System Software and Compiler Design

Lab Assignment 1

Name: Tushar Mittal

PRN: 1032200956

Roll No: PB68

Panel: B

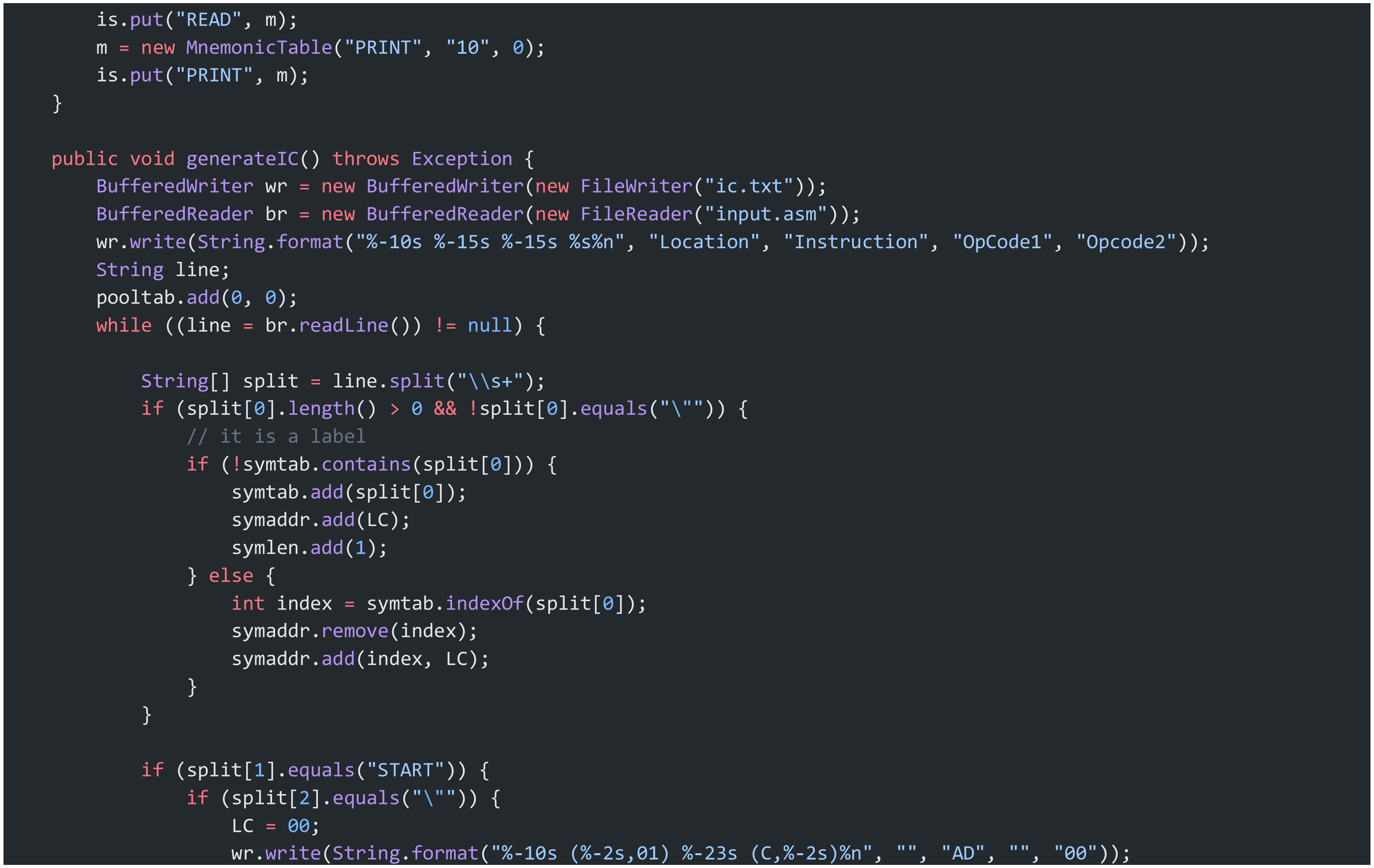
Batch: B2

Code:

