CSE 206 ASSINGMENT 1 REPORT

In the main method in the ComputerOrganization201808080845 Class, the text file given to the system as an argument is parsed.

Text file input type must be like -> 0 START
1 LOAD 20
2 STORE 200

For example, in line 2 store 200, "store" will be put in the “instructions” ArrayList and "200" will be put in the “operands” ArrayList. Rows that do not have a specific value will be 0.

The lists in the ComputerOrganization201808080845 class are sent to the CpuEmulator Class, and the status analysis of the values added to the lists and whether they are run according to the situation is evaluated in this class.

Key things in code

```
class CpuEmulator {

    ArrayList<InstructionSet> instructions;
    ArrayList<Integer> operands;
    int IR = 0;
    int[] M = new int[256];
    int AC = 0;
    int Flag = 0;
    boolean Halt;
```

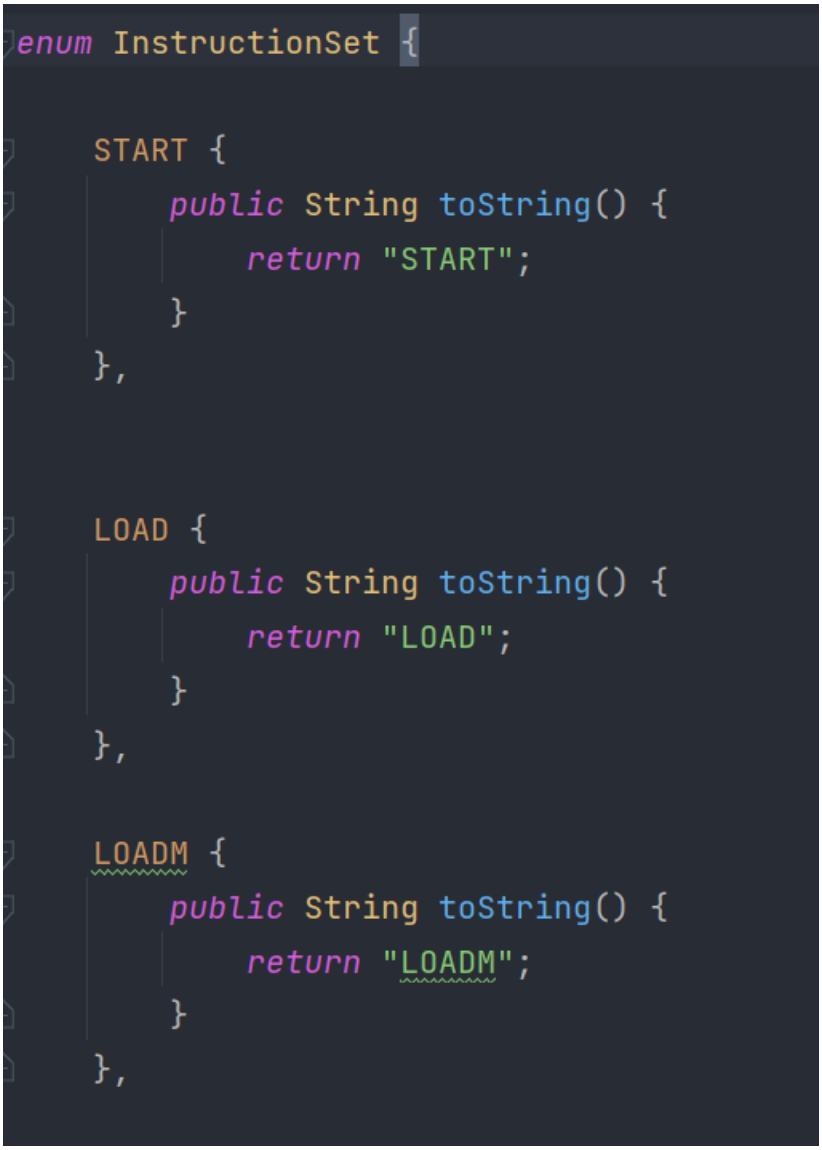
The instruction Arraylist contains the constructs to be run in order.
The Operands ArrayList contains the operands that the respective constructors must perform.
Halt the boolean variable is about whether the code will continue to run.
The "M" array represents memory size.
The IR stands for “Instruction register” and represents the instruction to be executed.
AC stands for the accumulator.

```
if (!instructions.contains(InstructionSet.START)) {
    System.exit( status: 0);
}

while (!instructions.get(0).equals(InstructionSet.START)) {
    instructions.remove( index: 0);
    operands.remove( index: 0);
}
```

This section was written in order to eliminate whether the code contains the start instructor, and if it does, the pre-start constructs are eliminated.

```
while (!Halt) {

    InstructionSet instruction;
    int operand;
    instruction = instructions.get(IR);
    operand = operands.get(IR);

    IR++;

    switch (instruction) {

        case START:
            break;
        case LOAD:
            LOAD(operand);
            break;
        case LOADM:
            LOADM(operand);
            break;
```

This section runs the relevant instructor's method by scanning the lists created based on the scanned text.



```
enum InstructionSet {

    START {
        public String toString() {
            return "START";
        }
    },


    LOAD {
        public String toString() {
            return "LOAD";
        }
    },

    LOADM {
        public String toString() {
            return "LOADM";
        }
    },
```

In the switch-case section, I used the enum structure while directing the input to the method. In this way, I tried to prevent the code from crashing when making changes to strings associated with more than one point, and also due to some details such as typos in the conditions of the strings.

Instructors Related Methods

```java
public void JMP(int X) {
    IR = X;
}

public void ADD(int X) {
    AC += X;
}

public void ADDM(int line) {
    AC += M[line];
}

public void SUBM(int line) {
    AC -= M[line];
}

public void SUB(int X) {
    AC += X;
}

public void MUL(int X) {

    AC *= X;

}
```

```java
public void LOAD(int X) {
    AC = X;
}

public void LOADM(int line) {
    AC = M[line];
}

public void STORE(int line) {
    M[line] = AC;
}

public void CMPM(int line) {
    Flag = Integer.compare(AC, M[line]);
}

public void CJMP(int X) {
    if (Flag > 0) {
        IR = X;
    }
}
```

```java
public void MULM(int line) {
    AC *= M[line];
}

public void DISP() {

    System.out.println("AC: " + AC);

}

public void HALT() {

    Halt = true;

}
```