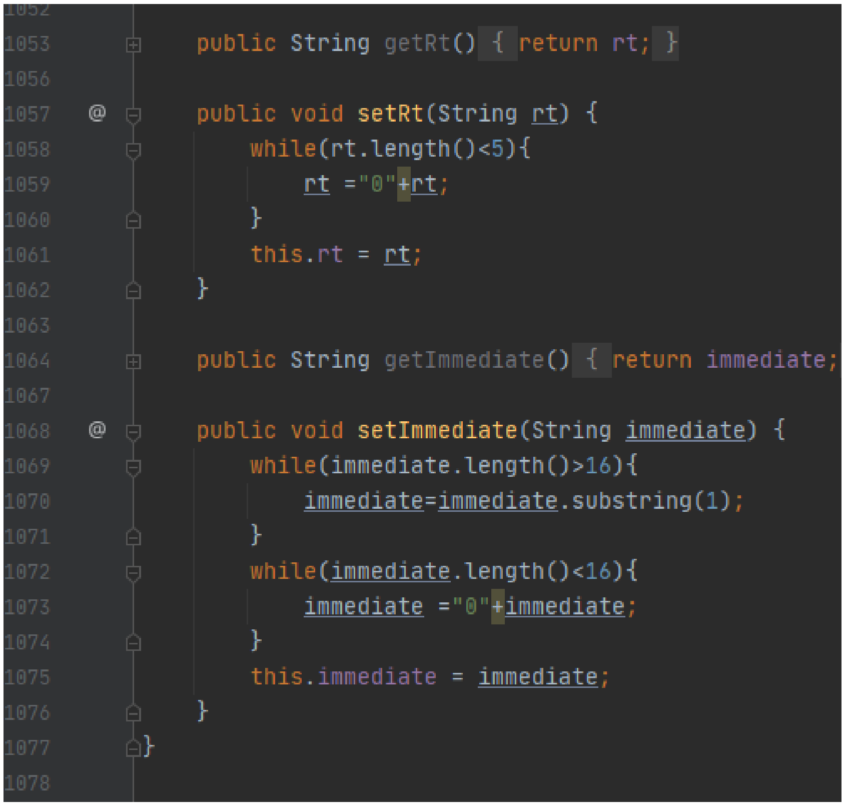
```
    String rd = Integer.toBinaryString(x);

    x=0;
    while(!newInstructionSplitted[2].equals(registerNames[x])){
        x++;
    }

    String rs = Integer.toBinaryString(x);                                     /// SET THE REGISTER ADDRESSES FROM LOOKUP TABLE

    x=0;
    while(!newInstructionSplitted[3].equals(registerNames[x])){
        x++;
    }

    String rt = Integer.toBinaryString(x);

    //add $s0 $s1 $s2

    r.setRd(rd);
    r.setRs(rs);                                                /// SET THE REGISTER ADDRESSES FROM LOOKUP TABLE
    r.setRt(rt);
    System.out.println("Hexadecimal value is : "+r.getFullRtypeHex());
}
```

For general I- TYPE :

Here is our Object oriented I TYPE

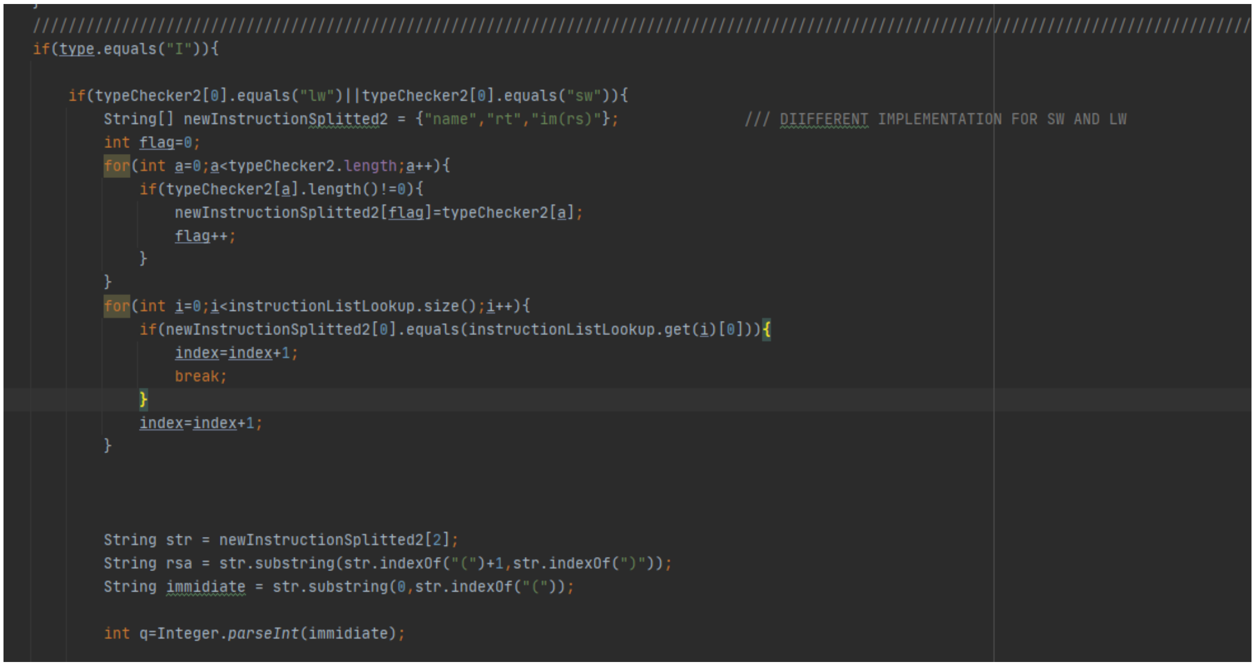


```
class Itype extends Instruction{
    String opCode; // opcode for instruction types (6 bits)
    String rs; // register containing base address (5 bits)
    String rt; // register destination/source (5 bits)
    String immediate; // value or offset (16 bits)
    String FullItypeHex;

    public Itype() throws IOException {
        super();
    }

    public String getFullItypeHex() {
        String adder = this.opCode+this.rs+this.rt+this.immediate;
        return ("0x"+binaryToHex(adder).trim());
    }

    public String getOpCode() { return opCode; }

    public void setOpCode(String opCode) { this.opCode = opCode; }
    public String getRs() { return rs; }

    public void setRs(String rs) {
        while(rs.length()<5){
            rs ="0"+rs;
        }
        this.rs = rs;
    }

    public String getRt() { return rt; }
```

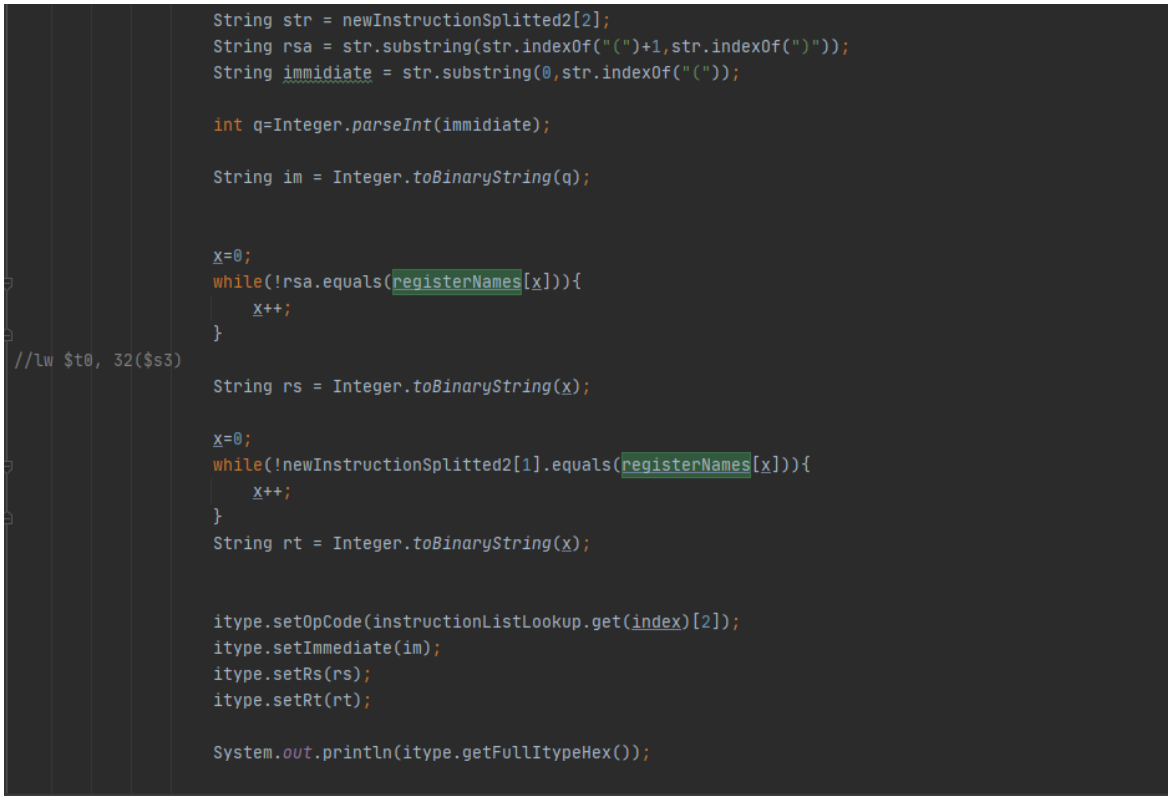


```
    public String getRt() { return rt; }

    public void setRt(String rt) {
        while(rt.length()<5){
            rt ="0"+rt;
        }
        this.rt = rt;
    }

    public String getImmediate() { return immediate;

    public void setImmediate(String immediate) {
        while(immediate.length()>16){
            immediate=immediate.substring(1);
        }
        while(immediate.length()<16){
            immediate ="0"+immediate;
        }
        this.immediate = immediate;
    }
}
```

If its SW OR LW

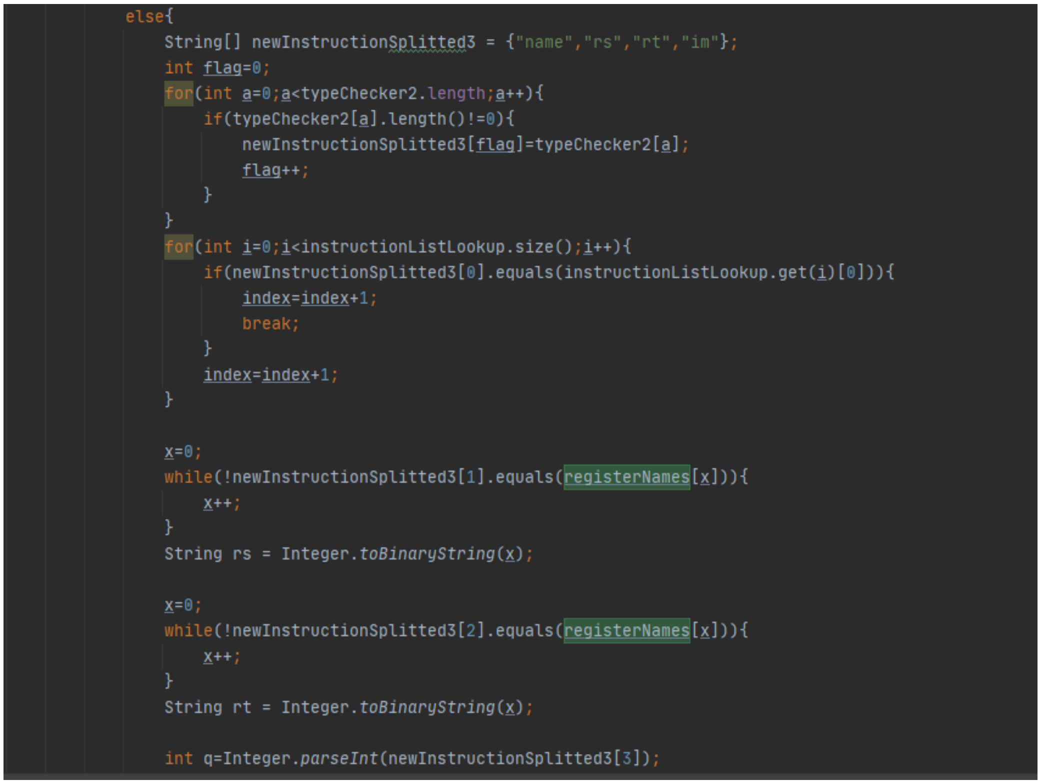


```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
if(type.equals("I")){

    if(typeChecker2[0].equals("lw")||typeChecker2[0].equals("sw")){
        String[] newInstructionSplitted2 = {"name","rt","im(rs)"};              /// DIIFFERENT IMPLEMENTATION FOR SW AND LW
        int flag=0;
        for(int a=0;a<typeChecker2.length;a++){
            if(typeChecker2[a].length()!=0){
                newInstructionSplitted2[flag]=typeChecker2[a];
                flag++;
            }
        }
        for(int i=0;i<instructionListLookup.size();i++){
            if(newInstructionSplitted2[0].equals(instructionListLookup.get(i)[0])){
                index=index+1;
                break;
            }
            index=index+1;
        }



        String str = newInstructionSplitted2[2];
        String rsa = str.substring(str.indexOf("(")+1,str.indexOf(")"));
        String immidiate = str.substring(0,str.indexOf("("));

        int q=Integer.parseInt(immidiate);
```

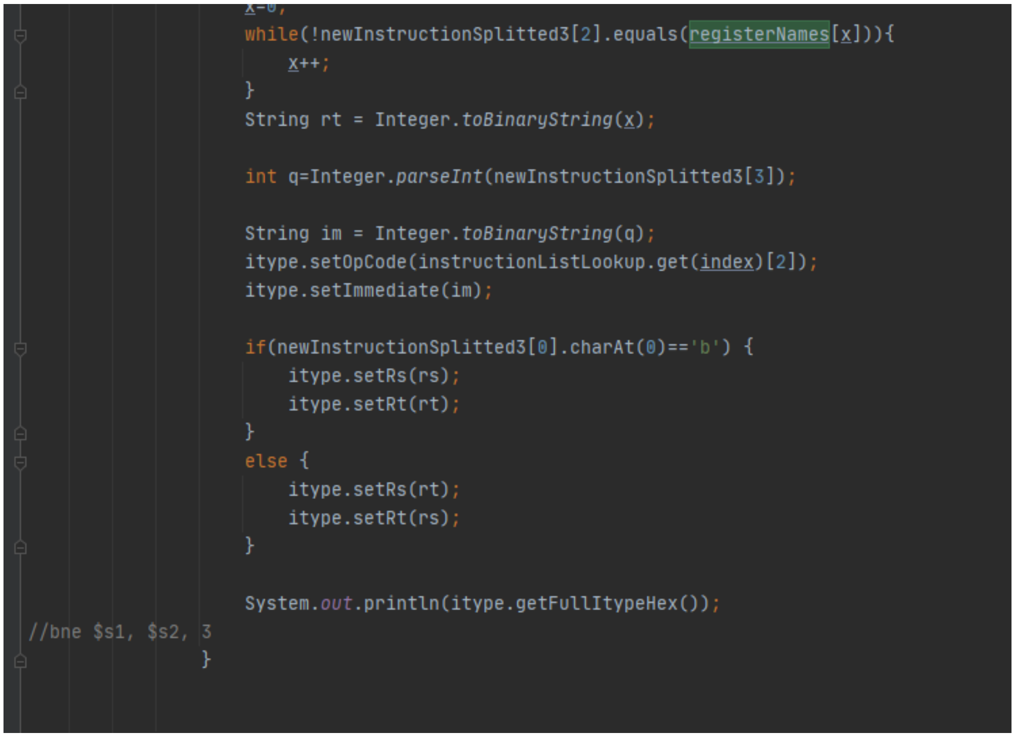


```
             String str = newInstructionSplitted2[2];
             String rsa = str.substring(str.indexOf("(")+1,str.indexOf(")"));
             String immidiate = str.substring(0,str.indexOf("("));

             int q=Integer.parseInt(immidiate);

             String im = Integer.toBinaryString(q);


             x=0;
             while(!rsa.equals(registerNames[x])){
                 x++;
             }
//lw $t0, 32($s3)
             String rs = Integer.toBinaryString(x);

             x=0;
             while(!newInstructionSplitted2[1].equals(registerNames[x])){
                 x++;
             }
             String rt = Integer.toBinaryString(x);


             itype.setOpCode(instructionListLookup.get(index)[2]);
             itype.setImmediate(im);
             itype.setRs(rs);
             itype.setRt(rt);

             System.out.println(itype.getFullItypeHex());
```

IF NOT ;

