Phase 4: Activation Records and Instruction Selection

Student #1 - Micah Galos | SID - 862082339
Student #2 - Najmeh Mohammadi Arani | SID - 862216499
Student #3 - Steven Strickland | SID - 862155853

Due: Dec 5, 2021, 11:59 PM

1. Requirements and Specifications

The requirements of the final phase are to translate various VaporM programs into MIPs. The specifics needed are to map VaporM registers and stacks to MIPs. As well as map VaporM instructions to MIPs instructions. Since the stack grows from high address to low address--we need to traverse the stack downwards. When we are done with the stack. We need to pop all the registers in the stack and revert back to high address once finishing.

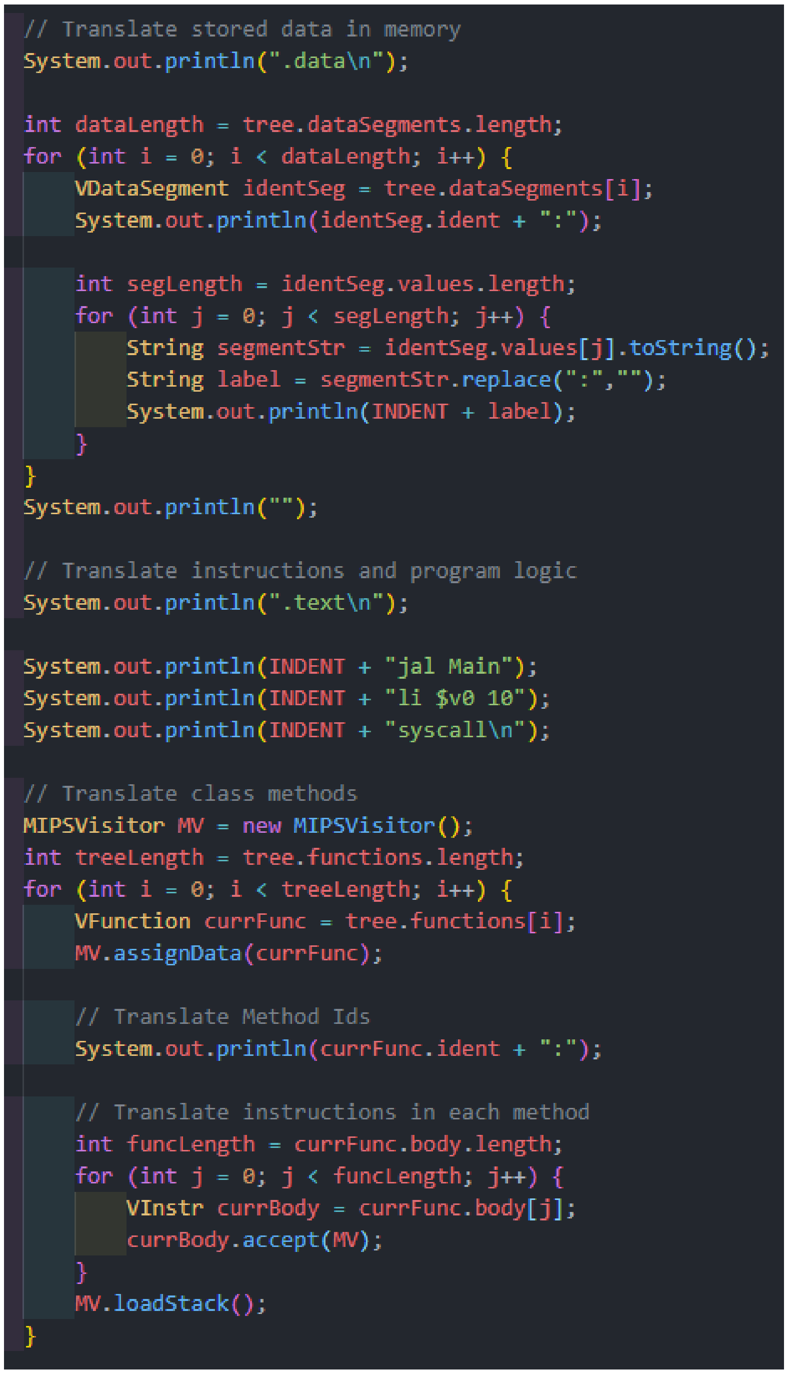
2. Design

The design approach of this phase was very straightforward. Given the updated V2VM.java file, we only need to figure out translating the various MIPs instructions given the vapor-parser libraries. There is only one new file added which is the MIPSVisitor which handles the translation.