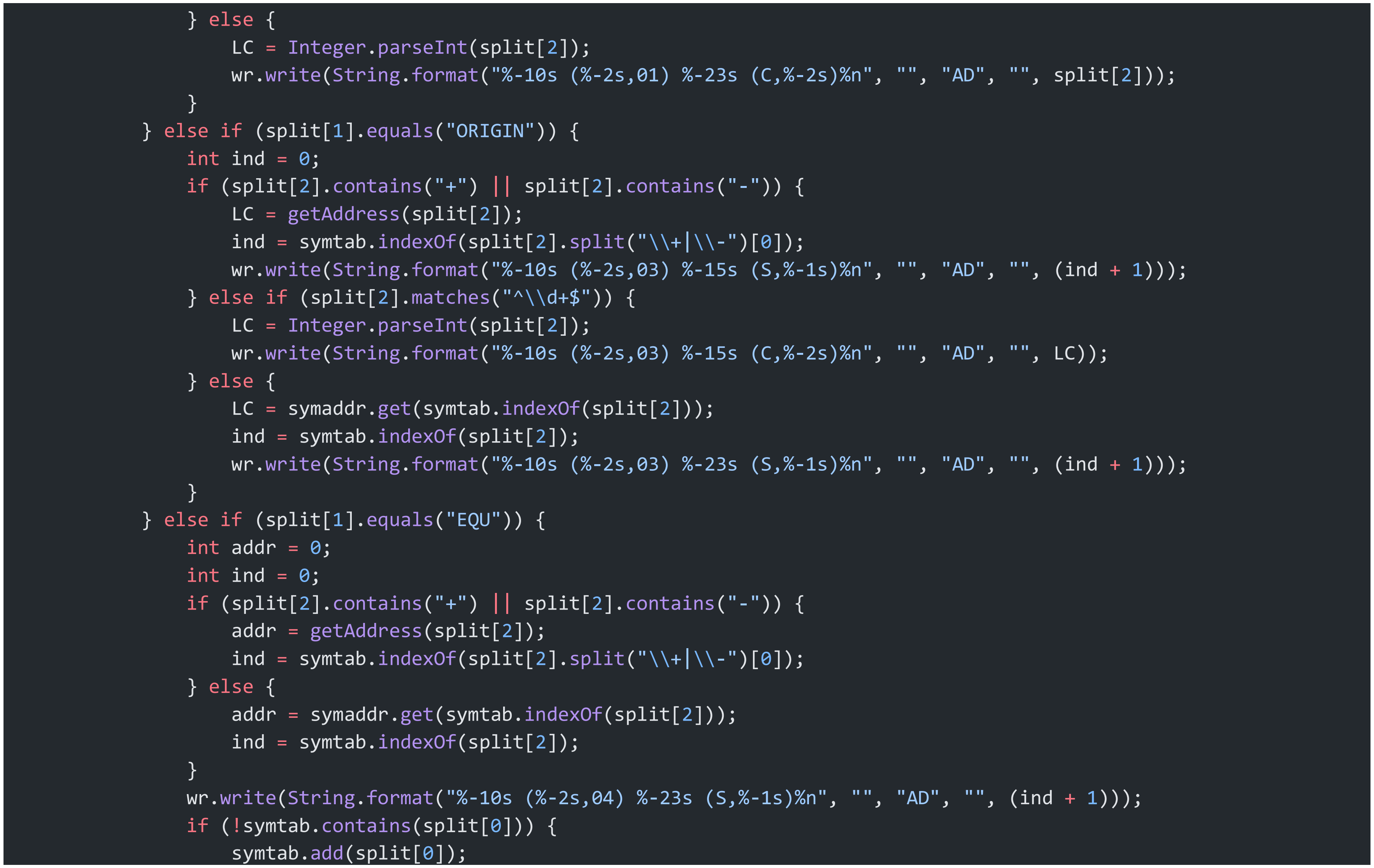
PassOne.java

```java
import java.io.BufferedReader;

import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Hashtable;
```