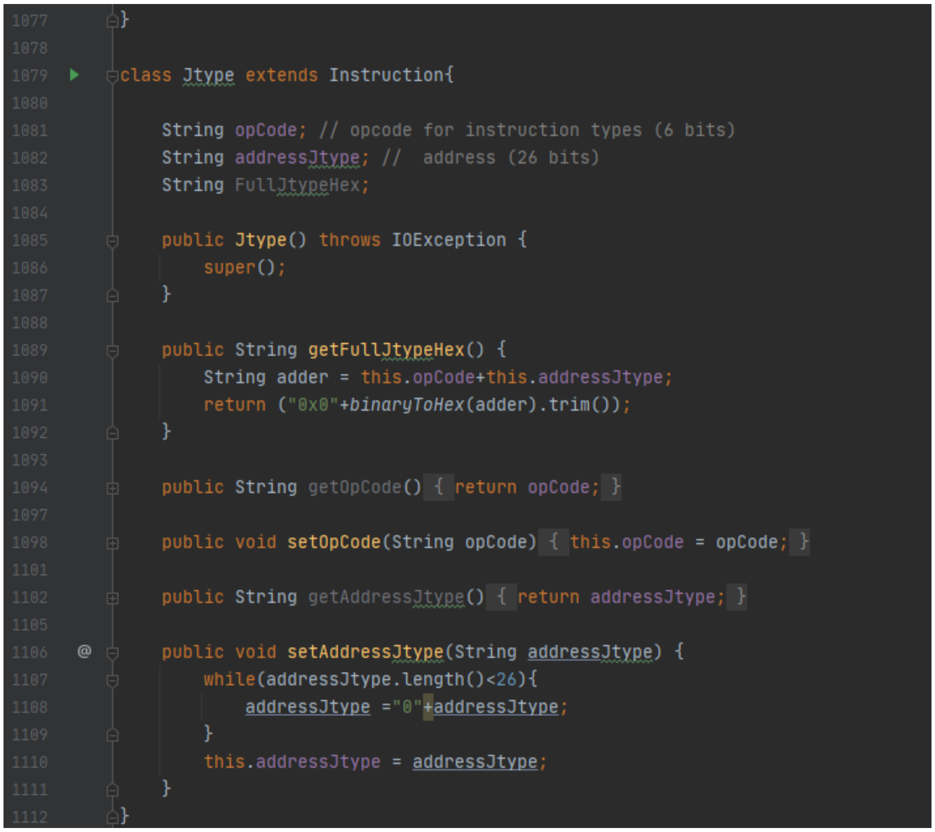


```
else{
    String[] newInstructionSplitted3 = {"name","rs","rt","im"};
    int flag=0;
    for(int a=0;a<typeChecker2.length;a++){
        if(typeChecker2[a].length()!=0){
            newInstructionSplitted3[flag]=typeChecker2[a];
            flag++;
        }
    }
    for(int i=0;i<instructionListLookup.size();i++){
        if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){
            index=index+1;
            break;
        }
        index=index+1;
    }

    x=0;
    while(!newInstructionSplitted3[1].equals(registerNames[x])){
        x++;
    }
    String rs = Integer.toBinaryString(x);

    x=0;
    while(!newInstructionSplitted3[2].equals(registerNames[x])){
        x++;
    }
    String rt = Integer.toBinaryString(x);

    int q=Integer.parseInt(newInstructionSplitted3[3]);
```

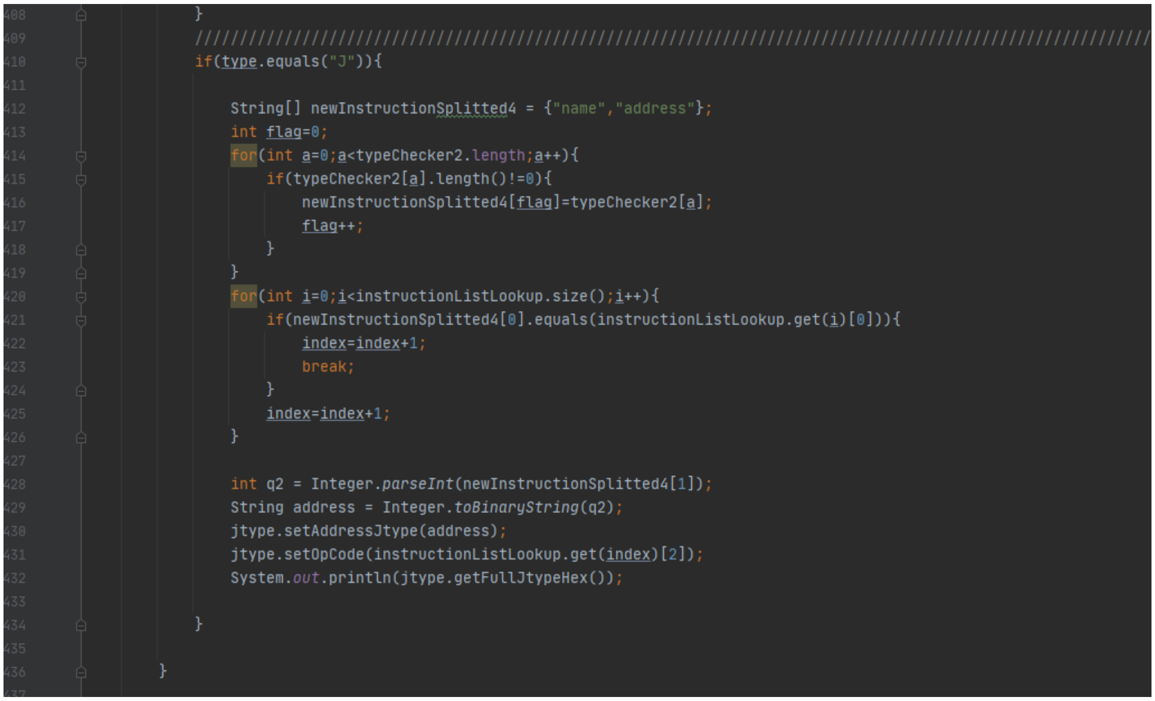


```
                    x=0;
                    while(!newInstructionSplitted3[2].equals(registerNames[x])){
                        x++;
                    }
                    String rt = Integer.toBinaryString(x);

                    int q=Integer.parseInt(newInstructionSplitted3[3]);

                    String im = Integer.toBinaryString(q);
                    itype.setOpCode(instructionListLookup.get(index)[2]);
                    itype.setImmediate(im);

                    if(newInstructionSplitted3[0].charAt(0)=='b') {
                        itype.setRs(rs);
                        itype.setRt(rt);
                    }
                    else {
                        itype.setRs(rt);
                        itype.setRt(rs);
                    }

                    System.out.println(itype.getFullItypeHex());
//bne $s1, $s2, 3
                }
```

Here is our Object oriented J TYPE



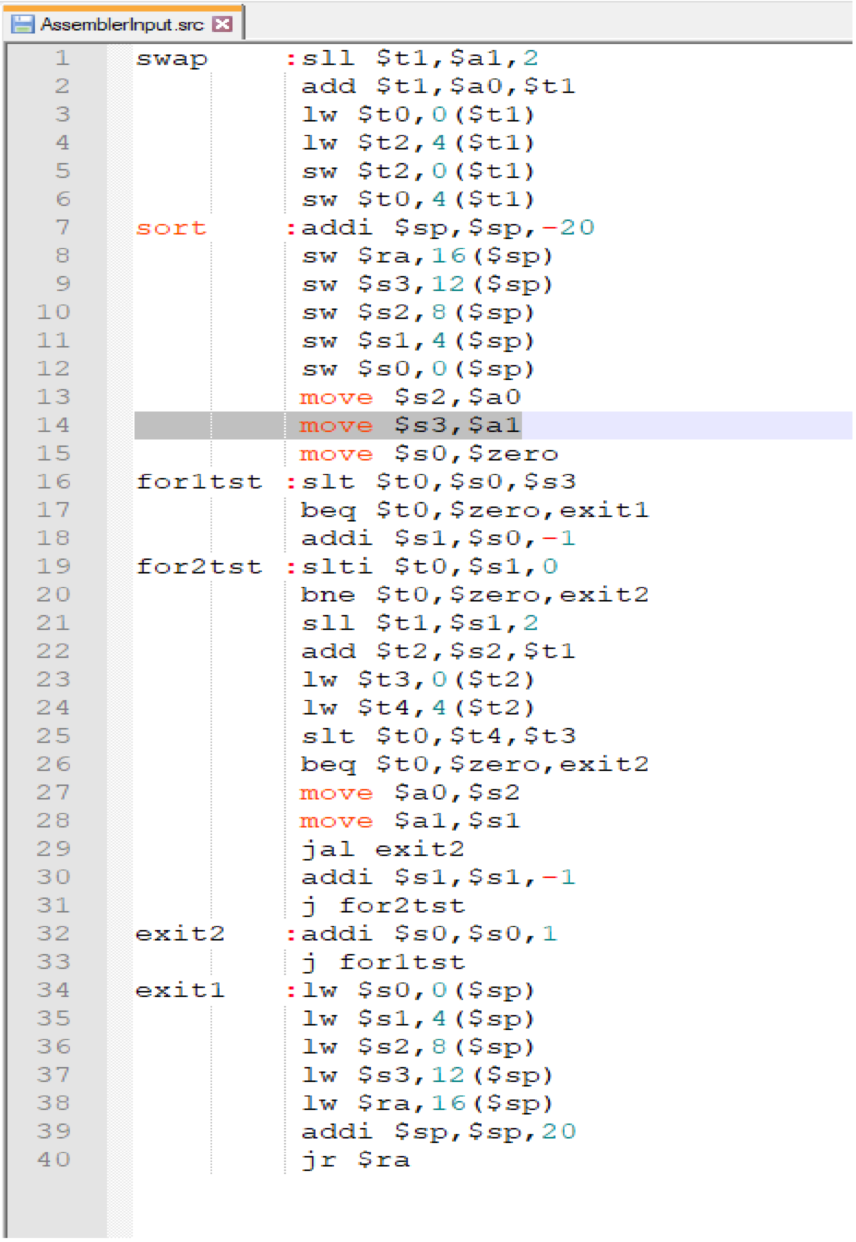
```java
}

class Jtype extends Instruction{

    String opCode; // opcode for instruction types (6 bits)
    String addressJtype; //  address (26 bits)
    String FullJtypeHex;

    public Jtype() throws IOException {
        super();
    }

    public String getFullJtypeHex() {
        String adder = this.opCode+this.addressJtype;
        return ("0x0"+binaryToHex(adder).trim());
    }

    public String getOpCode() { return opCode; }

    public void setOpCode(String opCode) { this.opCode = opCode; }

    public String getAddressJtype() { return addressJtype; }

    public void setAddressJtype(String addressJtype) {
        while(addressJtype.length()<26){
            addressJtype ="0"+addressJtype;
        }
        this.addressJtype = addressJtype;
    }
}
```



```java
        }
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        if(type.equals("J")){

            String[] newInstructionSplitted4 = {"name","address"};
            int flag=0;
            for(int a=0;a<typeChecker2.length;a++){
                if(typeChecker2[a].length()!=0){
                    newInstructionSplitted4[flag]=typeChecker2[a];
                    flag++;
                }
            }
            for(int i=0;i<instructionListLookup.size();i++){
                if(newInstructionSplitted4[0].equals(instructionListLookup.get(i)[0])){
                    index=index+1;
                    break;
                }
                index=index+1;
            }

            int q2 = Integer.parseInt(newInstructionSplitted4[1]);
            String address = Integer.toBinaryString(q2);
            jtype.setAddressJtype(address);
            jtype.setOpCode(instructionListLookup.get(index)[2]);
            System.out.println(jtype.getFullJtypeHex());

        }

    }
```

How Batch Mode works

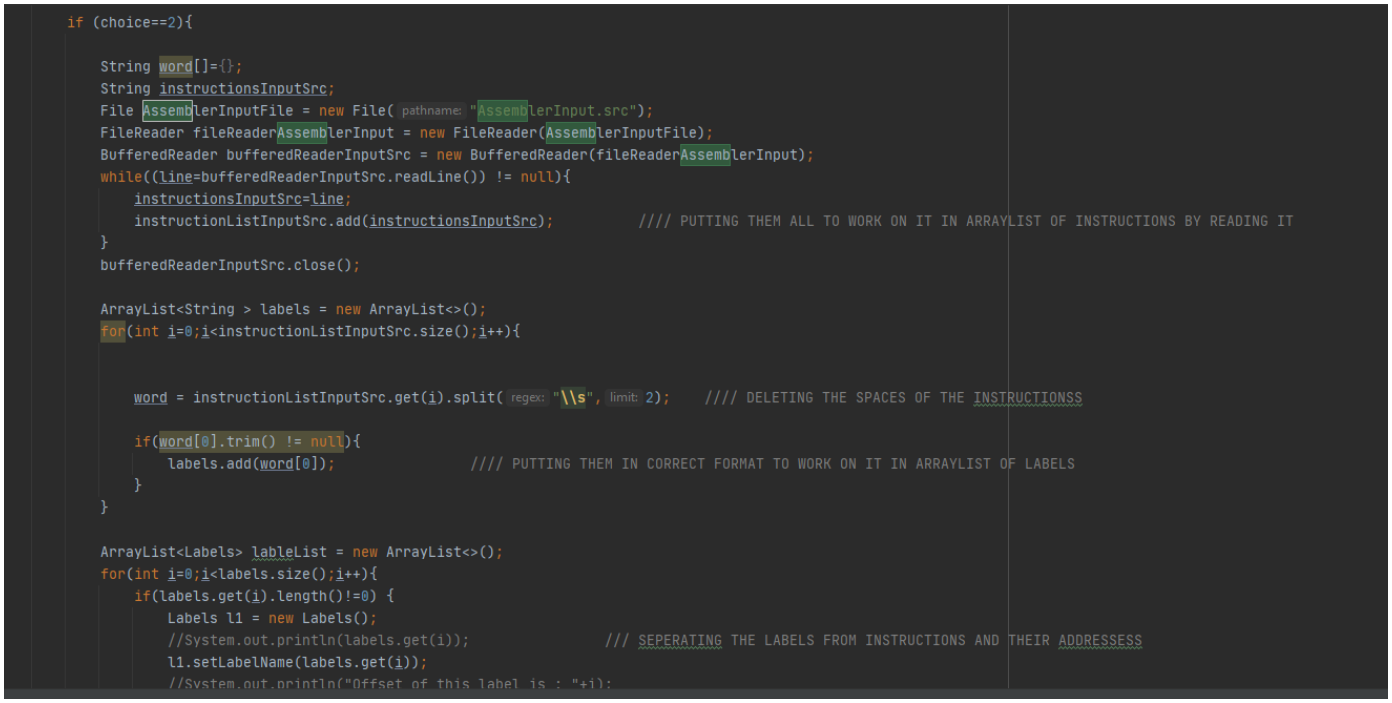
IT WORKS WITH THE SAME CODE LOGIC AS THE INTERACTIVE MODE, BUT IN HERE BESIDES THE SHOWING RESULTS, IT GETS THE INSTRUCTIONS FROM OUR “SRC” FILE OF INPUTS “AssemblerInput.src” ;



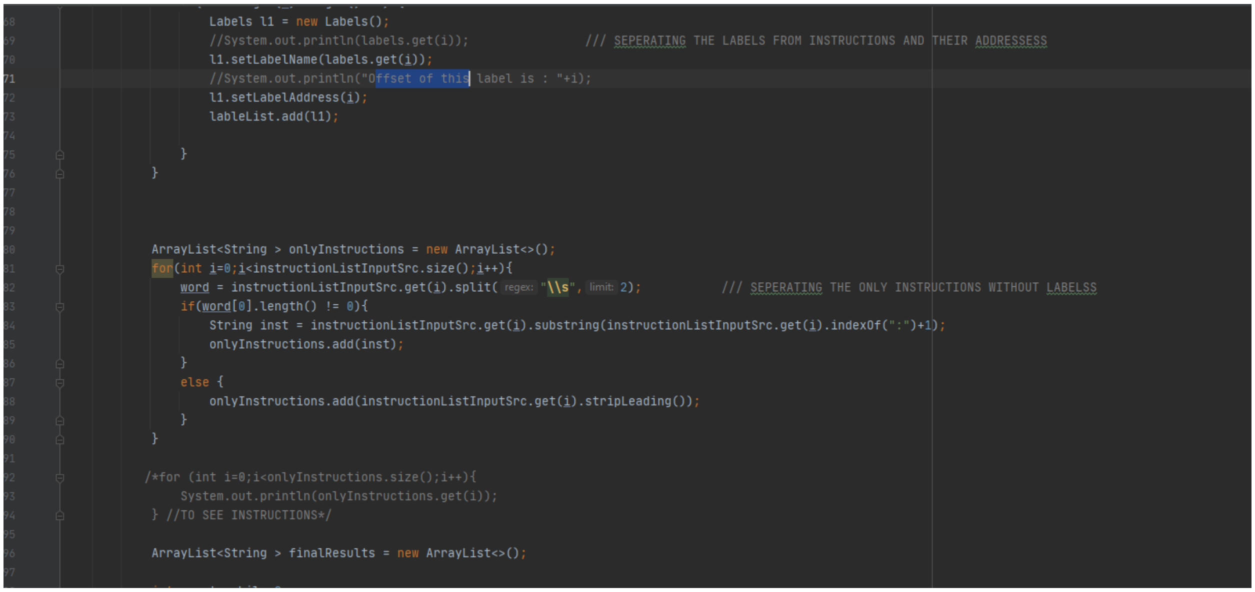
AssemblerInput.src

```
swap     :sll $t1,$a1,2
          add $t1,$a0,$t1
          lw $t0,0($t1)
          lw $t2,4($t1)
          sw $t2,0($t1)
          sw $t0,4($t1)
sort     :addi $sp,$sp,-20
          sw $ra,16($sp)
          sw $s3,12($sp)
          sw $s2,8($sp)
          sw $s1,4($sp)
          sw $s0,0($sp)
          move $s2,$a0
          move $s3,$a1
          move $s0,$zero
for1tst  :slt $t0,$s0,$s3
          beq $t0,$zero,exit1
          addi $s1,$s0,-1
for2tst  :slti $t0,$s1,0
          bne $t0,$zero,exit2
          sll $t1,$s1,2
          add $t2,$s2,$t1
          lw $t3,0($t2)
          lw $t4,4($t2)
          slt $t0,$t4,$t3
          beq $t0,$zero,exit2
          move $a0,$s2
          move $a1,$s1
          jal exit2
          addi $s1,$s1,-1
          j for2tst
exit2    :addi $s0,$s0,1
          j for1tst
exit1    :lw $s0,0($sp)
          lw $s1,4($sp)
          lw $s2,8($sp)
          lw $s3,12($sp)
          lw $ra,16($sp)
          addi $sp,$sp,20
          jr $ra
```