a. Phase 4 Additions

i. V2VM Updates

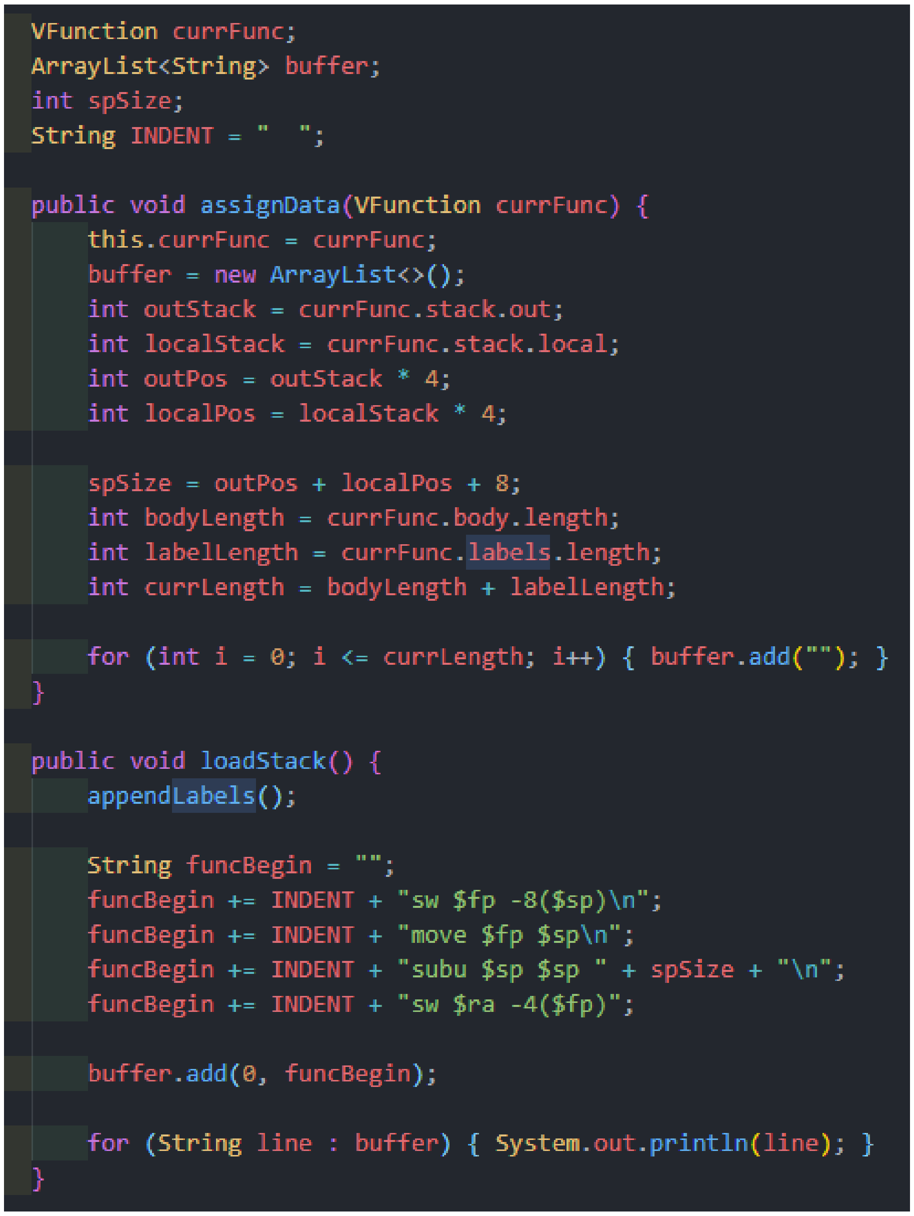
Main translation of VaporM to MIPS occurs in the main of V2VM. These two for-loops output the data segments and methods from the VaporM input files from VaporProgram’s object tree.



```
// Translate stored data in memory
System.out.println(".data\n");

int dataLength = tree.dataSegments.length;
for (int i = 0; i < dataLength; i++) {
    VDataSegment identSeg = tree.dataSegments[i];
    System.out.println(identSeg.ident + ":");

    int segLength = identSeg.values.length;
    for (int j = 0; j < segLength; j++) {
        String segmentStr = identSeg.values[j].toString();
        String label = segmentStr.replace(":","");
        System.out.println(INDENT + label);
    }
}
System.out.println("");

// Translate instructions and program logic
System.out.println(".text\n");

System.out.println(INDENT + "jal Main");
System.out.println(INDENT + "li $v0 10");
System.out.println(INDENT + "syscall\n");

// Translate class methods
MIPSVisitor MV = new MIPSVisitor();
int treeLength = tree.functions.length;
for (int i = 0; i < treeLength; i++) {
    VFunction currFunc = tree.functions[i];
    MV.assignData(currFunc);

    // Translate Method Ids
    System.out.println(currFunc.ident + ":");

    // Translate instructions in each method
    int funcLength = currFunc.body.length;
    for (int j = 0; j < funcLength; j++) {
        VInstr currBody = currFunc.body[j];
        currBody.accept(MV);
    }
    MV.loadStack();
}
```