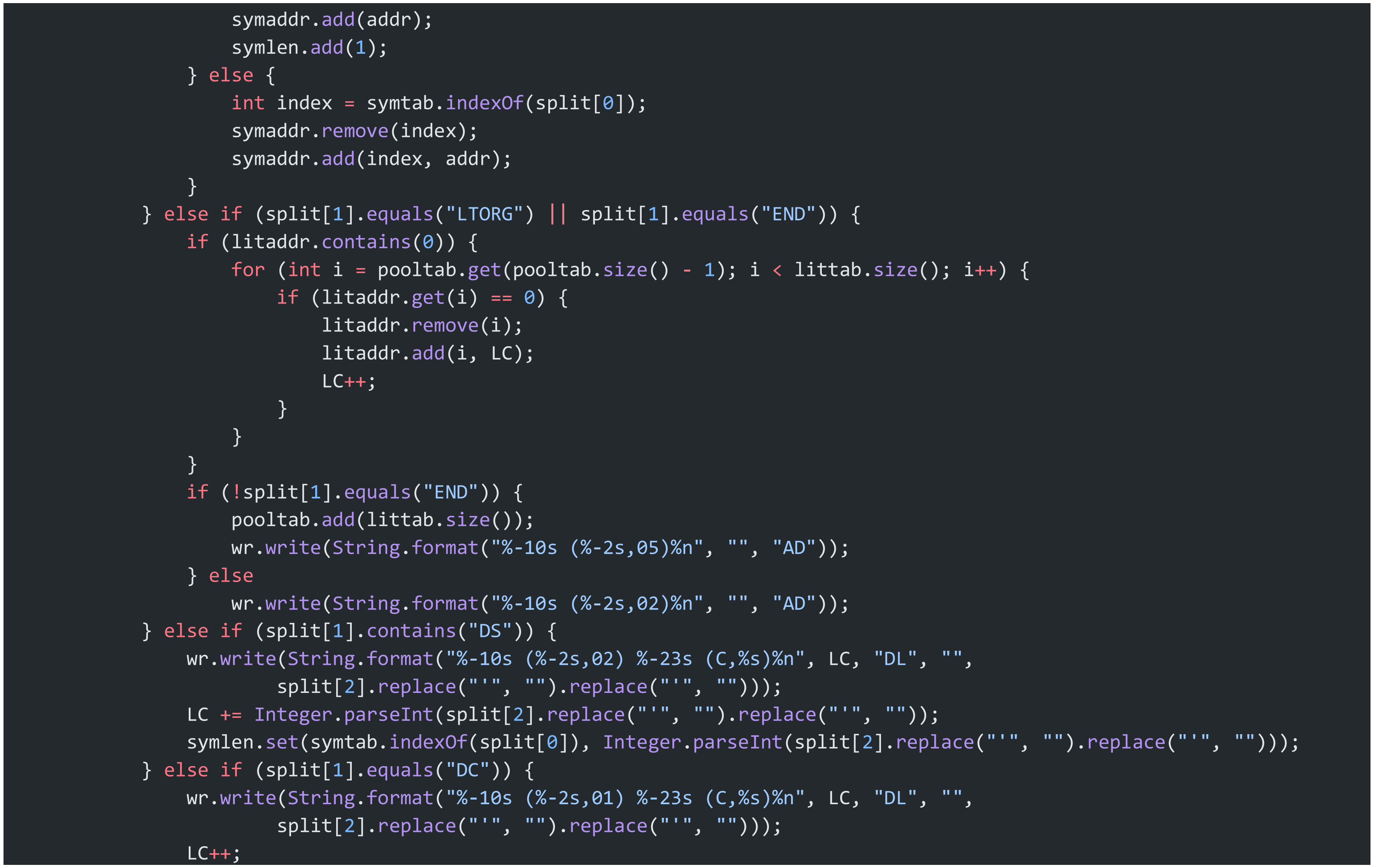
```java
public class PassOne {

    Hashtable<String, MnemonicTable> is = new Hashtable<>();
    ArrayList<String> symtab = new ArrayList<>();
    ArrayList<Integer> symaddr = new ArrayList<>();
    ArrayList<Integer> symlen = new ArrayList<>();
    ArrayList<String> littab = new ArrayList<>();
    ArrayList<Integer> litaddr = new ArrayList<>();
    ArrayList<Integer> pooltab = new ArrayList<>();
    int LC = 0;

    public void createIS() {
        MnemonicTable m = new MnemonicTable("STOP", "00", 0);
        is.put("STOP", m);
        m = new MnemonicTable("ADD", "01", 0);
        is.put("ADD", m);
        m = new MnemonicTable("SUB", "02", 0);
        is.put("SUB", m);
        m = new MnemonicTable("MULT", "03", 0);
        is.put("MULT", m);
        m = new MnemonicTable("MOVER", "04", 0);
        is.put("MOVER", m);
        m = new MnemonicTable("MOVEM", "05", 0);
        is.put("MOVEM", m);
        m = new MnemonicTable("COMP", "06", 0);
        is.put("COMP", m);
        m = new MnemonicTable("BC", "07", 0);
        is.put("BC", m);
        m = new MnemonicTable("DIV", "08", 0);
        is.put("DIV", m);
        m = new MnemonicTable("READ", "09", 0);
```