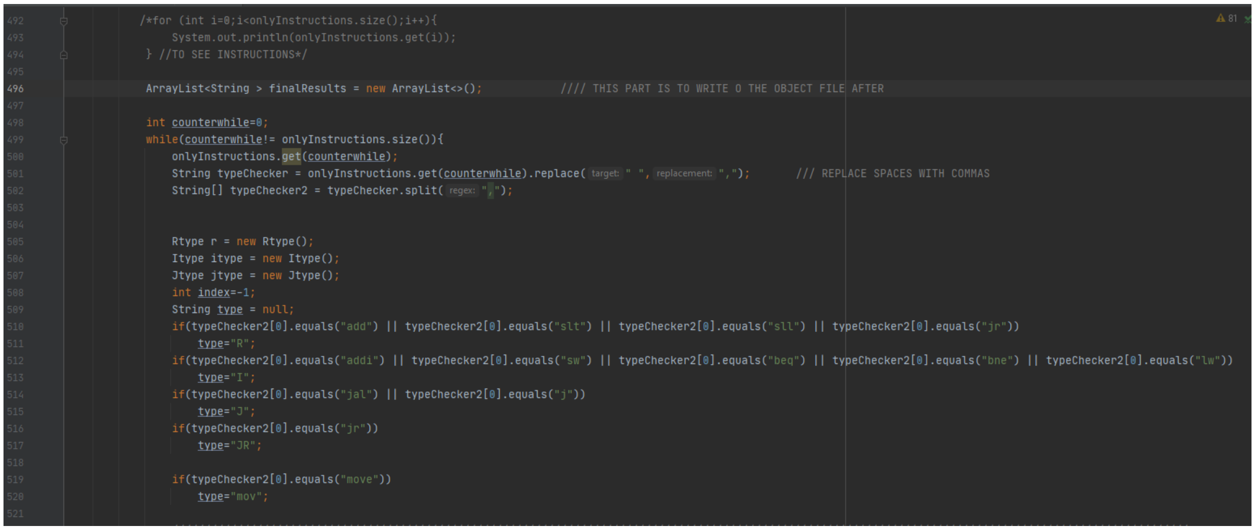
With this part of code ;



```
if (choice==2){

    String word[]={};
    String instructionsInputSrc;
    File AssemblerInputFile = new File( pathname: "AssemblerInput.src");
    FileReader fileReaderAssemblerInput = new FileReader(AssemblerInputFile);
    BufferedReader bufferedReaderInputSrc = new BufferedReader(fileReaderAssemblerInput);
    while((line=bufferedReaderInputSrc.readLine()) != null){
        instructionsInputSrc=line;
        instructionListInputSrc.add(instructionsInputSrc);          //// PUTTING THEM ALL TO WORK ON IT IN ARRAYLIST OF INSTRUCTIONS BY READING IT
    }
    bufferedReaderInputSrc.close();

    ArrayList<String > labels = new ArrayList<>();
    for(int i=0;i<instructionListInputSrc.size();i++){


        word = instructionListInputSrc.get(i).split( regex: "\\s", limit: 2);    //// DELETING THE SPACES OF THE INSTRUCTIONSS

        if(word[0].trim() != null){
            labels.add(word[0]);                //// PUTTING THEM IN CORRECT FORMAT TO WORK ON IT IN ARRAYLIST OF LABELS
        }
    }

    ArrayList<Labels> lableList = new ArrayList<>();
    for(int i=0;i<labels.size();i++){
        if(labels.get(i).length()!=0) {
            Labels l1 = new Labels();
            //System.out.println(labels.get(i));                /// SEPERATING THE LABELS FROM INSTRUCTIONS AND THEIR ADDRESSESS
            l1.setLabelName(labels.get(i));
            //System.out.println("Offset of this label is : "+i);
```

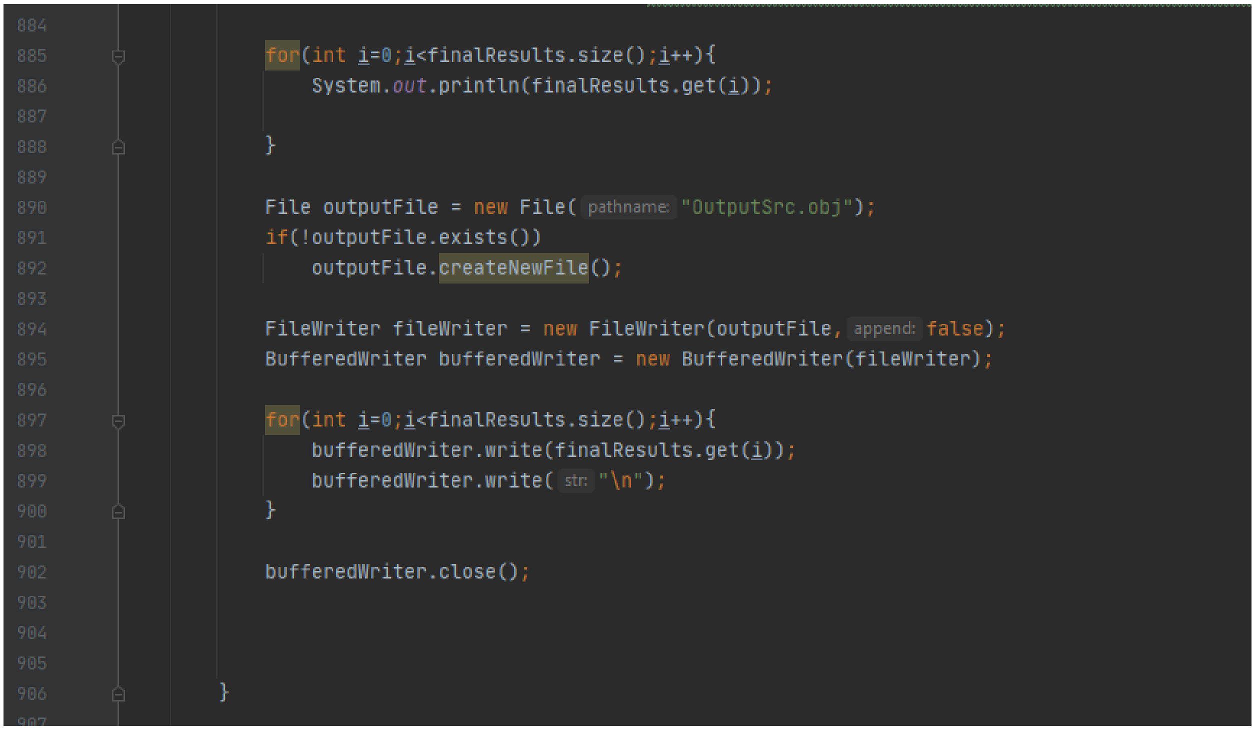


```
            Labels l1 = new Labels();
            //System.out.println(labels.get(i));                /// SEPERATING THE LABELS FROM INSTRUCTIONS AND THEIR ADDRESSESS
            l1.setLabelName(labels.get(i));
            //System.out.println("Offset of this label is : "+i);
            l1.setLabelAddress(i);
            lableList.add(l1);

        }
    }



    ArrayList<String > onlyInstructions = new ArrayList<>();
    for(int i=0;i<instructionListInputSrc.size();i++){
        word = instructionListInputSrc.get(i).split( regex: "\\s", limit: 2);          /// SEPERATING THE ONLY INSTRUCTIONS WITHOUT LABELSS
        if(word[0].length() != 0){
            String inst = instructionListInputSrc.get(i).substring(instructionListInputSrc.get(i).indexOf(":")+1);
            onlyInstructions.add(inst);
        }
        else {
            onlyInstructions.add(instructionListInputSrc.get(i).stripLeading());
        }
    }

   /*for (int i=0;i<onlyInstructions.size();i++){
        System.out.println(onlyInstructions.get(i));
    } //TO SEE INSTRUCTIONS*/

    ArrayList<String > finalResults = new ArrayList<>();
```



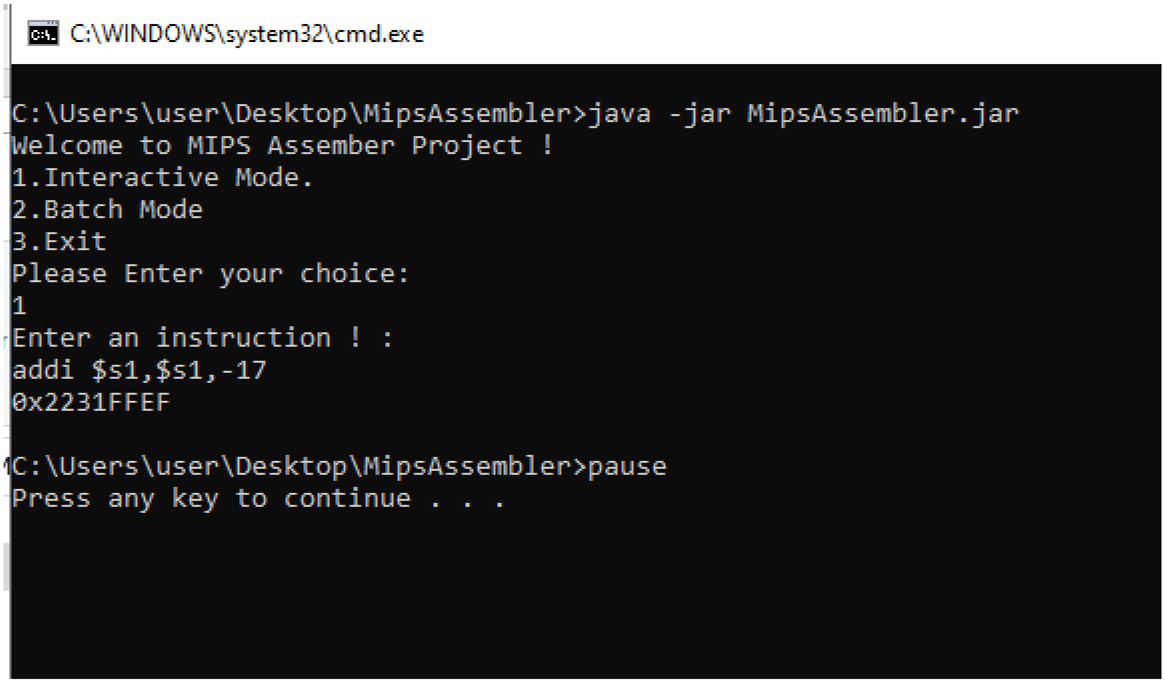
```
/*for (int i=0;i<onlyInstructions.size();i++){
    System.out.println(onlyInstructions.get(i));
} //TO SEE INSTRUCTIONS*/

ArrayList<String > finalResults = new ArrayList<>();            //// THIS PART IS TO WRITE O THE OBJECT FILE AFTER

int counterwhile=0;
while(counterwhile!= onlyInstructions.size()){
    onlyInstructions.get(counterwhile);
    String typeChecker = onlyInstructions.get(counterwhile).replace( target: " ", replacement: ",");       /// REPLACE SPACES WITH COMMAS
    String[] typeChecker2 = typeChecker.split( regex: ",");


    Rtype r = new Rtype();
    Itype itype = new Itype();
    Jtype jtype = new Jtype();
    int index=-1;
    String type = null;
    if(typeChecker2[0].equals("add") || typeChecker2[0].equals("slt") || typeChecker2[0].equals("sll") || typeChecker2[0].equals("jr"))
        type="R";
    if(typeChecker2[0].equals("addi") || typeChecker2[0].equals("sw") || typeChecker2[0].equals("beq") || typeChecker2[0].equals("bne") || typeChecker2[0].equals("lw"))
        type="I";
    if(typeChecker2[0].equals("jal") || typeChecker2[0].equals("j"))
        type="J";
    if(typeChecker2[0].equals("jr"))
        type="JR";

    if(typeChecker2[0].equals("move"))
        type="mov";

```

AND FINALLY IN THIS PART IN A SERIALIZABLE SECURE OBJECT OF “OutputSrc.obj” IS WRITTEN IN TO IT FOR YOU TO CHECK



```

        for(int i=0;i<finalResults.size();i++){
            System.out.println(finalResults.get(i));

        }

        File outputFile = new File( pathname: "OutputSrc.obj");
        if(!outputFile.exists())
            outputFile.createNewFile();

        FileWriter fileWriter = new FileWriter(outputFile, append: false);
        BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

        for(int i=0;i<finalResults.size();i++){
            bufferedWriter.write(finalResults.get(i));
            bufferedWriter.write( str: "\n");
        }

        bufferedWriter.close();



    }
```

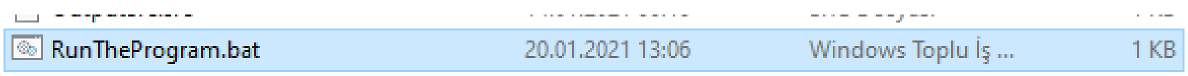
Screen capture soft copy of a session to show how your program handles the input instruction “addi $s1, $s1, -17” in interactive mode.



```
C:\Users\user\Desktop\MipsAssembler>java -jar MipsAssembler.jar
Welcome to MIPS Assember Project !
1.Interactive Mode.
2.Batch Mode
3.Exit
Please Enter your choice:
1
Enter an instruction ! :
addi $s1,$s1,-17
0x2231FFEF

C:\Users\user\Desktop\MipsAssembler>pause
Press any key to continue . . .
```

How to run the program ;

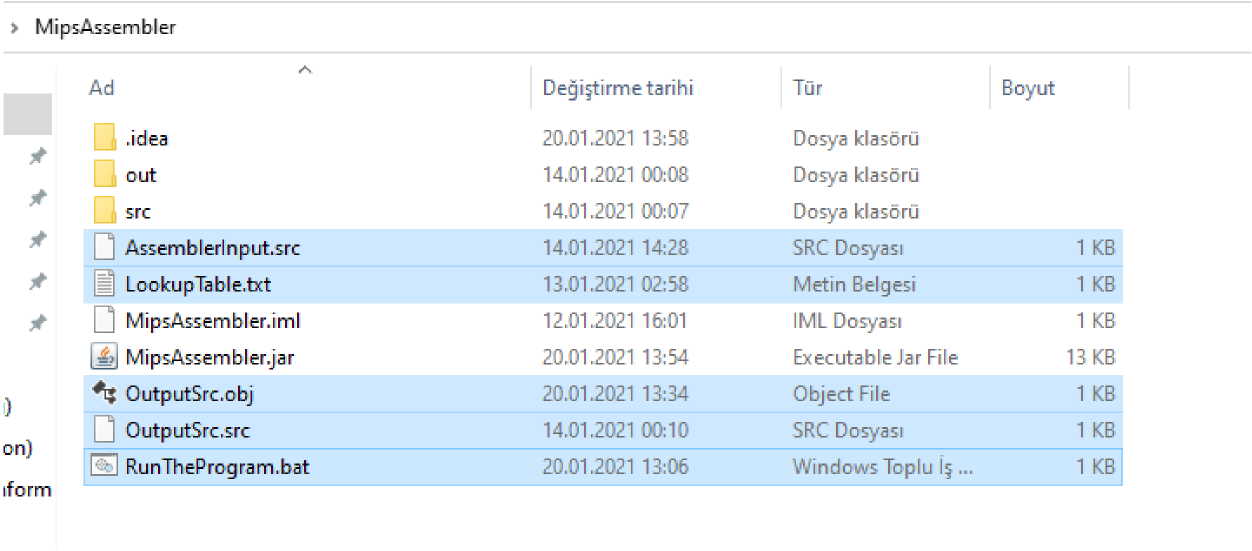
You can put the code to a compiler to see how initial program works as well too, Also we created a bat file for you to run easily (executable)



RunTheProgram.bat 20.01.2021 13:06 Windows Toplu İş ... 1 KB

Just double click the file

You can also see our external files as well.



› MipsAssembler

Ad	Değiştirme tarihi	Tür	Boyut
.idea	20.01.2021 13:58	Dosya klasörü	
out	14.01.2021 00:08	Dosya klasörü	
src	14.01.2021 00:07	Dosya klasörü	
AssemblerInput.src	14.01.2021 14:28	SRC Dosyası	1 KB
LookupTable.txt	13.01.2021 02:58	Metin Belgesi	1 KB
MipsAssembler.iml	12.01.2021 16:01	IML Dosyası	1 KB
MipsAssembler.jar	20.01.2021 13:54	Executable Jar File	13 KB
OutputSrc.obj	20.01.2021 13:34	Object File	1 KB
OutputSrc.src	14.01.2021 00:10	SRC Dosyası	1 KB
RunTheProgram.bat	20.01.2021 13:06	Windows Toplu İş ...	1 KB

Conclusion

Correctness – it does the everything right exactly.

Readability – We pay attention to structure, modularity i.e. use of function calls for common tasks, proper commenting of the code with very understandable variable names.

Flexibility – It is very easy   to extend support to a new instruction added to the MIPS ISA thanks to the our design and lookup table and Harcoded things are perfectly available.