ii. MIPSVisitor

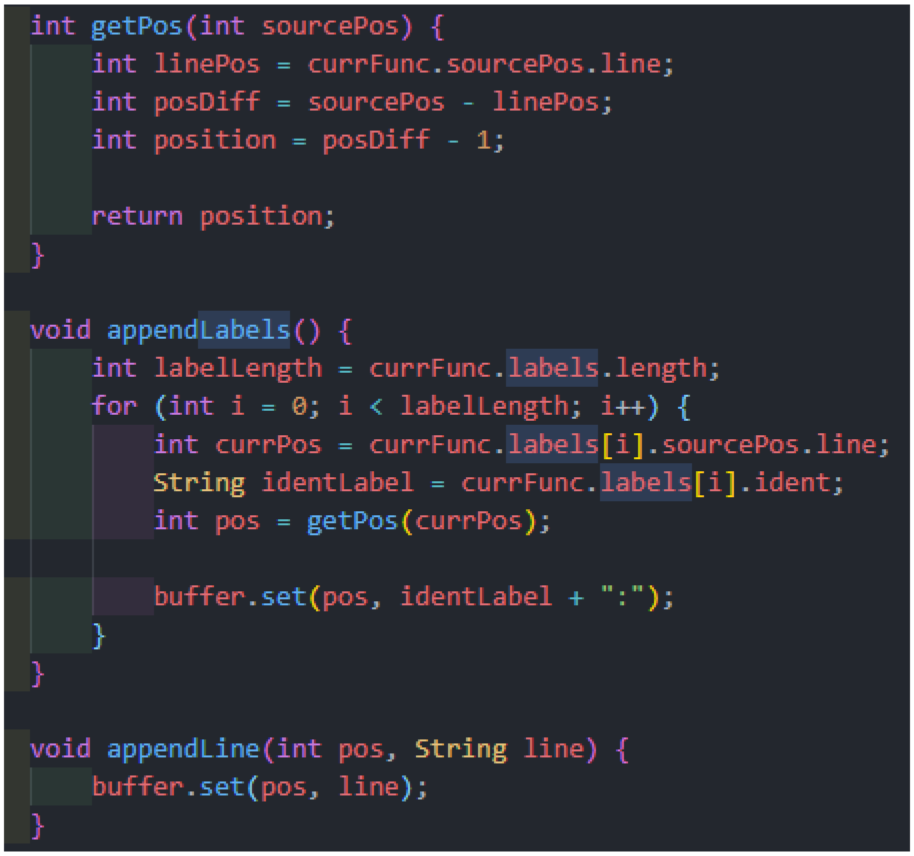
MIPSVisitor is the extended class of the Visitor maintained in the vapor-parser.jar file. The two images shown are the initial set-up which helps V2VM.java at translation. Pushing and popping the stack every time we access a class's method and assigning the data based on their positions in the stack--handled by a buffer arraylist. Method loadStack is where we initialize the stack and their pointers for each method accessed in the class. Below are portions of the Visitor code.



```java
VFunction currFunc;
ArrayList<String> buffer;
int spSize;
String INDENT = "  ";

public void assignData(VFunction currFunc) {
    this.currFunc = currFunc;
    buffer = new ArrayList<>();
    int outStack = currFunc.stack.out;
    int localStack = currFunc.stack.local;
    int outPos = outStack * 4;
    int localPos = localStack * 4;

    spSize = outPos + localPos + 8;
    int bodyLength = currFunc.body.length;
    int labelLength = currFunc.labels.length;
    int currLength = bodyLength + labelLength;

    for (int i = 0; i <= currLength; i++) { buffer.add(""); }
}

public void loadStack() {
    appendLabels();

    String funcBegin = "";
    funcBegin += INDENT + "sw $fp -8($sp)\n";
    funcBegin += INDENT + "move $fp $sp\n";
    funcBegin += INDENT + "subu $sp $sp " + spSize + "\n";
    funcBegin += INDENT + "sw $ra -4($fp)";

    buffer.add(0, funcBegin);

    for (String line : buffer) { System.out.println(line); }
}
```

