CSE 206 Computer Organization — ALİ ÇOLAK

CPU EMULATOR — 201908080064

23/05/2022

1 - > Firstly, I've define an accumulator, a flag, a program counter, an integer array with size of 256 to keep the memory, an arraylist to keep the instructions and a boolean variable to keep the program status.

```
public static boolean STATUS = false;
public static int AC, PC, F = 0;
public static int[] M = new int[256];
public static ArrayList<String> INSTRUCTIONS ;
```

2 -> Then I take the filename as program arguments and pass the filename as a parameter to the getData method and assign the returned ArrayList to my INSTRUCTIONS ArrayList

```
INSTRUCTIONS = getData(args[0]);
```

3 - > Then I created a method called getData which returns the instructions as a String ArrayList to read the data from the file.

In this method, I get the lines by reading the lines of the file.

I don't add ArrayList if line is comment line.

I capitalize the entire line and discard any unnecessary spaces.

I add the last string to ArrayList

```
private static ArrayList<String> getData(String path)  {

    ArrayList<String> commands = new ArrayList<>();
    File file = new File(path);
    try {
        Scanner scanner = new Scanner(file);
        while (scanner.hasNextLine()) {
            String str = scanner.nextLine().toUpperCase().trim();
            if (!str.contains("%")) {
                while (str.contains("  "))
                    str = str.replace( target: "  ",  replacement: " ");
                str = str.substring(str.indexOf(" ") + 1);
                if (!str.equals("")) commands.add(str);
            }
        }
        scanner.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return commands;
}
```

4 -> I then define a for loop to read the lines. I assign the value in the ArrayList to the array with the split function

Example

currentInstruction = START or LOAD 200

If for the DISP , START and HALT functions

instruction = START

For other functions

instruction = LOAD

value = 200

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

    String[] currentInstruction = INSTRUCTIONS.get(i).split( regex: " ");

    PC = i + 1;

    String instruction = currentInstruction[0];
    int value = 0;

    if (!(instruction.equals("START") || instruction.equals("DISP") || instruction.equals("HALT"))) {
        value = Integer.parseInt(currentInstruction[1]);
    }
```

5 - > Then I defined my methods for instructions based on the basic instructions set you gave

```
private static void START() {
    STATUS = true;
}

private static void LOAD(int value) {
    AC = value;
}

private static void LOADM(int index) {
    AC = M[index];
}

private static void STORE(int index) {
    M[index] = AC;
}

private static void CMPM(int index) {
    if (AC > M[index]) {
        F = 1;
    } else if (AC < M[index]) {
        F = -1;
    } else {
        F = 0;
    }
}

private static void CJMP(int go_to) {
    if (F > 0) {
        PC = go_to;
    }
}

private static void JMP(int go_to) {
    PC = go_to;
}
```

```
private static void ADD(int value) {
    AC += value;
}

private static void ADDM(int index) {
    AC += M[index];
}

private static void SUB(int value) {
    AC -= value;
}

private static void SUBM(int index) {
    AC -= M[index];
}

private static void MUL(int value) {
    AC *= value;
}

private static void MULM(int index) {
    AC *= M[index];
}

private static void DISP() {
    System.out.println(AC);
}

private static void HALT() {
    STATUS = false;
}
```

6 - Finally, I call the necessary methods with the switch case according to the lines I have received.

```
if (STATUS || instruction.equals("START"))
    switch (instruction) {
        case "START" ->
                START();
        case "LOAD" ->
                LOAD(value);
        case "LOADM" ->
                LOADM(value);
        case "STORE" ->
                STORE(value);
        case "CMPM" ->
                CMPM(value);
        case "CJMP" -> {
            CJMP(value);
            i = PC - 1;
        }
        case "JMP" -> {
            JMP(value);
            i = PC - 1;
        }
        case "ADD" ->
                ADD(value);
        case "ADDM" ->
                ADDM(value);
        case "SUBM" ->
                SUBM(value);
        case "SUB" ->
                SUB(value);
        case "MUL" ->
                MUL(value);
        case "MULM" ->
                MULM(value);
        case "DISP" ->
                DISP();
        case "HALT" ->
                HALT();
    }
```