User friendliness – It can handle errors made by the assembly programmer. Gives a good idea to the user on what to fix. program handle labels properly in the program

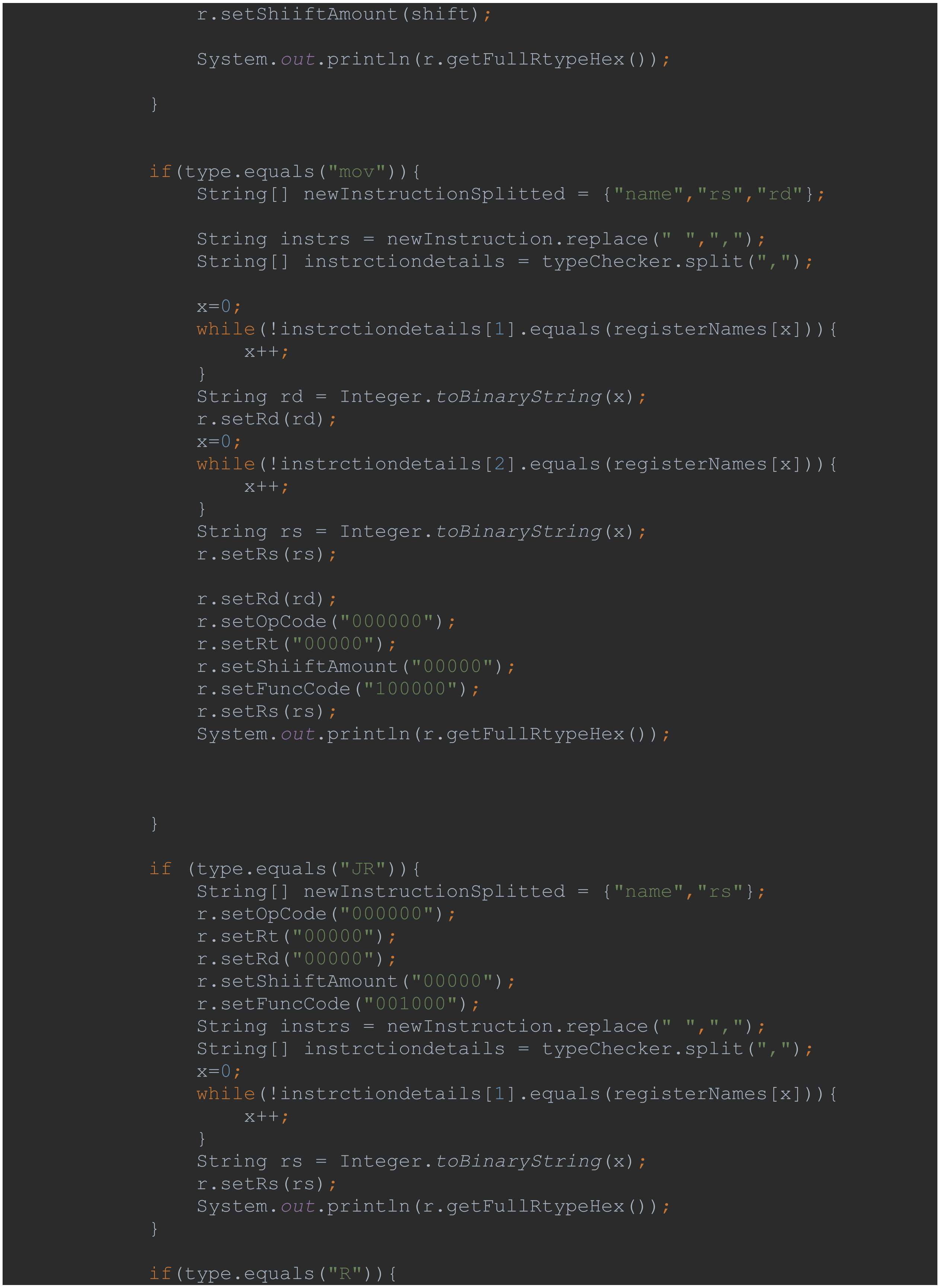
As we completed this project, we observed that designing an assembler can be a nerve-wracking task for novice engineers. Hence, it requires extreme level of patience, hardware knowledge and engineering skills. Moreover, we also learnt new classes and methods which are used for parsing thanks to this project. We believe we met expectations on this project. Our goal was to get the correct hexadecimal outputs that machine can recognize. We ran multiple test cases on *EWO Assembler* and we observed that we achieved our goal.

To conclude we can call this project as “ our best coding work in METU “ because we have used hundreds of method and functionalitys together with a great error handling. We believe we have deserved very high and good grade, and we are so thankful to our Instructor and TA’s for this wonderful project to work on and for everything.

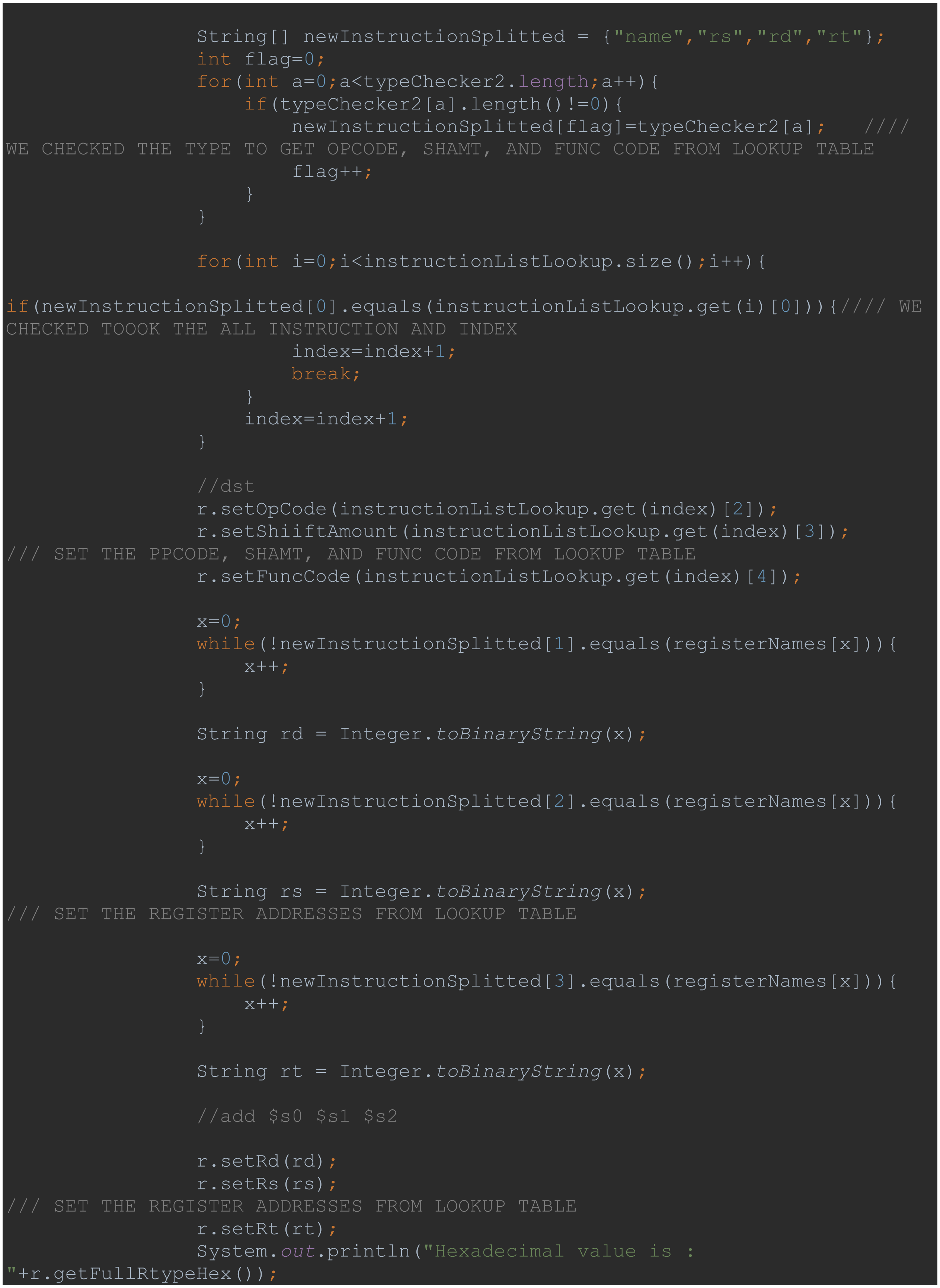
Soft copy of code

```
package com.omercelalCng331;

import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

public class Instruction {


    int InstructionType; //1  R-type, 2  I-type, 3 J-type
    String startingAddress="10000000000000000001000000000000";   // initial
address

    public Instruction() throws IOException {
    }


    public static String binaryToHex(String binaryNumber){   ////
        return (String.format("%35X", Long.parseLong(binaryNumber,2)));
    }


    public int getInstructionType() {
        return InstructionType;
    }

    public void setInstructionType(int instructionType) {
        InstructionType = instructionType;
    }

    public int Menu() throws Exception{

        int selection = 0;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Welcome to MIPS Assember Project !");
        System.out.println("1.Interactive Mode.");
        System.out.println("2.Batch Mode");
        System.out.println("3.Exit");
        System.out.println("Please Enter your choice: ");
        try {
            selection = scanner.nextInt();
        }catch(Exception error) {  // exception of error handled if the user 
enters wrong input
            System.out.println("Make a Valid Choice !!!");
            selection = Menu();
        }
        return selection;
    }



    public static void main(String[] args) throws Exception {

        Instruction assembler = new Instruction();
        File LookUpFile = new File("LookupTable.txt");

        String[] registerNames={};  /// im going to read the whole line 
seperately with this ,
        String[] instructionsLookup ={};
        String line;
        ArrayList<String[]> instructionListLookup = new ArrayList<>();  // 
and store these seperated datas in my array list
        ArrayList<String> instructionListInputSrc = new ArrayList<>();  // 
and store these seperated datas in my array list
        FileReader fileReaderLookup = new FileReader(LookUpFile);
        BufferedReader bufferedReaderLookUp = new 
BufferedReader(fileReaderLookup);

        String text = bufferedReaderLookUp.readLine();


        registerNames =text.split(",");
        /*
        for(int i =0;i<registerNames.length;i++){
            System.out.println(registerNames[i]);
        }

        String deneme = "$t5";

        int x=0;
        while(!deneme.equals(registerNames[x])){
            x++;
        }

        System.out.println(x);*/

        while((line=bufferedReaderLookUp.readLine()) != null){
            instructionsLookup=line.split(",");
            instructionListLookup.add(instructionsLookup);
        }
        bufferedReaderLookUp.close();

        int choice;
        choice =assembler.Menu();


///////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////
```

```
///
        if (choice==1){

            System.out.println("Enter an instruction ! :");
            Scanner scanner = new Scanner(System.in);
            String newInstruction=scanner.nextLine();
            String typeChecker = newInstruction.replace(" ",",");
            String[] typeChecker2 = typeChecker.split(",");

            Rtype r = new Rtype();
            Itype itype = new Itype();
            Jtype jtype = new Jtype();
            int index=-1;
            String type = null;
            if(typeChecker2[0].equals("add") || typeChecker2[0].equals("slt")
|| typeChecker2[0].equals("jr"))
                type="R";
            if(typeChecker2[0].equals("addi") || typeChecker2[0].equals("sw")
|| typeChecker2[0].equals("beq") || typeChecker2[0].equals("bne") ||
typeChecker2[0].equals("lw"))
                type="I";
            if(typeChecker2[0].equals("jal") || typeChecker2[0].equals("j"))
                type="J";
            if(typeChecker2[0].equals("jr"))
                type="JR";
            if(typeChecker2[0].equals("move"))
                type="mov";
            if(typeChecker2[0].equals("sll"))
                type="sll";
            if( typeChecker2[0].equals("slti"))
                type ="slti";


            //          sll $t1,$s1,2



/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///
            int x;


            if(type.equals("slti")){
                String[] newInstructionSplitted3 = {"name","rs","rt","im"};
                int flag=0;
                for(int a=0;a<typeChecker2.length;a++){
                    if(typeChecker2[a].length()!=0){
                        newInstructionSplitted3[flag]=typeChecker2[a];
                        flag++;
                    }
                }
                for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){ ///
WE CHECK THE LOOKUPTABLE TO IDENTIFY REGISTER ADDRESSES
                        index=index+1;       //// AND KEEP TRACK THE INDEX OF
ITTT
                        break;
                    }
                    index=index+1;
                }

                x=0;
                while(!newInstructionSplitted3[1].equals(registerNames[x])){
//// AND HERE WE ASSIGNED THEM BY CHECKING
                    x++;
                }
                String rs = Integer.toBinaryString(x);

                x=0;
                while(!newInstructionSplitted3[2].equals(registerNames[x])){
//// AND HERE WE ASSIGNED THEM BY CHECKING
                    x++;
                }
                String rt = Integer.toBinaryString(x);

                int q=Integer.parseInt(newInstructionSplitted3[3]);

                String im = Integer.toBinaryString(q);
                itype.setOpCode(instructionListLookup.get(index)[2]);
                itype.setImmediate(im);
                itype.setRs(rt);
                itype.setRt(rs);
                System.out.println(itype.getFullItypeHex());

            }


            if(type.equals("sll")){
                String[] newInstructionSplittedsll =
{"name","rd","rt","shiftamount"};
                r.setOpCode("000000");
                r.setRs("00000");
                r.setFuncCode("000000");
                String instrs = newInstruction.replace(" ",",");
                String[] instrctiondetails = typeChecker.split(",");
                x=0;
                while(!instrctiondetails[1].equals(registerNames[x])){
                    x++;
                }
                String rd = Integer.toBinaryString(x);
                r.setRd(rd);

                x=0;
                while(!instrctiondetails[2].equals(registerNames[x])){
                    x++;
                }
                String rt = Integer.toBinaryString(x);
                r.setRt(rt);
                System.out.println(r.getFullRtypeHex());

                int shamount = Integer.parseInt(instrctiondetails[3]);
                String shift = Integer.toBinaryString(shamount);
```

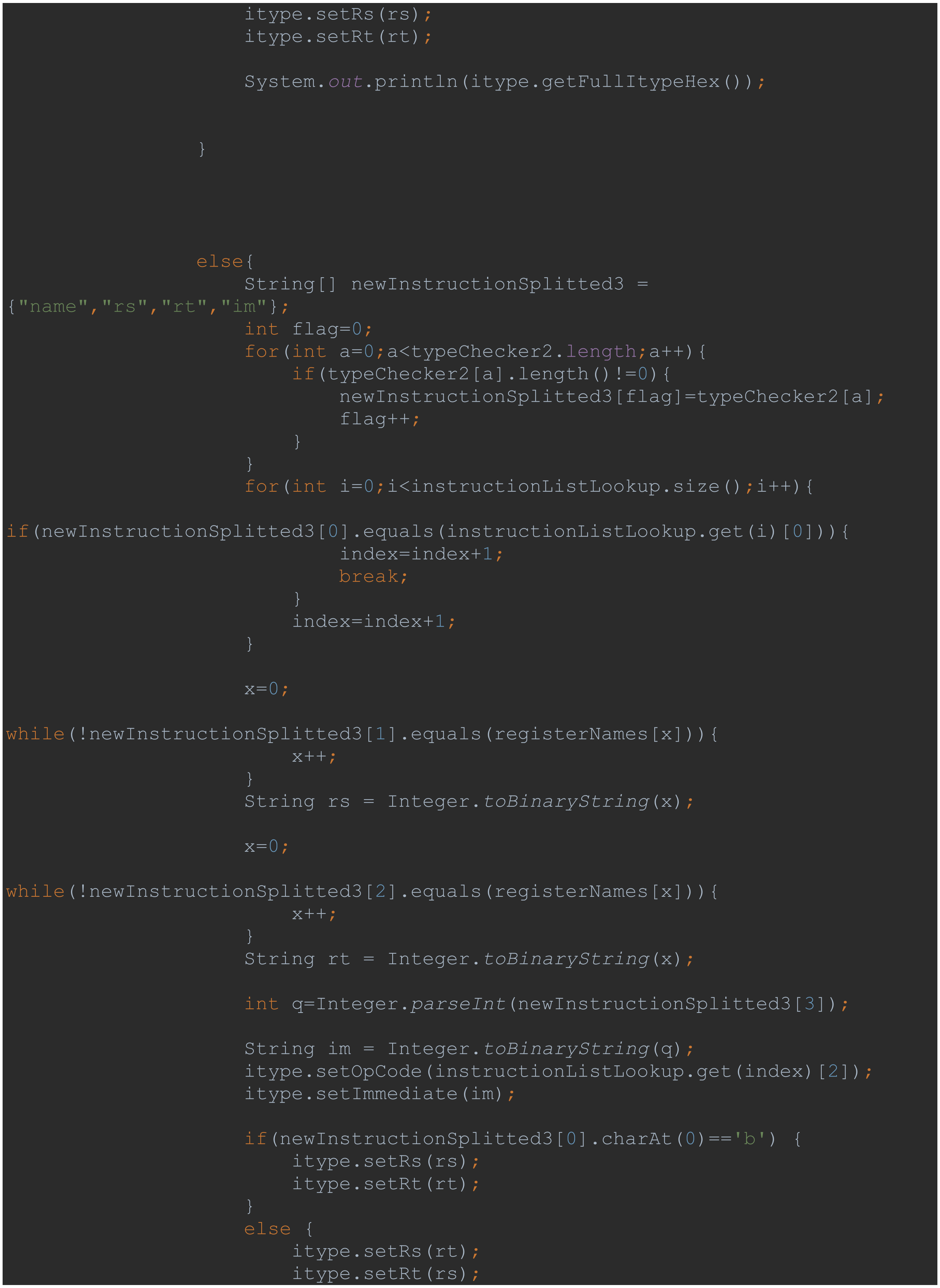


```
            r.setShiiftAmount(shift);

            System.out.println(r.getFullRtypeHex());

        }


        if(type.equals("mov")){
            String[] newInstructionSplitted = {"name","rs","rd"};

            String instrs = newInstruction.replace(" ",",");
            String[] instrctiondetails = typeChecker.split(",");

            x=0;
            while(!instrctiondetails[1].equals(registerNames[x])){
                x++;
            }
            String rd = Integer.toBinaryString(x);
            r.setRd(rd);
            x=0;
            while(!instrctiondetails[2].equals(registerNames[x])){
                x++;
            }
            String rs = Integer.toBinaryString(x);
            r.setRs(rs);

            r.setRd(rd);
            r.setOpCode("000000");
            r.setRt("00000");
            r.setShiiftAmount("00000");
            r.setFuncCode("100000");
            r.setRs(rs);
            System.out.println(r.getFullRtypeHex());



        }

        if (type.equals("JR")){
            String[] newInstructionSplitted = {"name","rs"};
            r.setOpCode("000000");
            r.setRt("00000");
            r.setRd("00000");
            r.setShiiftAmount("00000");
            r.setFuncCode("001000");
            String instrs = newInstruction.replace(" ",",");
            String[] instrctiondetails = typeChecker.split(",");
            x=0;
            while(!instrctiondetails[1].equals(registerNames[x])){
                x++;
            }
            String rs = Integer.toBinaryString(x);
            r.setRs(rs);
            System.out.println(r.getFullRtypeHex());
        }

        if(type.equals("R")){
```