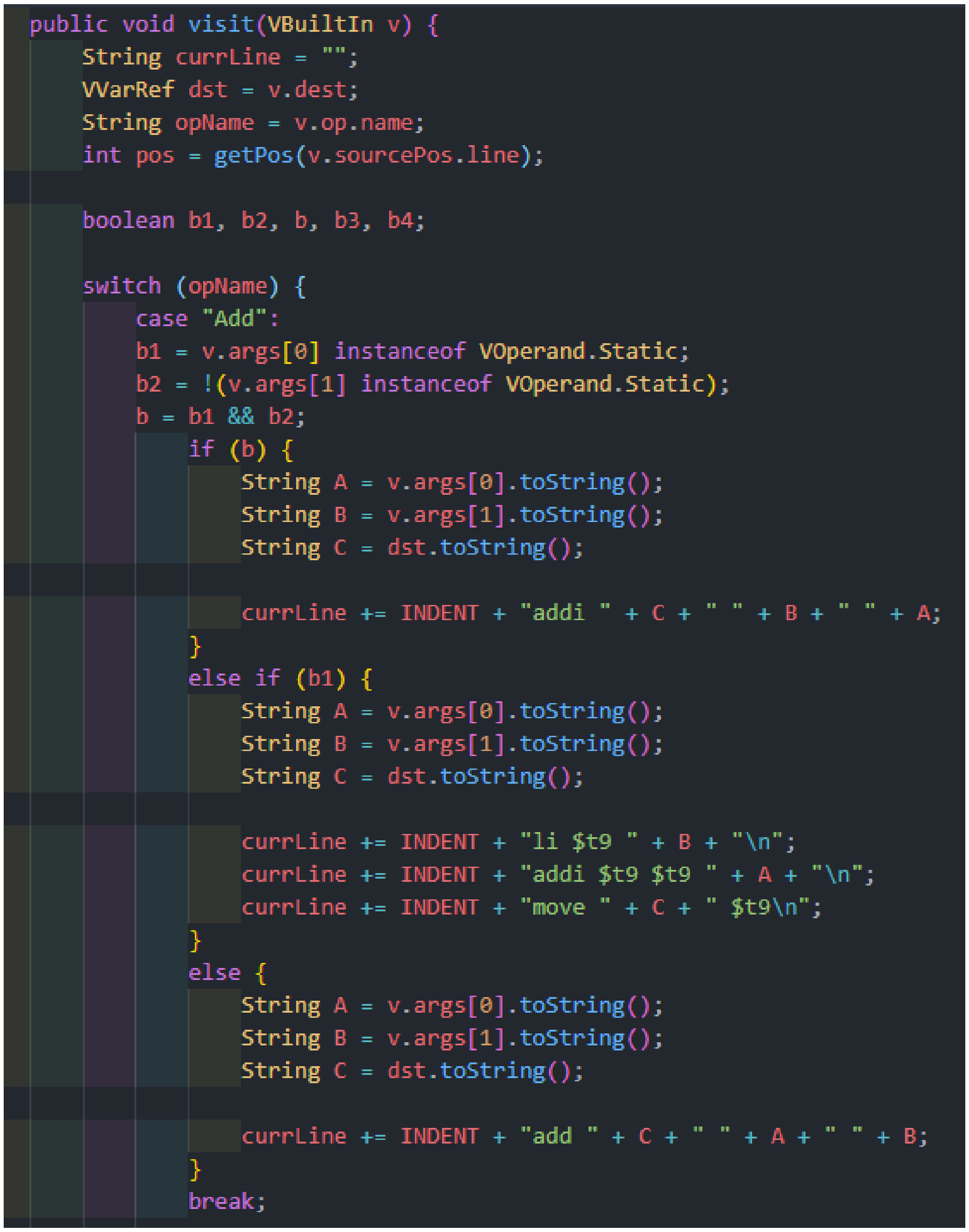
Data Set-Up for Translation

This section here handles translating registers and stack positions from VaporM to MIPS in the visitor class. AppendLabels initializes the stack for each method in the class upon entering.



```java
int getPos(int sourcePos) {
    int linePos = currFunc.sourcePos.line;
    int posDiff = sourcePos - linePos;
    int position = posDiff - 1;

    return position;
}

void appendLabels() {
    int labelLength = currFunc.labels.length;
    for (int i = 0; i < labelLength; i++) {
        int currPos = currFunc.labels[i].sourcePos.line;
        String identLabel = currFunc.labels[i].ident;
        int pos = getPos(currPos);

        buffer.set(pos, identLabel + ":");
    }
}

void appendLine(int pos, String line) {
    buffer.set(pos, line);
}
```