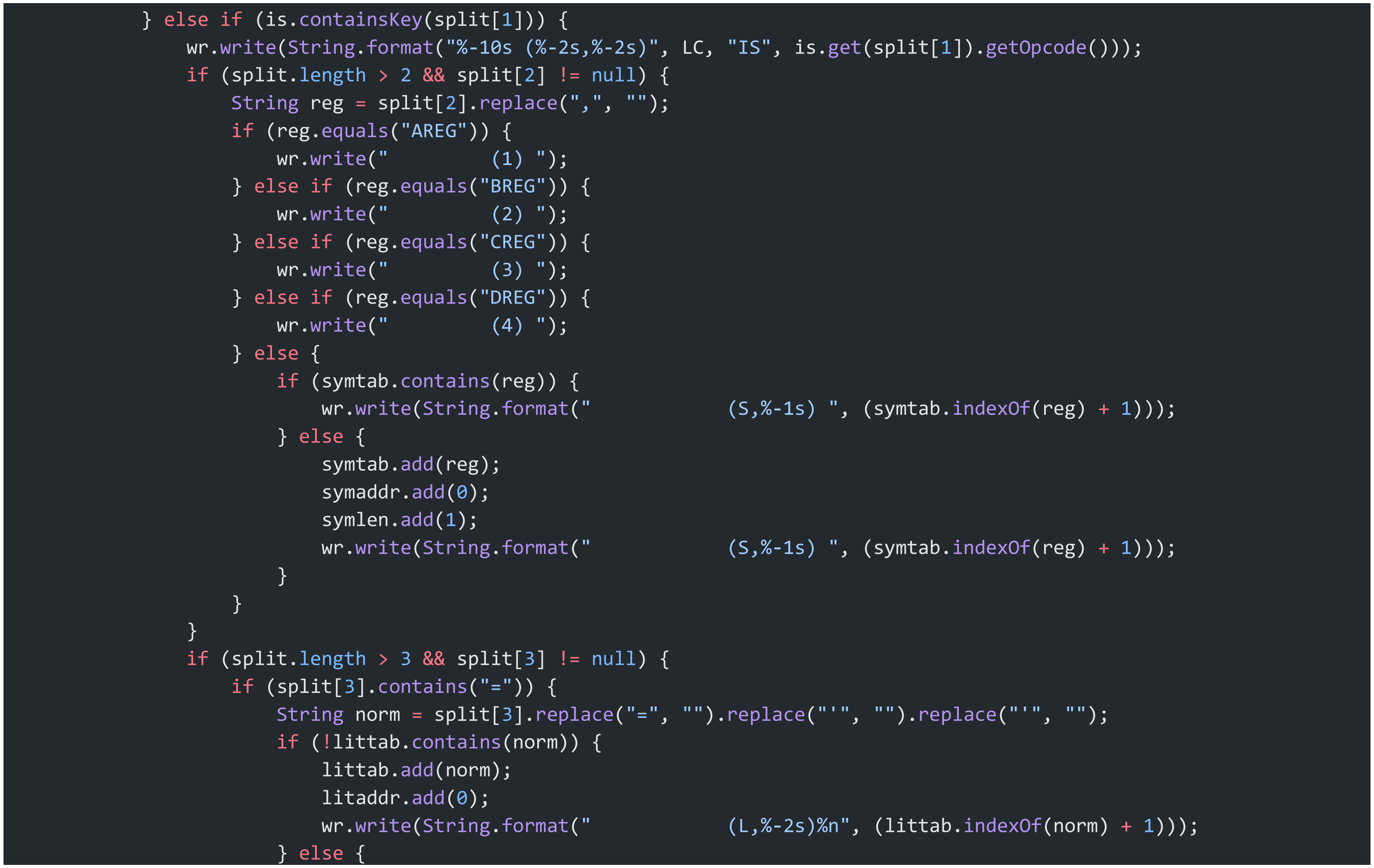
```java
        is.put("READ", m);
        m = new MnemonicTable("PRINT", "10", 0);
        is.put("PRINT", m);
    }

    public void generateIC() throws Exception {
        BufferedWriter wr = new BufferedWriter(new FileWriter("ic.txt"));
        BufferedReader br = new BufferedReader(new FileReader("input.asm"));
        wr.write(String.format("%-10s %-15s %-15s %s%n", "Location", "Instruction", "OpCode1", "Opcode2"));
        String line;
        pooltab.add(0, 0);
        while ((line = br.readLine()) != null) {

            String[] split = line.split("\\s+");
            if (split[0].length() > 0 && !split[0].equals("\"")) {
                // it is a label
                if (!symtab.contains(split[0])) {
                    symtab.add(split[0]);
                    symaddr.add(LC);
                    symlen.add(1);
                } else {
                    int index = symtab.indexOf(split[0]);
                    symaddr.remove(index);
                    symaddr.add(index, LC);
                }
            }

            if (split[1].equals("START")) {
                if (split[2].equals("\"")) {
                    LC = 00;
                    wr.write(String.format("%-10s (%-2s,01) %-23s (C,%-2s)%n", "", "AD", "", "00"));
```



```java
                } else {
                    LC = Integer.parseInt(split[2]);
                    wr.write(String.format("%-10s (%-2s,01) %-23s (C,%-2s)%n", "", "AD", "", split[2]));
                }