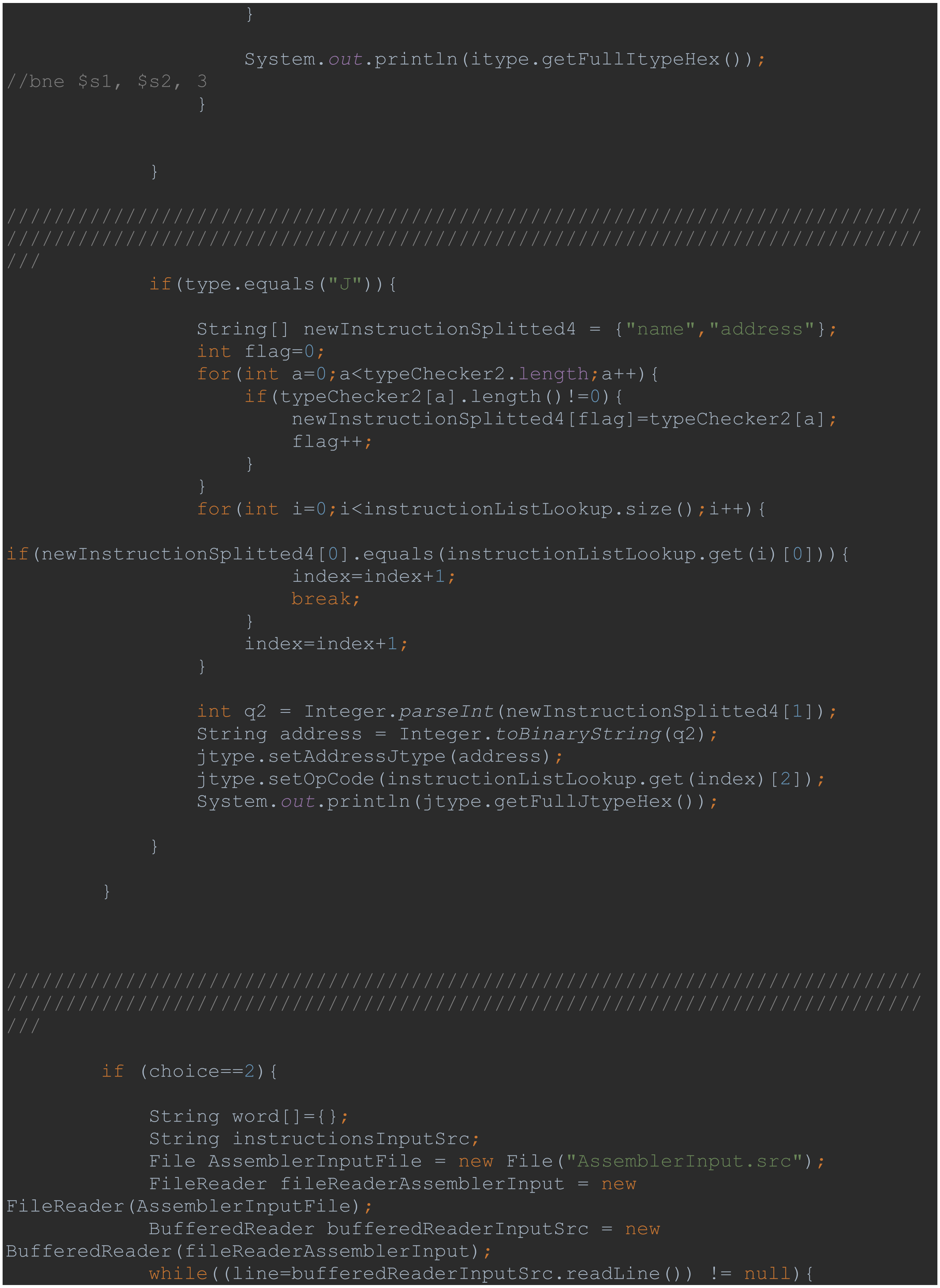


```
                String[] newInstructionSplitted = {"name","rs","rd","rt"};
                int flag=0;
                for(int a=0;a<typeChecker2.length;a++){
                    if(typeChecker2[a].length()!=0){
                        newInstructionSplitted[flag]=typeChecker2[a];    ////
WE CHECKED THE TYPE TO GET OPCODE, SHAMT, AND FUNC CODE FROM LOOKUP TABLE
                        flag++;
                    }
                }

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

if(newInstructionSplitted[0].equals(instructionListLookup.get(i)[0])){//// WE
CHECKED TOOOK THE ALL INSTRUCTION AND INDEX
                        index=index+1;
                        break;
                    }
                    index=index+1;
                }

                //dst
                r.setOpCode(instructionListLookup.get(index)[2]);
                r.setShiiftAmount(instructionListLookup.get(index)[3]);
/// SET THE PPCODE, SHAMT, AND FUNC CODE FROM LOOKUP TABLE
                r.setFuncCode(instructionListLookup.get(index)[4]);

                x=0;
                while(!newInstructionSplitted[1].equals(registerNames[x])){
                    x++;
                }

                String rd = Integer.toBinaryString(x);

                x=0;
                while(!newInstructionSplitted[2].equals(registerNames[x])){
                    x++;
                }

                String rs = Integer.toBinaryString(x);
/// SET THE REGISTER ADDRESSES FROM LOOKUP TABLE

                x=0;
                while(!newInstructionSplitted[3].equals(registerNames[x])){
                    x++;
                }

                String rt = Integer.toBinaryString(x);

                //add $s0 $s1 $s2

                r.setRd(rd);
                r.setRs(rs);
/// SET THE REGISTER ADDRESSES FROM LOOKUP TABLE
                r.setRt(rt);
                System.out.println("Hexadecimal value is :
"+r.getFullRtypeHex());
```

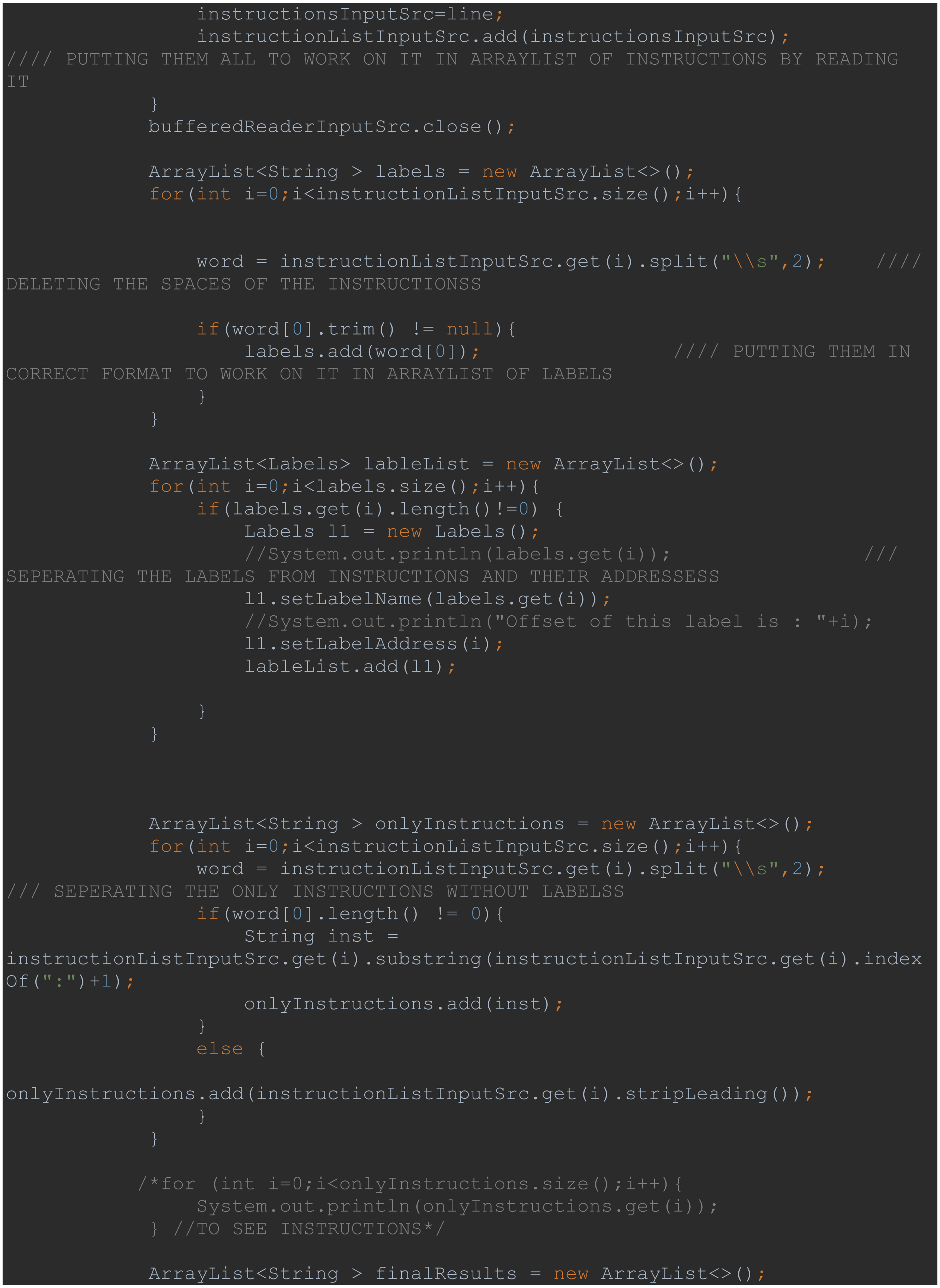
```
            }

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///
            if(type.equals("I")){


if(typeChecker2[0].equals("lw")||typeChecker2[0].equals("sw")){
                    String[] newInstructionSplitted2 =
{"name","rt","im(rs)"};              /// DIIFERENT IMPLEMENTATION FOR SW AND
LW
                    int flag=0;
                    for(int a=0;a<typeChecker2.length;a++){
                        if(typeChecker2[a].length()!=0){
                            newInstructionSplitted2[flag]=typeChecker2[a];
                            flag++;
                        }
                    }
                    for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted2[0].equals(instructionListLookup.get(i)[0])){
                            index=index+1;
                            break;
                        }
                        index=index+1;
                    }



                    String str = newInstructionSplitted2[2];
                    String rsa =
str.substring(str.indexOf("(")+1,str.indexOf(")"));
                    String immidiate = str.substring(0,str.indexOf("("));

                    int q=Integer.parseInt(immidiate);

                    String im = Integer.toBinaryString(q);


                    x=0;
                    while(!rsa.equals(registerNames[x])){
                        x++;
                    }
//lw $t0, 32($s3)
                    String rs = Integer.toBinaryString(x);

                    x=0;

while(!newInstructionSplitted2[1].equals(registerNames[x])){
                        x++;
                    }
                    String rt = Integer.toBinaryString(x);


                    itype.setOpCode(instructionListLookup.get(index)[2]);
                    itype.setImmediate(im);
```



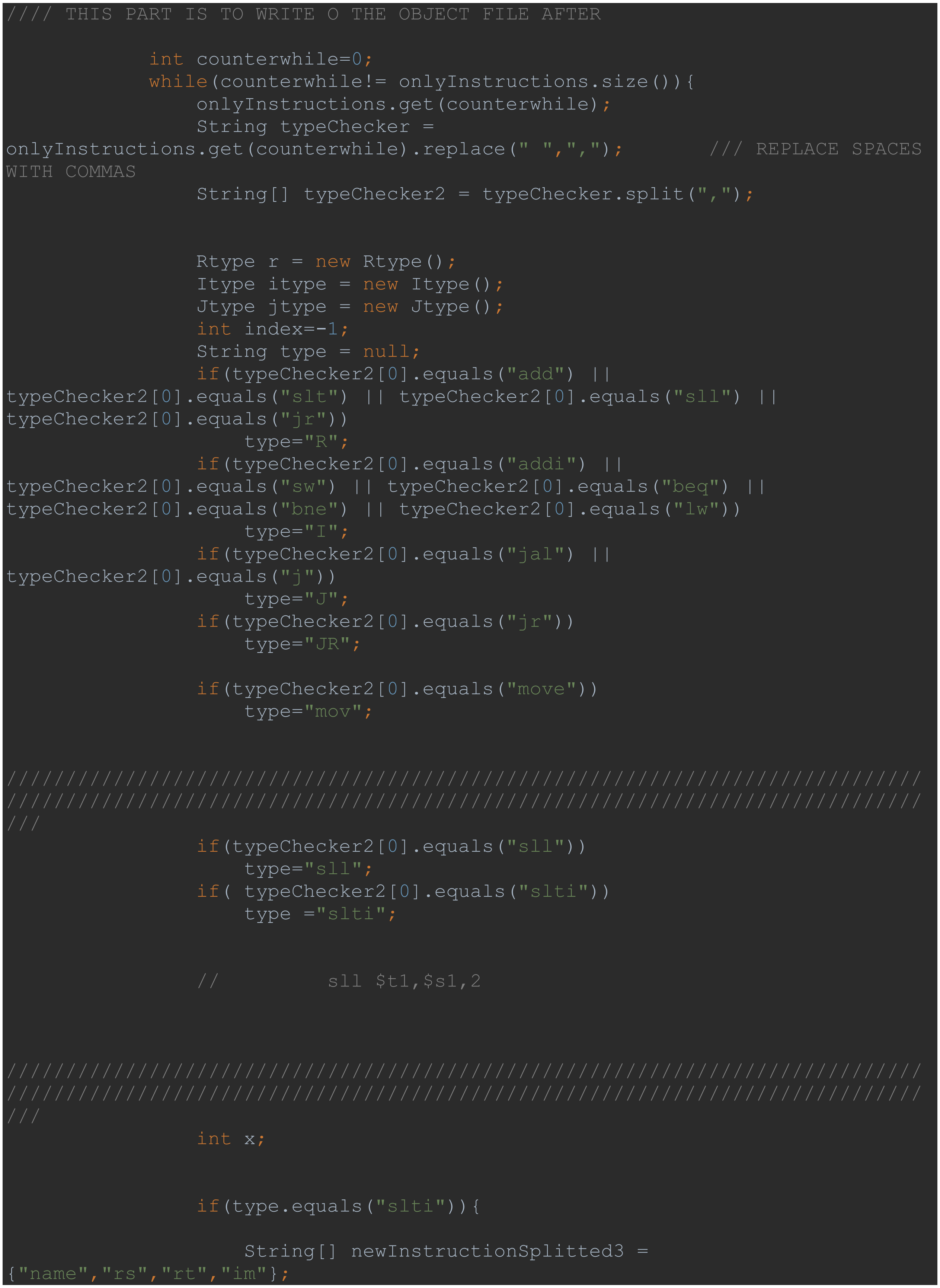
```
                    itype.setRs(rs);
                    itype.setRt(rt);

                    System.out.println(itype.getFullItypeHex());


                }




                else{
                    String[] newInstructionSplitted3 =
{"name","rs","rt","im"};
                    int flag=0;
                    for(int a=0;a<typeChecker2.length;a++){
                        if(typeChecker2[a].length()!=0){
                            newInstructionSplitted3[flag]=typeChecker2[a];
                            flag++;
                        }
                    }
                    for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){
                            index=index+1;
                            break;
                        }
                        index=index+1;
                    }

                    x=0;

while(!newInstructionSplitted3[1].equals(registerNames[x])){
                        x++;
                    }
                    String rs = Integer.toBinaryString(x);

                    x=0;

while(!newInstructionSplitted3[2].equals(registerNames[x])){
                        x++;
                    }
                    String rt = Integer.toBinaryString(x);

                    int q=Integer.parseInt(newInstructionSplitted3[3]);

                    String im = Integer.toBinaryString(q);
                    itype.setOpCode(instructionListLookup.get(index)[2]);
                    itype.setImmediate(im);

                    if(newInstructionSplitted3[0].charAt(0)=='b') {
                        itype.setRs(rs);
                        itype.setRt(rt);
                    }
                    else {
                        itype.setRs(rt);
                        itype.setRt(rs);
```