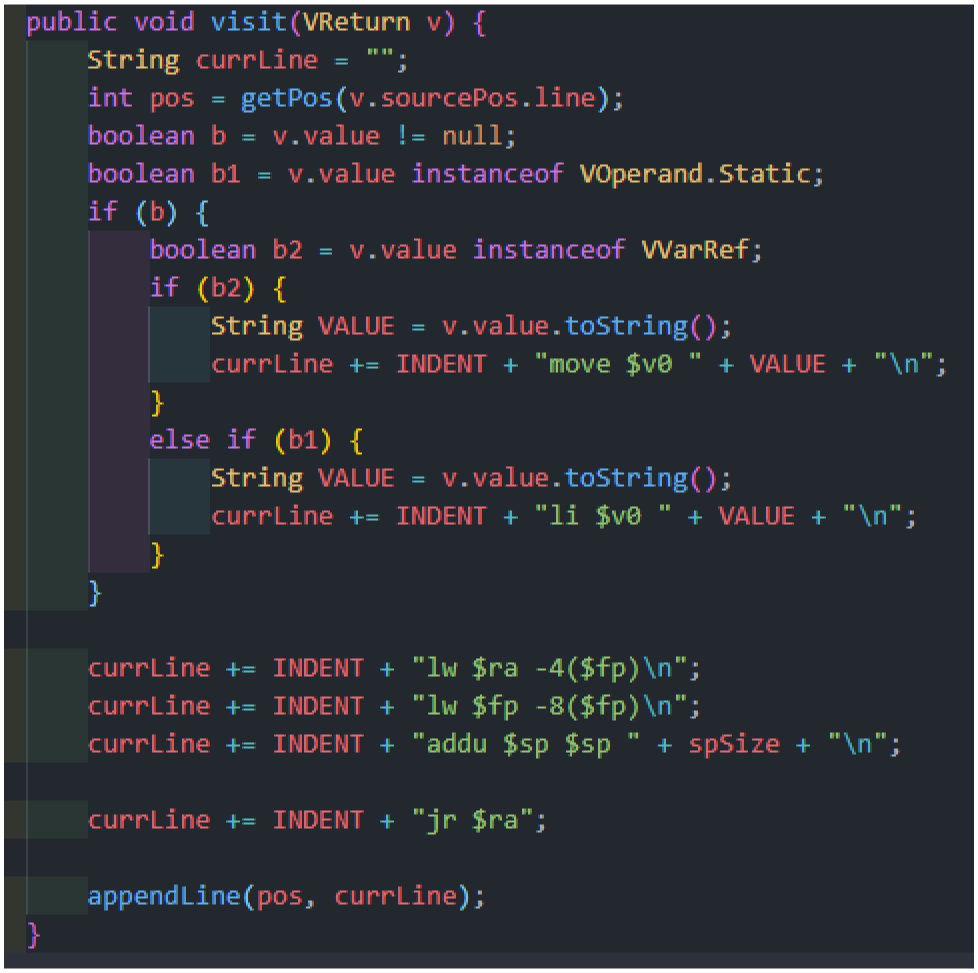
Translation Methods



```java
public void visit(VBuiltIn v) {
    String currLine = "";
    VVarRef dst = v.dest;
    String opName = v.op.name;
    int pos = getPos(v.sourcePos.line);

    boolean b1, b2, b, b3, b4;

    switch (opName) {
        case "Add":
        b1 = v.args[0] instanceof VOperand.Static;
        b2 = !(v.args[1] instanceof VOperand.Static);
        b = b1 && b2;
            if (b) {
                String A = v.args[0].toString();
                String B = v.args[1].toString();
                String C = dst.toString();

                currLine += INDENT + "addi " + C + " " + B + " " + A;
            }
            else if (b1) {
                String A = v.args[0].toString();
                String B = v.args[1].toString();
                String C = dst.toString();

                currLine += INDENT + "li $t9 " + B + "\n";
                currLine += INDENT + "addi $t9 $t9 " + A + "\n";
                currLine += INDENT + "move " + C + " $t9\n";
            }
            else {
                String A = v.args[0].toString();
                String B = v.args[1].toString();
                String C = dst.toString();

                currLine += INDENT + "add " + C + " " + A + " " + B;
            }
            break;
```

Visit Method for an Add Built-In Instruction Op

Handled by a switch statement for each operation name. The variable currLine is the string we want to translate the instructions from VaporM to MIPs. If the op from VaporM was an Add operation, then we translate to which makes up an Add operation in MIPS. Same goes for the rest of the Built-In Ops and their passed in arguments and if those arguments are an instance of the vapor-parser type.