            } else if (split[1].equals("ORIGIN")) {
                int ind = 0;
                if (split[2].contains("+") || split[2].contains("-")) {
                    LC = getAddress(split[2]);
                    ind = symtab.indexOf(split[2].split("\\+|\\-")[0]);
                    wr.write(String.format("%-10s (%-2s,03) %-15s (S,%-1s)%n", "", "AD", "", (ind + 1)));
                } else if (split[2].matches("^\\d+$")) {
                    LC = Integer.parseInt(split[2]);
                    wr.write(String.format("%-10s (%-2s,03) %-15s (C,%-2s)%n", "", "AD", "", LC));
                } else {
                    LC = symaddr.get(symtab.indexOf(split[2]));
                    ind = symtab.indexOf(split[2]);
                    wr.write(String.format("%-10s (%-2s,03) %-23s (S,%-1s)%n", "", "AD", "", (ind + 1)));
                }
            } else if (split[1].equals("EQU")) {
                int addr = 0;
                int ind = 0;
                if (split[2].contains("+") || split[2].contains("-")) {
                    addr = getAddress(split[2]);
                    ind = symtab.indexOf(split[2].split("\\+|\\-")[0]);
                } else {
                    addr = symaddr.get(symtab.indexOf(split[2]));
                    ind = symtab.indexOf(split[2]);
                }
                wr.write(String.format("%-10s (%-2s,04) %-23s (S,%-1s)%n", "", "AD", "", (ind + 1)));
                if (!symtab.contains(split[0])) {
                    symtab.add(split[0]);
```