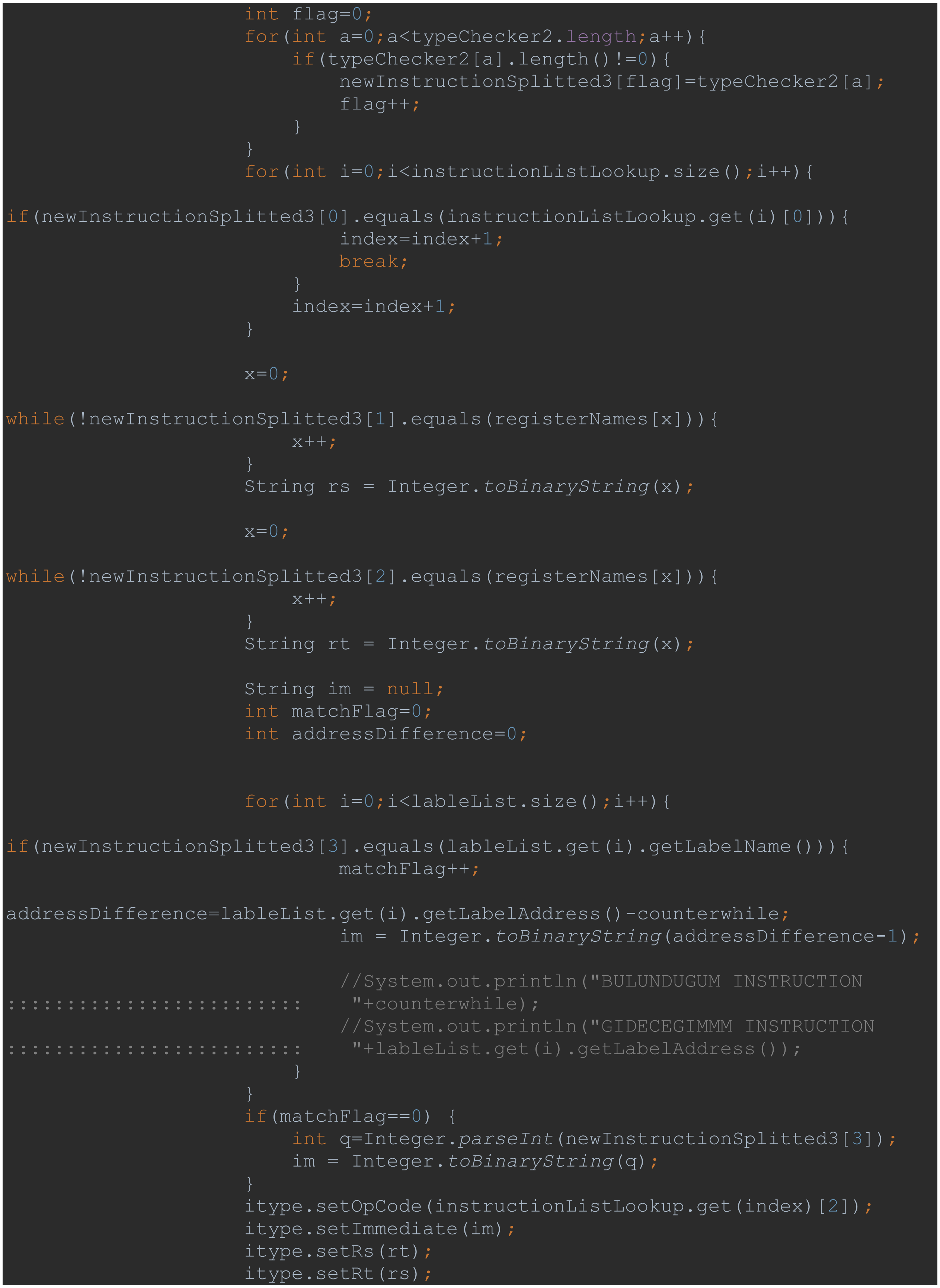
```
                    }

                    System.out.println(itype.getFullItypeHex());
//bne $s1, $s2, 3
                }


            }

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///
            if(type.equals("J")){

                String[] newInstructionSplitted4 = {"name","address"};
                int flag=0;
                for(int a=0;a<typeChecker2.length;a++){
                    if(typeChecker2[a].length()!=0){
                        newInstructionSplitted4[flag]=typeChecker2[a];
                        flag++;
                    }
                }
                for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted4[0].equals(instructionListLookup.get(i)[0])){
                        index=index+1;
                        break;
                    }
                    index=index+1;
                }

                int q2 = Integer.parseInt(newInstructionSplitted4[1]);
                String address = Integer.toBinaryString(q2);
                jtype.setAddressJtype(address);
                jtype.setOpCode(instructionListLookup.get(index)[2]);
                System.out.println(jtype.getFullJtypeHex());

            }

        }




/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///

        if (choice==2){

            String word[]={};
            String instructionsInputSrc;
            File AssemblerInputFile = new File("AssemblerInput.src");
            FileReader fileReaderAssemblerInput = new
FileReader(AssemblerInputFile);
            BufferedReader bufferedReaderInputSrc = new
BufferedReader(fileReaderAssemblerInput);
            while((line=bufferedReaderInputSrc.readLine()) != null){
```



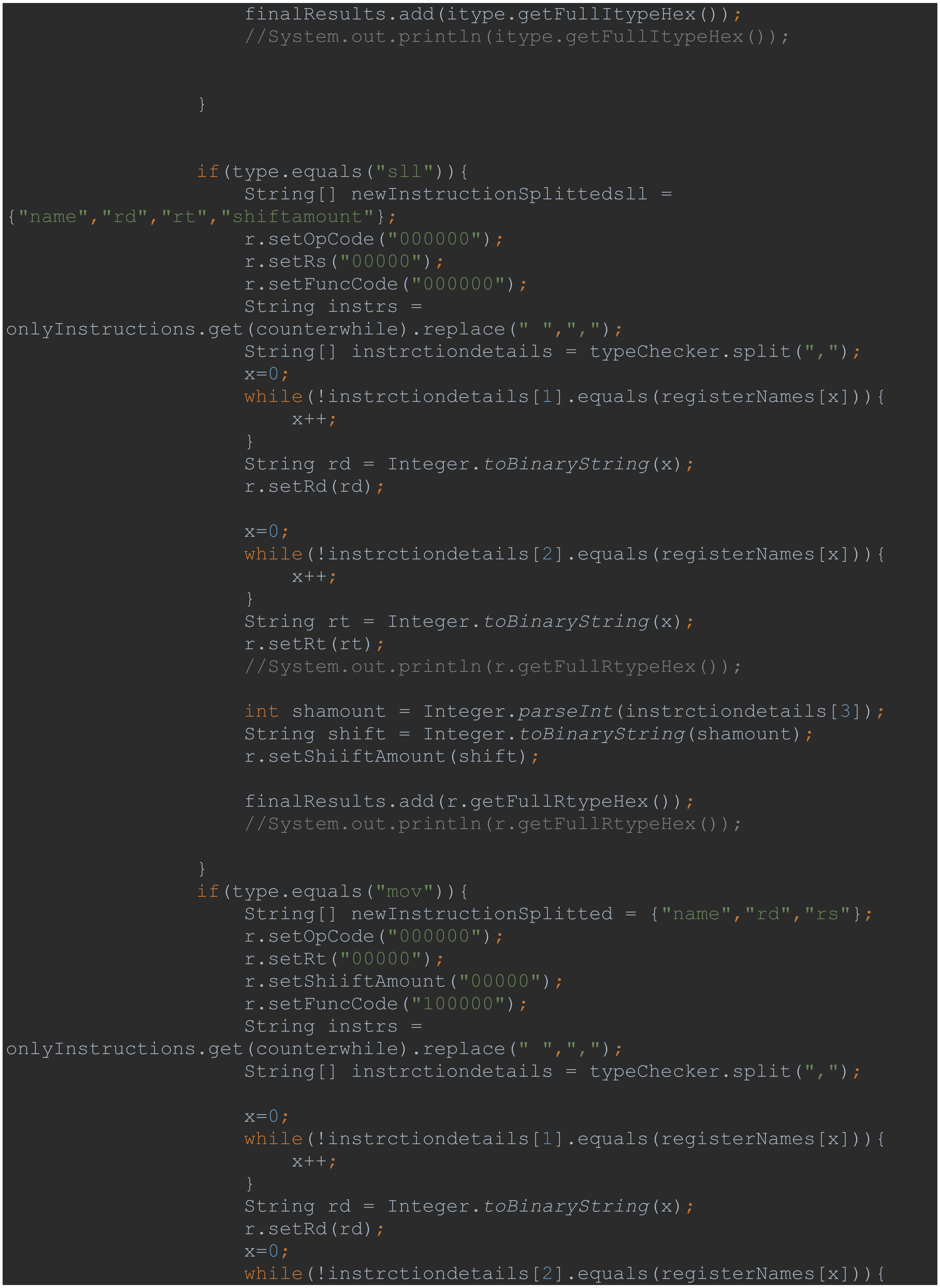
```
                instructionsInputSrc=line;
                instructionListInputSrc.add(instructionsInputSrc);
//// PUTTING THEM ALL TO WORK ON IT IN ARRAYLIST OF INSTRUCTIONS BY READING
IT
            }
            bufferedReaderInputSrc.close();

            ArrayList<String > labels = new ArrayList<>();
            for(int i=0;i<instructionListInputSrc.size();i++){


                word = instructionListInputSrc.get(i).split("\\s",2);    ////
DELETING THE SPACES OF THE INSTRUCTIONSS

                if(word[0].trim() != null){
                    labels.add(word[0]);                //// PUTTING THEM IN
CORRECT FORMAT TO WORK ON IT IN ARRAYLIST OF LABELS
                }
            }

            ArrayList<Labels> lableList = new ArrayList<>();
            for(int i=0;i<labels.size();i++){
                if(labels.get(i).length()!=0) {
                    Labels l1 = new Labels();
                    //System.out.println(labels.get(i));                ///
SEPERATING THE LABELS FROM INSTRUCTIONS AND THEIR ADDRESSESS
                    l1.setLabelName(labels.get(i));
                    //System.out.println("Offset of this label is : "+i);
                    l1.setLabelAddress(i);
                    lableList.add(l1);

                }
            }



            ArrayList<String > onlyInstructions = new ArrayList<>();
            for(int i=0;i<instructionListInputSrc.size();i++){
                word = instructionListInputSrc.get(i).split("\\s",2);
/// SEPERATING THE ONLY INSTRUCTIONS WITHOUT LABELSS
                if(word[0].length() != 0){
                    String inst =
instructionListInputSrc.get(i).substring(instructionListInputSrc.get(i).index
Of(":")+1);
                    onlyInstructions.add(inst);
                }
                else {

onlyInstructions.add(instructionListInputSrc.get(i).stripLeading());
                }
            }

           /*for (int i=0;i<onlyInstructions.size();i++){
                System.out.println(onlyInstructions.get(i));
            } //TO SEE INSTRUCTIONS*/

            ArrayList<String > finalResults = new ArrayList<>();
```



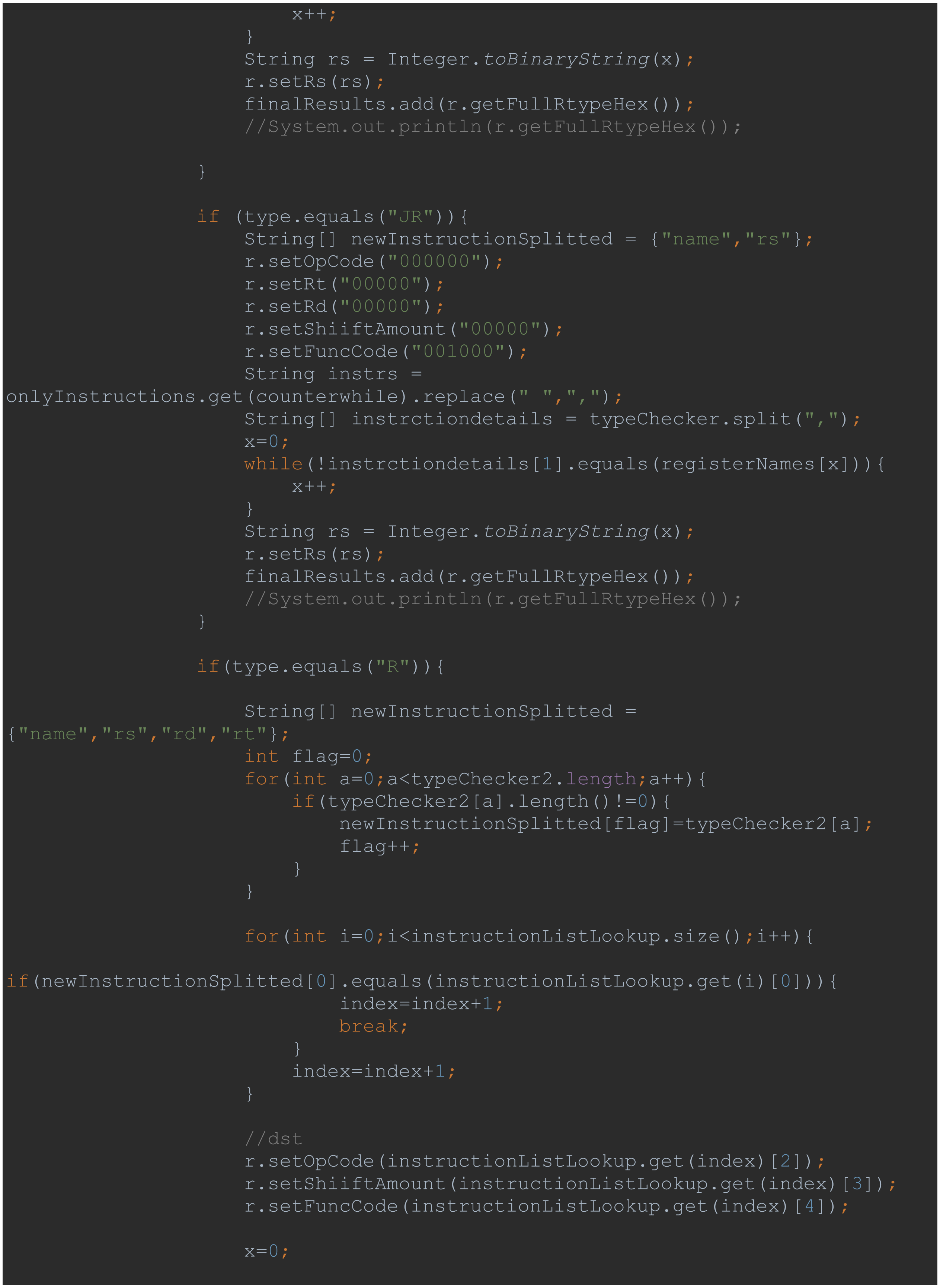
```
//// THIS PART IS TO WRITE O THE OBJECT FILE AFTER

            int counterwhile=0;
            while(counterwhile!= onlyInstructions.size()){
                onlyInstructions.get(counterwhile);
                String typeChecker =
onlyInstructions.get(counterwhile).replace(" ",",");       /// REPLACE SPACES
WITH COMMAS
                String[] typeChecker2 = typeChecker.split(",");


                Rtype r = new Rtype();
                Itype itype = new Itype();
                Jtype jtype = new Jtype();
                int index=-1;
                String type = null;
                if(typeChecker2[0].equals("add") ||
typeChecker2[0].equals("slt") || typeChecker2[0].equals("sll") ||
typeChecker2[0].equals("jr"))
                    type="R";
                if(typeChecker2[0].equals("addi") ||
typeChecker2[0].equals("sw") || typeChecker2[0].equals("beq") ||
typeChecker2[0].equals("bne") || typeChecker2[0].equals("lw"))
                    type="I";
                if(typeChecker2[0].equals("jal") ||
typeChecker2[0].equals("j"))
                    type="J";
                if(typeChecker2[0].equals("jr"))
                    type="JR";

                if(typeChecker2[0].equals("move"))
                    type="mov";


/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///
                if(typeChecker2[0].equals("sll"))
                    type="sll";
                if( typeChecker2[0].equals("slti"))
                    type ="slti";


                //         sll $t1,$s1,2




/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
///
                int x;


                if(type.equals("slti")){

                    String[] newInstructionSplitted3 =
{"name","rs","rt","im"};
```



```
                    int flag=0;
                    for(int a=0;a<typeChecker2.length;a++){
                        if(typeChecker2[a].length()!=0){
                            newInstructionSplitted3[flag]=typeChecker2[a];
                            flag++;
                        }
                    }
                    for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){
                            index=index+1;
                            break;
                        }
                        index=index+1;
                    }

                    x=0;

while(!newInstructionSplitted3[1].equals(registerNames[x])){
                        x++;
                    }
                    String rs = Integer.toBinaryString(x);

                    x=0;

while(!newInstructionSplitted3[2].equals(registerNames[x])){
                        x++;
                    }
                    String rt = Integer.toBinaryString(x);

                    String im = null;
                    int matchFlag=0;
                    int addressDifference=0;


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

if(newInstructionSplitted3[3].equals(lableList.get(i).getLabelName())){
                            matchFlag++;

addressDifference=lableList.get(i).getLabelAddress()-counterwhile;
                            im = Integer.toBinaryString(addressDifference-1);

                            //System.out.println("BULUNDUGUM INSTRUCTION
:::::::::::::::::::::::::    "+counterwhile);
                            //System.out.println("GIDECEGIMMM INSTRUCTION
:::::::::::::::::::::::::    "+lableList.get(i).getLabelAddress());
                        }
                    }
                    if(matchFlag==0) {
                        int q=Integer.parseInt(newInstructionSplitted3[3]);
                        im = Integer.toBinaryString(q);
                    }
                    itype.setOpCode(instructionListLookup.get(index)[2]);
                    itype.setImmediate(im);
                    itype.setRs(rt);
                    itype.setRt(rs);
```