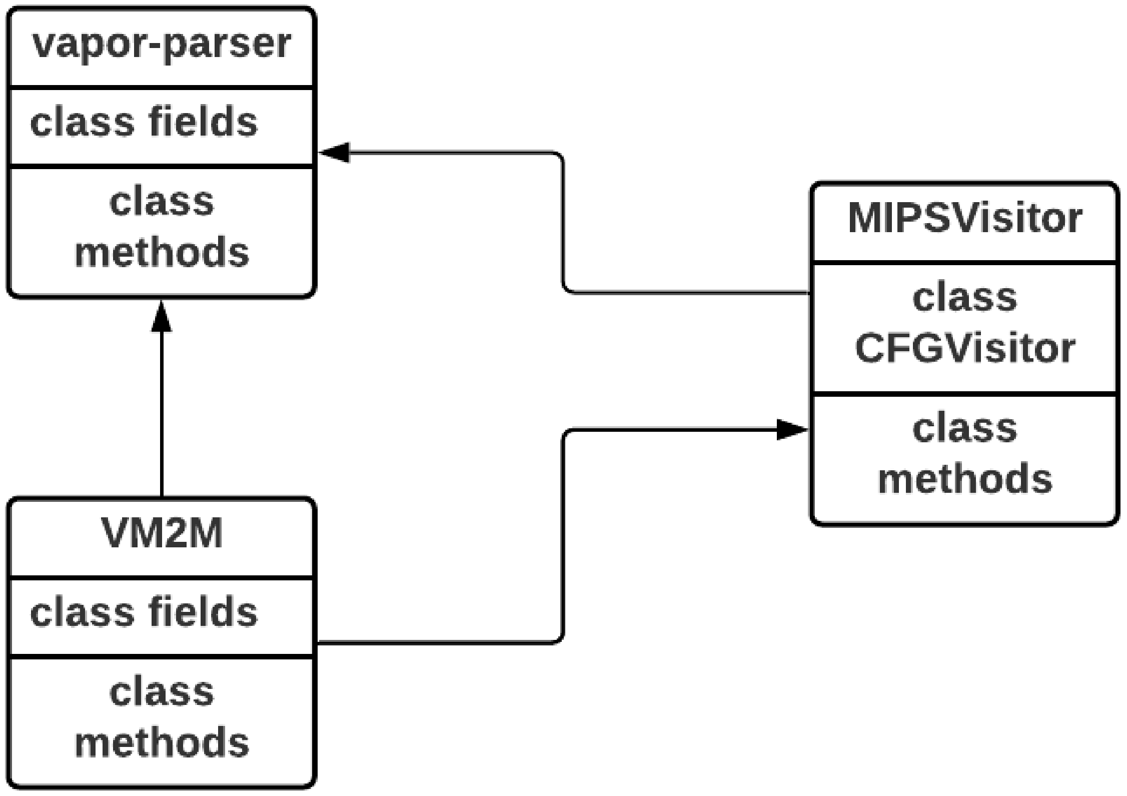


```java
public void visit(VReturn v) {
    String currLine = "";
    int pos = getPos(v.sourcePos.line);
    boolean b = v.value != null;
    boolean b1 = v.value instanceof VOperand.Static;
    if (b) {
        boolean b2 = v.value instanceof VVarRef;
        if (b2) {
            String VALUE = v.value.toString();
            currLine += INDENT + "move $v0 " + VALUE + "\n";
        }
        else if (b1) {
            String VALUE = v.value.toString();
            currLine += INDENT + "li $v0 " + VALUE + "\n";
        }
    }

    currLine += INDENT + "lw $ra -4($fp)\n";
    currLine += INDENT + "lw $fp -8($fp)\n";
    currLine += INDENT + "addu $sp $sp " + spSize + "\n";

    currLine += INDENT + "jr $ra";

    appendLine(pos, currLine);
}
```