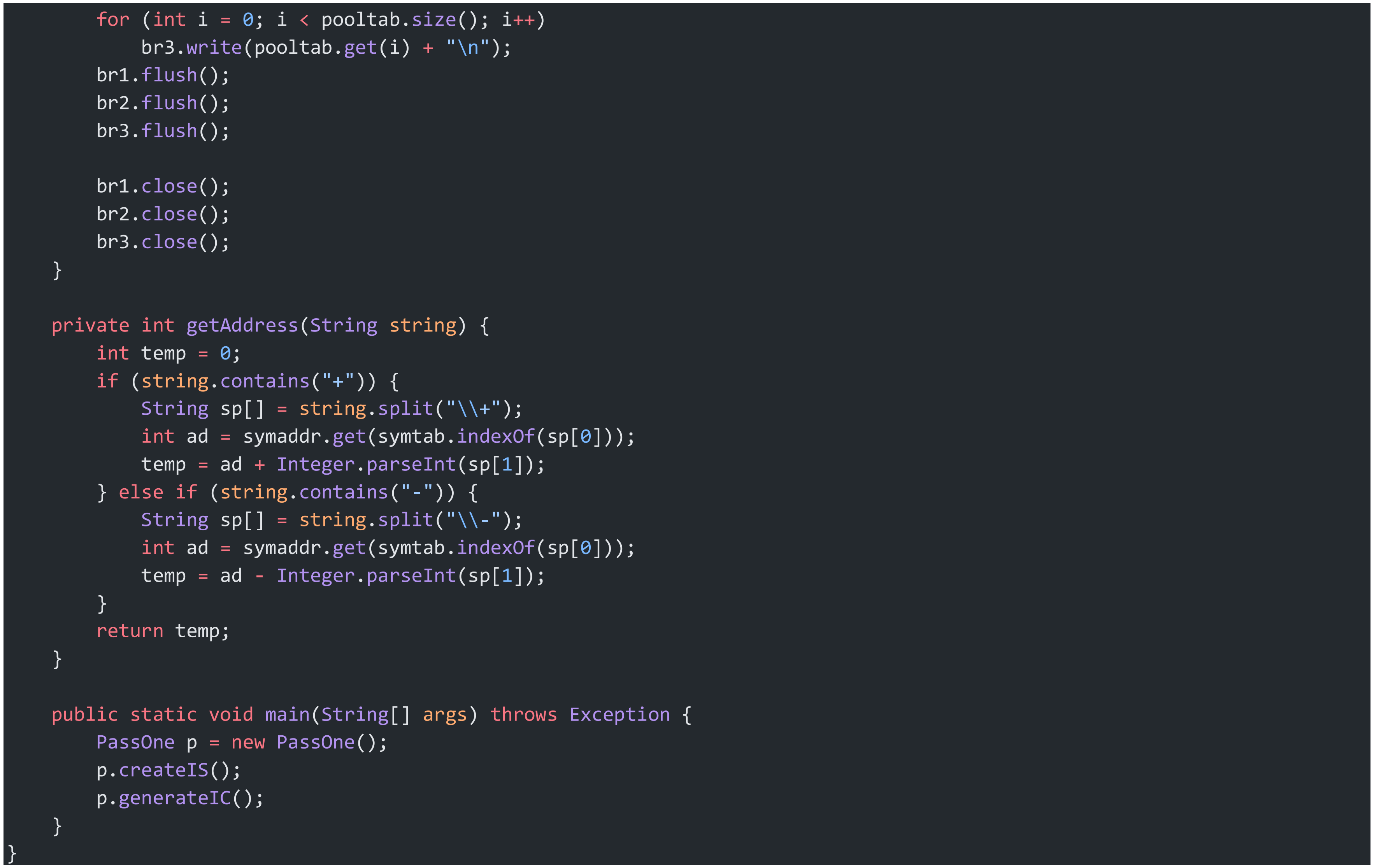


```
                symaddr.add(addr);
                symlen.add(1);
            } else {
                int index = symtab.indexOf(split[0]);
                symaddr.remove(index);
                symaddr.add(index, addr);
            }
        } else if (split[1].equals("LTORG") || split[1].equals("END")) {
            if (litaddr.contains(0)) {
                for (int i = pooltab.get(pooltab.size() - 1); i < littab.size(); i++) {
                    if (litaddr.get(i) == 0) {
                        litaddr.remove(i);
                        litaddr.add(i, LC);
                        LC++;
                    }
                }
            }
            if (!split[1].equals("END")) {
                pooltab.add(littab.size());
                wr.write(String.format("%-10s (%-2s,05)%n", "", "AD"));
            } else
                wr.write(String.format("%-10s (%-2s,02)%n", "", "AD"));
        } else if (split[1].contains("DS")) {
            wr.write(String.format("%-10s (%-2s,02) %-23s (C,%s)%n", LC, "DL", "",
                    split[2].replace("'", "").replace("'", "")));
            LC += Integer.parseInt(split[2].replace("'", "").replace("'", ""));
            symlen.set(symtab.indexOf(split[0]), Integer.parseInt(split[2].replace("'", "").replace("'", "")));
        } else if (split[1].equals("DC")) {
            wr.write(String.format("%-10s (%-2s,01) %-23s (C,%s)%n", LC, "DL", "",
                    split[2].replace("'", "").replace("'", "")));
            LC++;
```



```
            } else if (is.containsKey(split[1])) {
                wr.write(String.format("%-10s (%-2s,%-2s)", LC, "IS", is.get(split[1]).getOpcode()));
                if (split.length > 2 && split[2] != null) {
                    String reg = split[2].replace(",", "");
                    if (reg.equals("AREG")) {
                        wr.write("         (1) ");
                    } else if (reg.equals("BREG")) {
                        wr.write("         (2) ");
                    } else if (reg.equals("CREG")) {
                        wr.write("         (3) ");
                    } else if (reg.equals("DREG")) {
                        wr.write("         (4) ");
                    } else {
                        if (symtab.contains(reg)) {
                            wr.write(String.format("            (S,%-1s) ", (symtab.indexOf(reg) + 1)));
                        } else {
                            symtab.add(reg);
                            symaddr.add(0);
                            symlen.add(1);
                            wr.write(String.format("            (S,%-1s) ", (symtab.indexOf(reg) + 1)));
                        }
                    }
                }