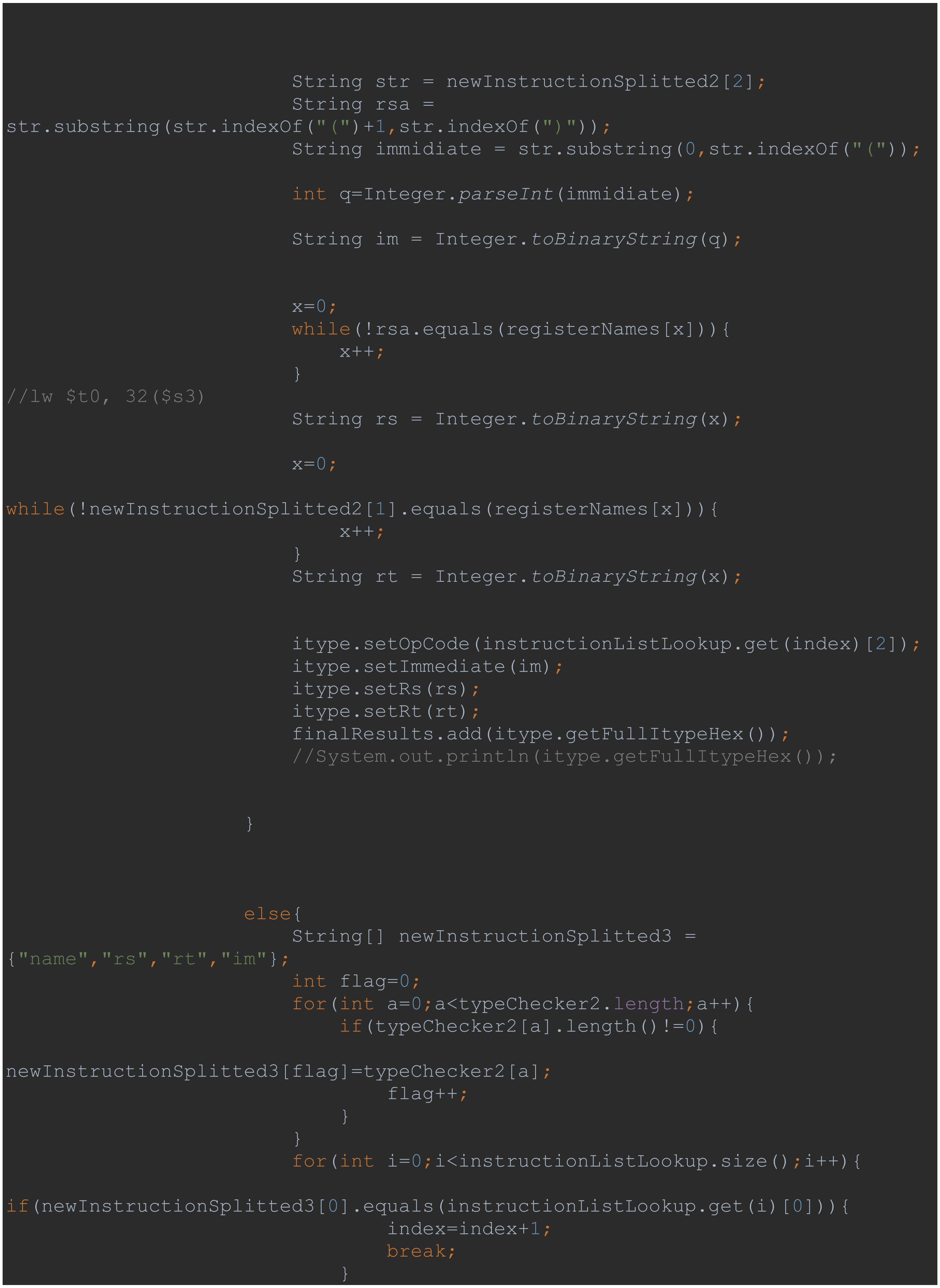


```
                    finalResults.add(itype.getFullItypeHex());
                    //System.out.println(itype.getFullItypeHex());


                }


                if(type.equals("sll")){
                    String[] newInstructionSplittedsll =
{"name","rd","rt","shiftamount"};
                    r.setOpCode("000000");
                    r.setRs("00000");
                    r.setFuncCode("000000");
                    String instrs =
onlyInstructions.get(counterwhile).replace(" ",",");
                    String[] instrctiondetails = typeChecker.split(",");
                    x=0;
                    while(!instrctiondetails[1].equals(registerNames[x])){
                        x++;
                    }
                    String rd = Integer.toBinaryString(x);
                    r.setRd(rd);

                    x=0;
                    while(!instrctiondetails[2].equals(registerNames[x])){
                        x++;
                    }
                    String rt = Integer.toBinaryString(x);
                    r.setRt(rt);
                    //System.out.println(r.getFullRtypeHex());

                    int shamount = Integer.parseInt(instrctiondetails[3]);
                    String shift = Integer.toBinaryString(shamount);
                    r.setShiiftAmount(shift);

                    finalResults.add(r.getFullRtypeHex());
                    //System.out.println(r.getFullRtypeHex());

                }
                if(type.equals("mov")){
                    String[] newInstructionSplitted = {"name","rd","rs"};
                    r.setOpCode("000000");
                    r.setRt("00000");
                    r.setShiiftAmount("00000");
                    r.setFuncCode("100000");
                    String instrs =
onlyInstructions.get(counterwhile).replace(" ",",");
                    String[] instrctiondetails = typeChecker.split(",");

                    x=0;
                    while(!instrctiondetails[1].equals(registerNames[x])){
                        x++;
                    }
                    String rd = Integer.toBinaryString(x);
                    r.setRd(rd);
                    x=0;
                    while(!instrctiondetails[2].equals(registerNames[x])){
```

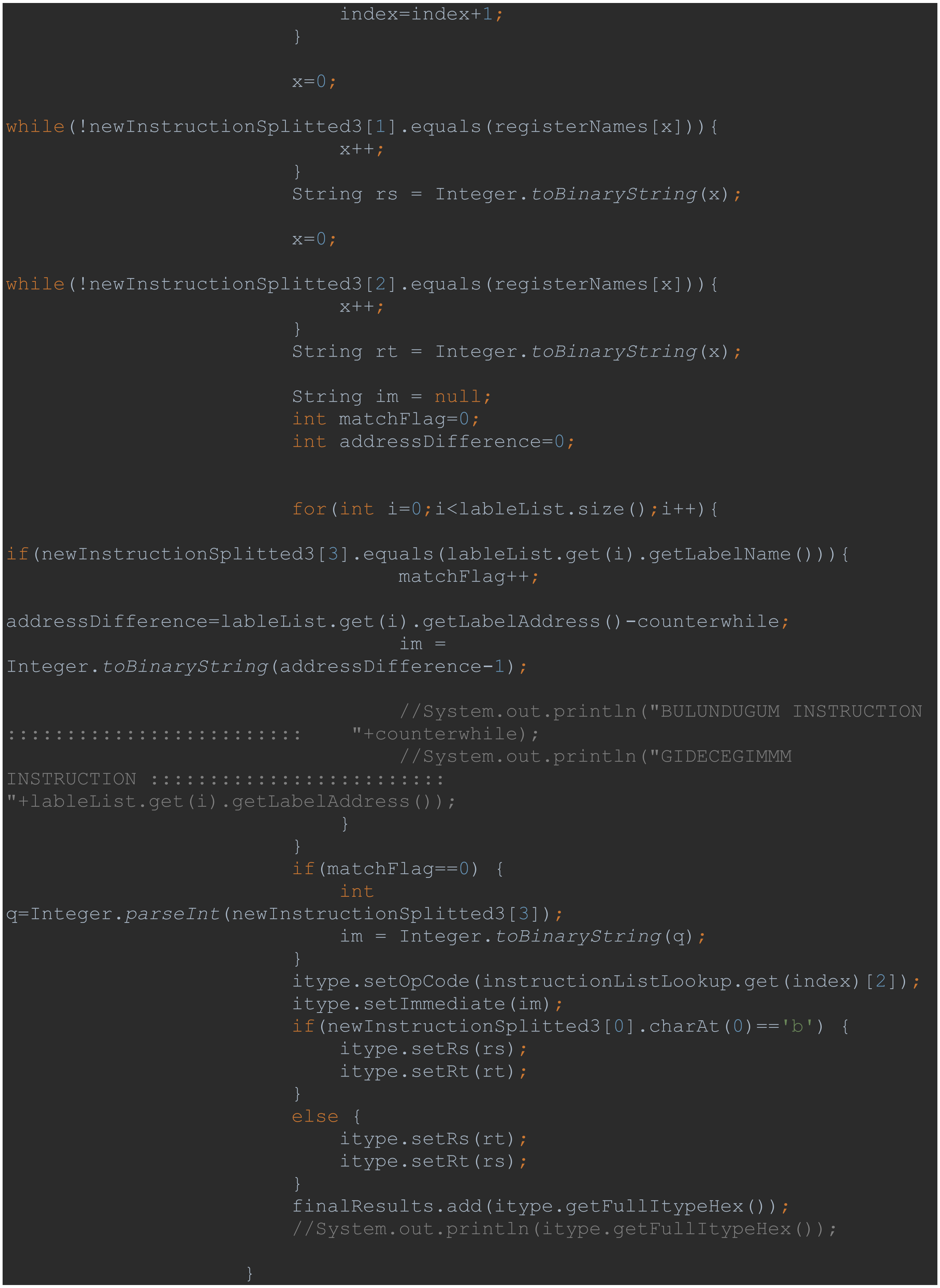


```
                        x++;
                    }
                    String rs = Integer.toBinaryString(x);
                    r.setRs(rs);
                    finalResults.add(r.getFullRtypeHex());
                    //System.out.println(r.getFullRtypeHex());

                }

                if (type.equals("JR")){
                    String[] newInstructionSplitted = {"name","rs"};
                    r.setOpCode("000000");
                    r.setRt("00000");
                    r.setRd("00000");
                    r.setShiiftAmount("00000");
                    r.setFuncCode("001000");
                    String instrs =
onlyInstructions.get(counterwhile).replace(" ",",");
                    String[] instrctiondetails = typeChecker.split(",");
                    x=0;
                    while(!instrctiondetails[1].equals(registerNames[x])){
                        x++;
                    }
                    String rs = Integer.toBinaryString(x);
                    r.setRs(rs);
                    finalResults.add(r.getFullRtypeHex());
                    //System.out.println(r.getFullRtypeHex());
                }

                if(type.equals("R")){

                    String[] newInstructionSplitted =
{"name","rs","rd","rt"};
                    int flag=0;
                    for(int a=0;a<typeChecker2.length;a++){
                        if(typeChecker2[a].length()!=0){
                            newInstructionSplitted[flag]=typeChecker2[a];
                            flag++;
                        }
                    }

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

if(newInstructionSplitted[0].equals(instructionListLookup.get(i)[0])){
                            index=index+1;
                            break;
                        }
                        index=index+1;
                    }

                    //dst
                    r.setOpCode(instructionListLookup.get(index)[2]);
                    r.setShiiftAmount(instructionListLookup.get(index)[3]);
                    r.setFuncCode(instructionListLookup.get(index)[4]);

                    x=0;
```

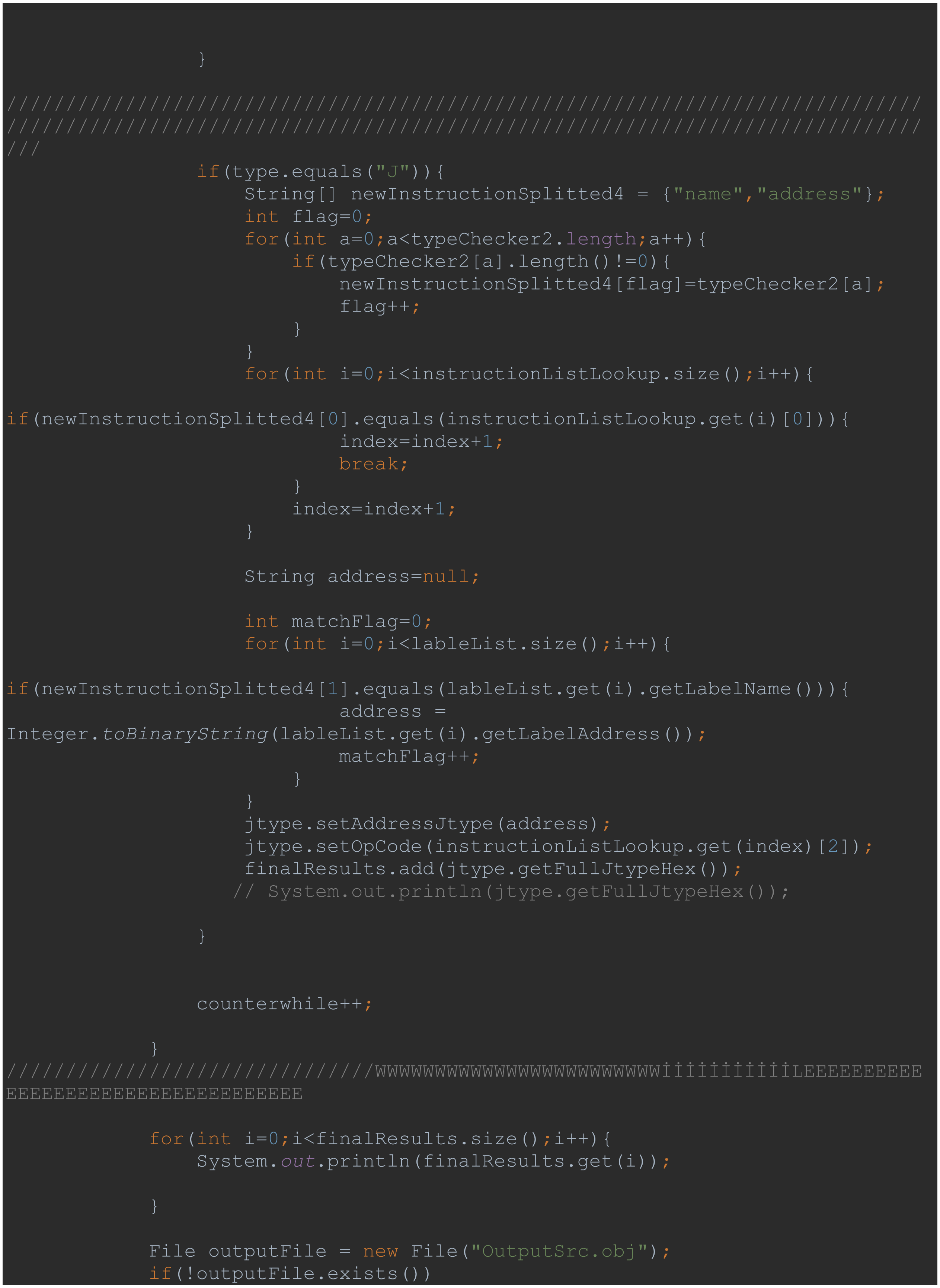
```
while(!newInstructionSplitted[1].equals(registerNames[x])){
                        x++;
                    }

                    String rd = Integer.toBinaryString(x);

                    x=0;

while(!newInstructionSplitted[2].equals(registerNames[x])){
                        x++;
                    }

                    String rs = Integer.toBinaryString(x);

                    x=0;

while(!newInstructionSplitted[3].equals(registerNames[x])){
                        x++;
                    }

                    String rt = Integer.toBinaryString(x);

                    //add $s0 $s1 $s2

                    r.setRd(rd);
                    r.setRs(rs);
                    r.setRt(rt);
                    finalResults.add(r.getFullRtypeHex());
                    //System.out.println("Hexadecimal value is :
"+r.getFullRtypeHex());
                }

//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
///
                if(type.equals("I")){
                    //addi $s1, $s2, 3

if(typeChecker2[0].equals("lw")||typeChecker2[0].equals("sw")){
                        String[] newInstructionSplitted2 =
{"name","rt","im(rs)"};
                        int flag=0;
                        for(int a=0;a<typeChecker2.length;a++){
                            if(typeChecker2[a].length()!=0){

newInstructionSplitted2[flag]=typeChecker2[a];
                                flag++;
                            }
                        }
                        for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted2[0].equals(instructionListLookup.get(i)[0])){
                                index=index+1;
                                break;
                            }
                            index=index+1;
                        }
```



```
                        String str = newInstructionSplitted2[2];
                        String rsa =
str.substring(str.indexOf("(")+1,str.indexOf(")"));
                        String immidiate = str.substring(0,str.indexOf("("));

                        int q=Integer.parseInt(immidiate);

                        String im = Integer.toBinaryString(q);


                        x=0;
                        while(!rsa.equals(registerNames[x])){
                            x++;
                        }
//lw $t0, 32($s3)
                        String rs = Integer.toBinaryString(x);

                        x=0;

while(!newInstructionSplitted2[1].equals(registerNames[x])){
                            x++;
                        }
                        String rt = Integer.toBinaryString(x);


                        itype.setOpCode(instructionListLookup.get(index)[2]);
                        itype.setImmediate(im);
                        itype.setRs(rs);
                        itype.setRt(rt);
                        finalResults.add(itype.getFullItypeHex());
                        //System.out.println(itype.getFullItypeHex());


                    }



                    else{
                        String[] newInstructionSplitted3 =
{"name","rs","rt","im"};
                        int flag=0;
                        for(int a=0;a<typeChecker2.length;a++){
                            if(typeChecker2[a].length()!=0){

newInstructionSplitted3[flag]=typeChecker2[a];
                                flag++;
                            }
                        }
                        for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted3[0].equals(instructionListLookup.get(i)[0])){
                                index=index+1;
                                break;
                            }
```