Visit Method for Return in a Class' Method

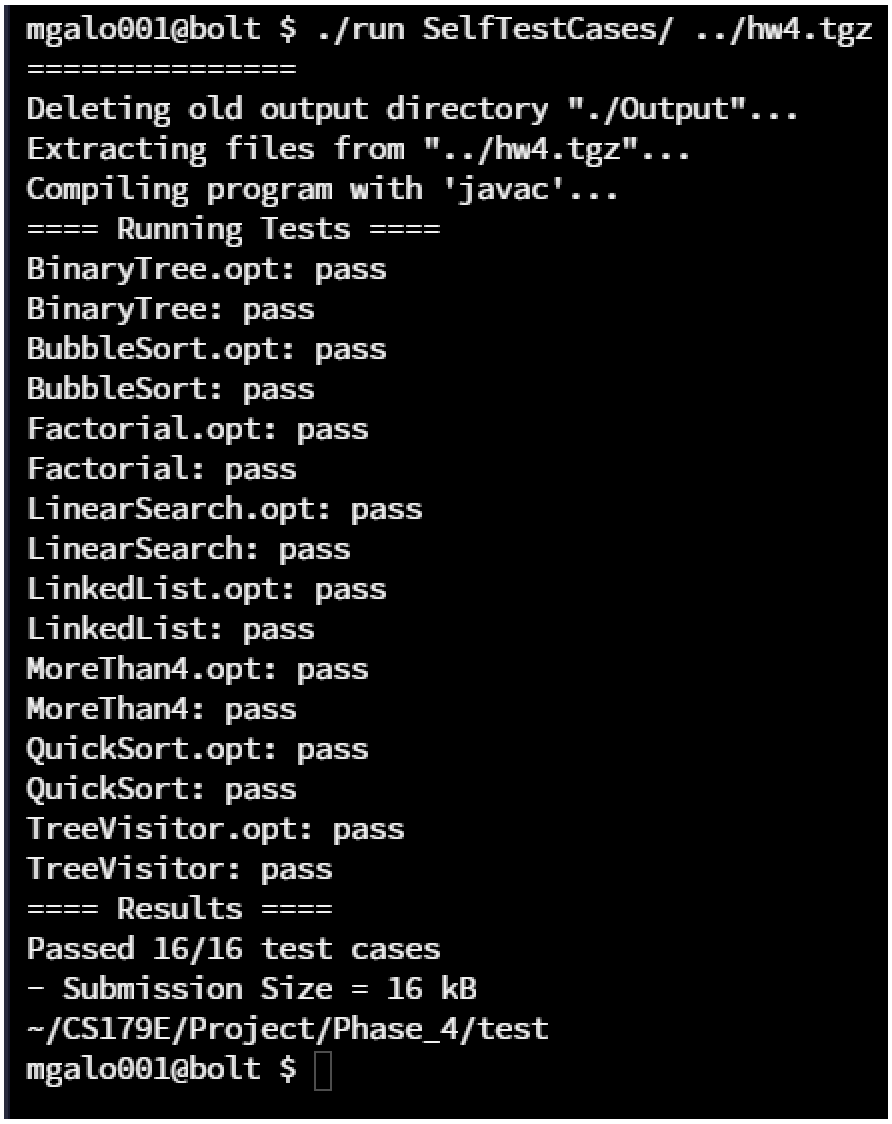
Once the instance of the return operand is checked, here is where we translate the MIPS instructions to pop all the register data and pointers in the stack and revert back to their original position. From their original positions, we can read for the next method of the class.

b. UML Diagram



3. Testing and Verification

For testing, like the last phase, we used the outputs for Factorial and BinaryTree as base case and stress test, respectively. The trickiest part of this phase would be instance checking each vapor class and determining the MIPS instruction such as VReturn, VBuiltIn, VMemRead, and VMemWrite.



```
mgalo001@bolt $ ./run SelfTestCases/ ../hw4.tgz
===============
Deleting old output directory "./Output"...
Extracting files from "../hw4.tgz"...
Compiling program with 'javac'...
==== Running Tests ====
BinaryTree.opt: pass
BinaryTree: pass
BubbleSort.opt: pass
BubbleSort: pass
Factorial.opt: pass
Factorial: pass
LinearSearch.opt: pass
LinearSearch: pass
LinkedList.opt: pass
LinkedList: pass
MoreThan4.opt: pass
MoreThan4: pass
QuickSort.opt: pass
QuickSort: pass
TreeVisitor.opt: pass
TreeVisitor: pass
==== Results ====
Passed 16/16 test cases
- Submission Size = 16 kB
~/CS179E/Project/Phase_4/test
mgalo001@bolt $
```