                if (split.length > 3 && split[3] != null) {
                    if (split[3].contains("=")) {
                        String norm = split[3].replace("=", "").replace("'", "").replace("'", "");
                        if (!littab.contains(norm)) {
                            littab.add(norm);
                            litaddr.add(0);
                            wr.write(String.format("            (L,%-2s)%n", (littab.indexOf(norm) + 1)));
                        } else {
```

```java
                            wr.write(String.format("            (L,%-2s)%n", (littab.indexOf(norm) + 1)));
                        }

                    } else if (symtab.contains(split[3])) {
                        wr.write(String.format("            (S,%-1s)%n", (symtab.indexOf(split[3]) + 1)));

                    } else {
                        symtab.add(split[3]);
                        symaddr.add(0);
                        symlen.add(1);
                        wr.write(String.format("            (S,%-1s)%n", (symtab.indexOf(split[3]) + 1)));
                    }
                }
                LC++;
            }
        }
        wr.flush();
        wr.close();
        BufferedWriter br1 = new BufferedWriter(new FileWriter("sym.txt"));
        BufferedWriter br2 = new BufferedWriter(new FileWriter("lit.txt"));
        BufferedWriter br3 = new BufferedWriter(new FileWriter("pool.txt"));

        br1.write(String.format("%-10s %-10s %-10s %-10s%n", "ID", "Symbol", "Address", "Length"));
        for (int i = 0; i < symtab.size(); i++)
            br1.write(String.format("%-10s %-10s %-10s %-10s%n", i + 1, symtab.get(i), symaddr.get(i), symlen.get(i)));

        br2.write(String.format("%-10s %-10s%n", "Literal", "Address"));
        for (int i = 0; i < littab.size(); i++)
            br2.write(String.format("%-10s %-10s%n", littab.get(i), litaddr.get(i)));

        br3.write("Pooltab\n");
```



```java
        for (int i = 0; i < pooltab.size(); i++)
            br3.write(pooltab.get(i) + "\n");
        br1.flush();
        br2.flush();
        br3.flush();

        br1.close();
        br2.close();
        br3.close();
    }

    private int getAddress(String string) {
        int temp = 0;
        if (string.contains("+")) {
            String sp[] = string.split("\\+");
            int ad = symaddr.get(symtab.indexOf(sp[0]));
            temp = ad + Integer.parseInt(sp[1]);
        } else if (string.contains("-")) {
            String sp[] = string.split("\\-");
            int ad = symaddr.get(symtab.indexOf(sp[0]));
            temp = ad - Integer.parseInt(sp[1]);
        }
        return temp;
    }

    public static void main(String[] args) throws Exception {
        PassOne p = new PassOne();
        p.createIS();
        p.generateIC();
    }
}
```