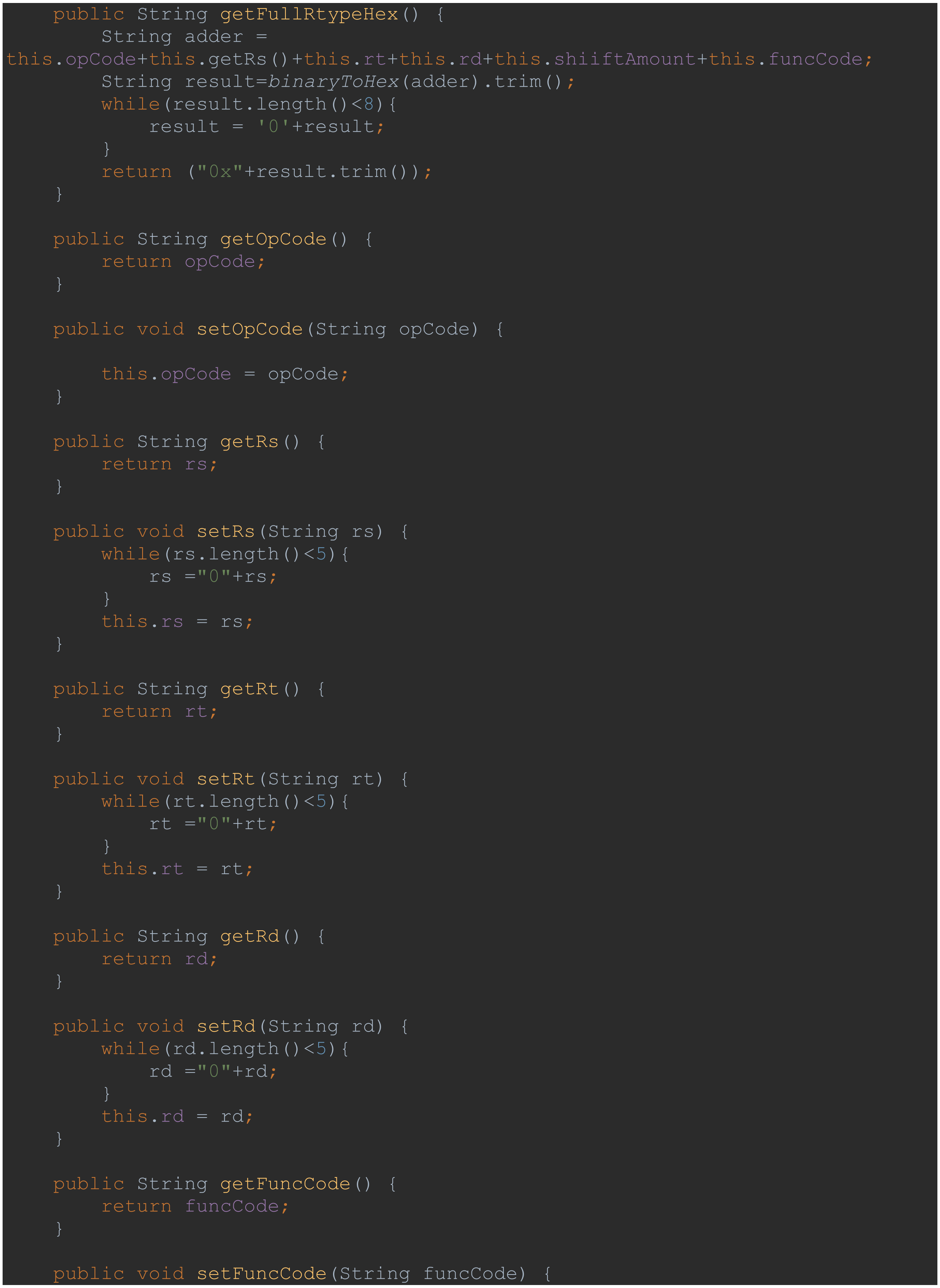


```
                            index=index+1;
                        }

                        x=0;

while(!newInstructionSplitted3[1].equals(registerNames[x])){
                            x++;
                        }
                        String rs = Integer.toBinaryString(x);

                        x=0;

while(!newInstructionSplitted3[2].equals(registerNames[x])){
                            x++;
                        }
                        String rt = Integer.toBinaryString(x);

                        String im = null;
                        int matchFlag=0;
                        int addressDifference=0;


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

if(newInstructionSplitted3[3].equals(lableList.get(i).getLabelName())){
                                 matchFlag++;

addressDifference=lableList.get(i).getLabelAddress()-counterwhile;
                                 im =
Integer.toBinaryString(addressDifference-1);

                                 //System.out.println("BULUNDUGUM INSTRUCTION
:::::::::::::::::::::::::    "+counterwhile);
                                 //System.out.println("GIDECEGIMMM
INSTRUCTION :::::::::::::::::::::::::
"+lableList.get(i).getLabelAddress());
                            }
                        }
                        if(matchFlag==0) {
                            int
q=Integer.parseInt(newInstructionSplitted3[3]);
                            im = Integer.toBinaryString(q);
                        }
                        itype.setOpCode(instructionListLookup.get(index)[2]);
                        itype.setImmediate(im);
                        if(newInstructionSplitted3[0].charAt(0)=='b') {
                            itype.setRs(rs);
                            itype.setRt(rt);
                        }
                        else {
                            itype.setRs(rt);
                            itype.setRt(rs);
                        }
                        finalResults.add(itype.getFullItypeHex());
                        //System.out.println(itype.getFullItypeHex());

                    }
```

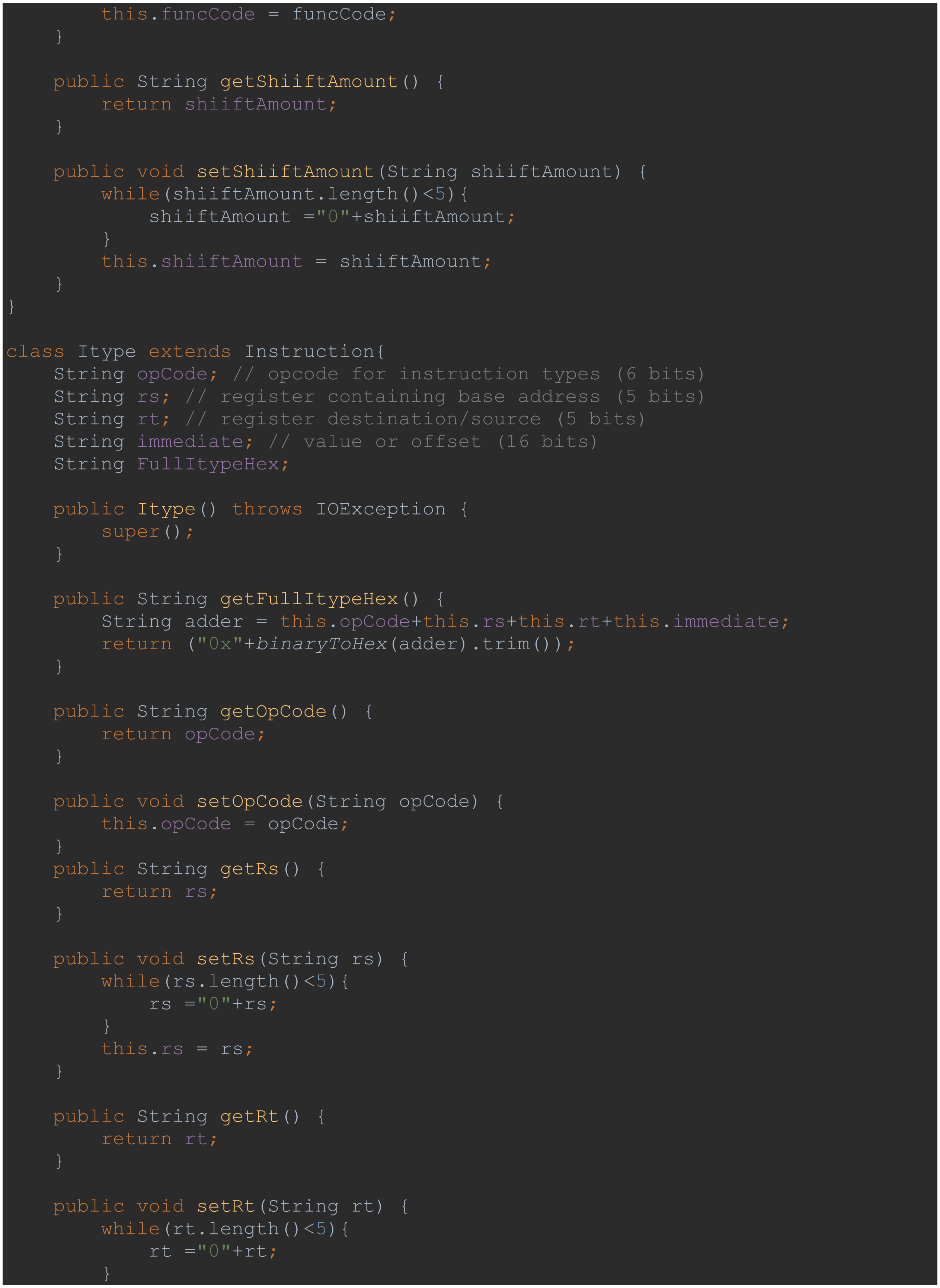


```
                }
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
///
                if(type.equals("J")){
                    String[] newInstructionSplitted4 = {"name","address"};
                    int flag=0;
                    for(int a=0;a<typeChecker2.length;a++){
                        if(typeChecker2[a].length()!=0){
                            newInstructionSplitted4[flag]=typeChecker2[a];
                            flag++;
                        }
                    }
                    for(int i=0;i<instructionListLookup.size();i++){

if(newInstructionSplitted4[0].equals(instructionListLookup.get(i)[0])){
                            index=index+1;
                            break;
                        }
                        index=index+1;
                    }

                    String address=null;

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

if(newInstructionSplitted4[1].equals(lableList.get(i).getLabelName())){
                            address =
Integer.toBinaryString(lableList.get(i).getLabelAddress());
                            matchFlag++;
                        }
                    }
                    jtype.setAddressJtype(address);
                    jtype.setOpCode(instructionListLookup.get(index)[2]);
                    finalResults.add(jtype.getFullJtypeHex());
                   // System.out.println(jtype.getFullJtypeHex());

                }


                counterwhile++;

            }
///////////////////////////////WWWWWWWWWWWWWWWWWWWWWWWWiiiiiiiiiiiLEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEE

            for(int i=0;i<finalResults.size();i++){
                System.out.println(finalResults.get(i));

            }

            File outputFile = new File("OutputSrc.obj");
            if(!outputFile.exists())
```

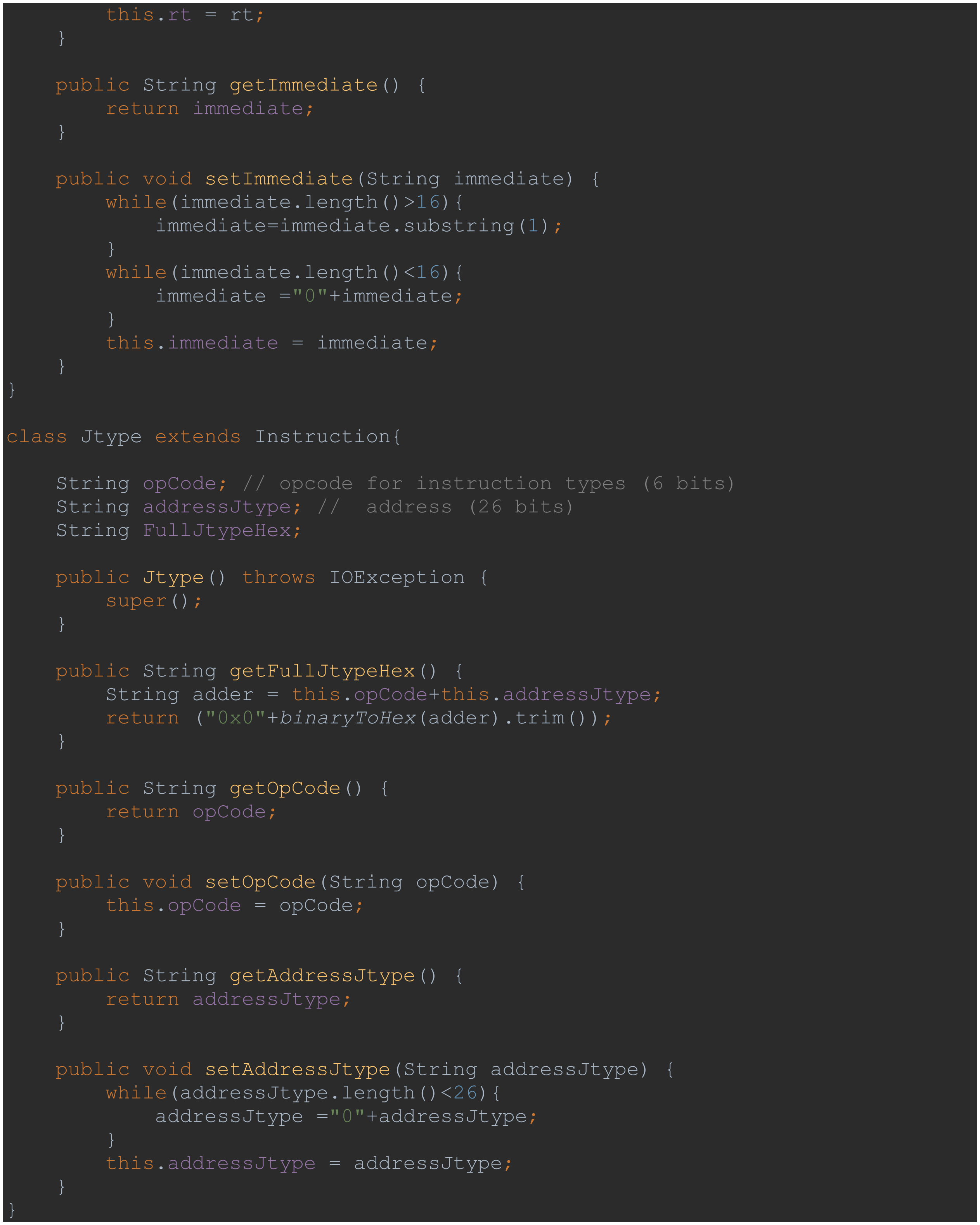
```java
                outputFile.createNewFile();

            FileWriter fileWriter = new FileWriter(outputFile,false);
            BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

            for(int i=0;i<finalResults.size();i++){
                bufferedWriter.write(finalResults.get(i));
                bufferedWriter.write("\n");
            }

            bufferedWriter.close();



        }

        }

    }


class Labels {
    String labelName;
    int labelAddress;

    public Labels(){}
    public String getLabelName() {
        return labelName;
    }

    public void setLabelName(String labelName) {
        this.labelName = labelName;
    }

    public int getLabelAddress() {
        return labelAddress;
    }
    public void setLabelAddress(int labelAddress) {
        this.labelAddress = labelAddress;
    }

}

class Rtype extends Instruction{
    String opCode; // opcode for instruction types (6 bits)
    String rs; // register containing base address (5 bits)
    String rt; // register destination/source (5 bits)
    String rd; // register destination (5 bits)
    String funcCode; // function code (identifies the specific R-format
instruction) (6 bits)
    String shiiftAmount="00000"; //shamt: shift amount (0 when N/A) (5 bits)
    String FullRtypeHex;

    public Rtype() throws IOException {
        super();
    }
```



```
    public String getFullRtypeHex() {
        String adder =
this.opCode+this.getRs()+this.rt+this.rd+this.shiiftAmount+this.funcCode;
        String result=binaryToHex(adder).trim();
        while(result.length()<8){
            result = '0'+result;
        }
        return ("0x"+result.trim());
    }

    public String getOpCode() {
        return opCode;
    }

    public void setOpCode(String opCode) {

        this.opCode = opCode;
    }

    public String getRs() {
        return rs;
    }

    public void setRs(String rs) {
        while(rs.length()<5){
            rs ="0"+rs;
        }
        this.rs = rs;
    }

    public String getRt() {
        return rt;
    }

    public void setRt(String rt) {
        while(rt.length()<5){
            rt ="0"+rt;
        }
        this.rt = rt;
    }

    public String getRd() {
        return rd;
    }

    public void setRd(String rd) {
        while(rd.length()<5){
            rd ="0"+rd;
        }
        this.rd = rd;
    }

    public String getFuncCode() {
        return funcCode;
    }

    public void setFuncCode(String funcCode) {
```



```
        this.funcCode = funcCode;
    }

    public String getShiiftAmount() {
        return shiiftAmount;
    }

    public void setShiiftAmount(String shiiftAmount) {
        while(shiiftAmount.length()<5){
            shiiftAmount ="0"+shiiftAmount;
        }
        this.shiiftAmount = shiiftAmount;
    }
}

class Itype extends Instruction{
    String opCode; // opcode for instruction types (6 bits)
    String rs; // register containing base address (5 bits)
    String rt; // register destination/source (5 bits)
    String immediate; // value or offset (16 bits)
    String FullItypeHex;

    public Itype() throws IOException {
        super();
    }

    public String getFullItypeHex() {
        String adder = this.opCode+this.rs+this.rt+this.immediate;
        return ("0x"+binaryToHex(adder).trim());
    }

    public String getOpCode() {
        return opCode;
    }

    public void setOpCode(String opCode) {
        this.opCode = opCode;
    }
    public String getRs() {
        return rs;
    }

    public void setRs(String rs) {
        while(rs.length()<5){
            rs ="0"+rs;
        }
        this.rs = rs;
    }

    public String getRt() {
        return rt;
    }

    public void setRt(String rt) {
        while(rt.length()<5){
            rt ="0"+rt;
        }
```



```java
        this.rt = rt;
    }

    public String getImmediate() {
        return immediate;
    }

    public void setImmediate(String immediate) {
        while(immediate.length()>16){
            immediate=immediate.substring(1);
        }
        while(immediate.length()<16){
            immediate ="0"+immediate;
        }
        this.immediate = immediate;
    }
}

class Jtype extends Instruction{

    String opCode; // opcode for instruction types (6 bits)
    String addressJtype; //  address (26 bits)
    String FullJtypeHex;

    public Jtype() throws IOException {
        super();
    }

    public String getFullJtypeHex() {
        String adder = this.opCode+this.addressJtype;
        return ("0x0"+binaryToHex(adder).trim());
    }

    public String getOpCode() {
        return opCode;
    }

    public void setOpCode(String opCode) {
        this.opCode = opCode;
    }

    public String getAddressJtype() {
        return addressJtype;
    }

    public void setAddressJtype(String addressJtype) {
        while(addressJtype.length()<26){
            addressJtype ="0"+addressJtype;
        }
        this.addressJtype = addressJtype;
    }
}
```