MnemoicTable.java

```java
public class MnemonicTable {
    private String mnemonic;
    private String opcode;
    private int format;

    public MnemonicTable(String mnemonic, String opcode, int format) {
        this.mnemonic = mnemonic;
        this.opcode = opcode;
        this.format = format;
    }

    public String getMnemonic() {
        return mnemonic;
    }

    public void setMnemonic(String mnemonic) {
        this.mnemonic = mnemonic;
    }

    public String getOpcode() {
        return opcode;
    }

    public void setOpcode(String opcode) {
        this.opcode = opcode;
    }

    public int getFormat() {
        return format;
    }

    public void setFormat(int format) {
        this.format = format;
    }

    @Override
    public String toString() {
        return "MnemonicTable{" +
                "mnemonic='" + mnemonic + '\'' +
                ", opcode='" + opcode + '\'' +
                ", format=" + format +
                '}';
    }
}
```

Input:

Input.asm

```
"  START    "   "

"  MOVEM   AREG, S1

L1    DIV    BREG, S2

"   MOVEM    BREG, S1

L2    EQU    L1

"   MOVEM    BREG, S1

S1     DC     '4'   "

S2     DS     '3'   "

    END
```

Output:

ic.txt

```
Location   Instruction    OpCode1        Opcode2

                (AD,01)                     (C,00)

0               (IS,05)         (1)          (S,1)

1               (IS,08)         (2)          (S,3)

2               (IS,05)         (2)          (S,1)
```

```
        (AD,04)                 (S,2)
3       (IS,05)     (2)         (S,1)
4       (DL,01)                 (C,4)
5       (DL,02)                 (C,3)
        (AD,02)
```

Lit.txt

```
Literal   Address
```

pool.txt

```
Pooltab
0
```

sym.txt

```
ID     Symbol    Address    Length
1      S1        4          1
2      L1        1          1
3      S2        5          3
4      L2        1          1
```