1.Main

```
class Main {
  public static void main(String[] args) {
        String fileName = "StaticTest.vm";
        VMTranslator VMTobj = new VMTranslator(fileName);
        VMTobj.translate();
  }
}
```

2.VMTranslator

```
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class VMTranslator {

        private static CodeWriter codeWriter;
        private static Parser parser;
        private String fileName;

        //constructor
        public VMTranslator(String inputFile){
        this.fileName = inputFile;
        }

        public void translate(){
        try{
        String outputFileName = fileName.substring(0, fileName.indexOf(".")) + ".asm";
        BufferedReader inputFile = new BufferedReader(new FileReader(fileName));
        codeWriter = new CodeWriter(outputFileName);
        String line;
        while ((line = inputFile.readLine()) != null) {
                parser = new Parser(line);
                switch (parser.commandType()) {
                case C_PUSH:
                case C_POP:
```

```
                codeWriter.writePushPop(parser.commandType(), parser.arg1(), parser.arg2());
                break;
                case C_ARITHMETIC:
                codeWriter.writeArithmetic(parser.arg1());
                break;
                case C_COMMENT:
                break;
                }
        }
        inputFile.close();
        codeWriter.close();
        } catch (final IOException ioe) {
        System.out.println(ioe);
        return;
        }
        }
}
```

3.Parser

```
public class Parser {

        private String currentLine;
        private String currentCommand;

        //constructor
        public Parser(String line) {
        this.currentLine = line;
        }

        public CommandType commandType() {
        currentCommand = currentLine;
        //only gathers info before the space " "
        String command = currentCommand.split(" ")[0];

        //return command types
        if (currentCommand.contains("/")){
        return CommandType.C_COMMENT;
        }
        switch (command) {
        case "push":
```

```
                return CommandType.C_PUSH;
        case "pop":
                return CommandType.C_POP;
        default:
                return CommandType.C_ARITHMETIC;
        }
        }

        public String arg1() {
        if (commandType() == CommandType.C_ARITHMETIC) {
        return currentCommand;
        }
        return currentCommand.split(" ")[1];
        }

        //returns index value
        public int arg2() {
        if (commandType() == CommandType.C_PUSH
                || commandType() == CommandType.C_POP
                || commandType() == CommandType.C_FUNCTION
                || commandType() == CommandType.C_CALL) {
        return Integer.valueOf(currentCommand.split(" ")[2]);
        }
        return 0;
        }
}
```

4.CodeWriter.java

```
import java.io.IOException;
import java.io.PrintWriter;

public class CodeWriter {

        private PrintWriter output;
        private final String fileName;

        private int labelCount = 0;

        //constructor
        public CodeWriter(String outputFileName) {
        output = null;
```

```
fileName = outputFileName;
try {
PrintWriter outputFile = new PrintWriter(outputFileName);
output = new PrintWriter(outputFile);
} catch (final IOException ioe) {
System.out.println(ioe);
return;
}
}

public void writePushPop(CommandType commandType, String segment, int index) {
switch (commandType) {
case C_PUSH:
        output.printf("// push %s %d\n", segment, index);
        switch (segment) {
        case "constant":
        output.println("@"+index);
        output.println("D=A");
        break;
        case "local":
        loadSegment("LCL", index);
        output.println("D=M");
        break;
        case "argument":
        loadSegment("ARG", index);
        output.println("D=M");
        break;
        case "this":
        loadSegment("THIS", index);
        output.println("D=M");
        break;
        case "that":
        loadSegment("THAT", index);
        output.println("D=M");
        break;
        case "pointer":
        output.println("@R"+ String.valueOf(3 + index));
        output.println("D=M");
        break;
        case "temp":
        output.println("@R"+ String.valueOf(5 + index));
        output.println("D=M");
        break;
        case "static":
```

```
            output.println("@"+ fileName.split("\\.")[0]+String.valueOf(index));
            output.println("D=M");
            }
            pushDToStack();
            break;
    case C_POP:
            output.printf("// pop %s %d\n", segment, index);
            switch (segment) {
            case "constant":
            output.println("@"+index);
            break;
            case "local":
            loadSegment("LCL", index);
            break;
            case "argument":
            loadSegment("ARG", index);
            break;
            case "this":
            loadSegment("THIS", index);
            break;
            case "that":
            loadSegment("THAT", index);
            break;
            case "pointer":
            output.println("@R"+ String.valueOf(3 + index));
            break;
            case "temp":
            output.println("@R"+ String.valueOf(5 + index));
            break;
            case "static":
            output.println("@"+ fileName.split("\\.")[0]+String.valueOf(index));
            break;
            }
            output.println("D=A");
            output.println("@R13");
            output.println("M=D");
            popStackToD();
            output.println("@R13");
            output.println("A=M");
            output.println("M=D");
            break;
    }
    }
```

```
public void writeArithmetic(String command) {
output.printf("// %s\n", command);
switch (command) {
case "add":
        popStackToD();
        decrementStackPointer();
        loadStackPointerToA();
        output.println("M=D+M");
        incrementStackPointer();
        break;
case "sub":
        popStackToD();
        decrementStackPointer();
        loadStackPointerToA();
        output.println("M=M-D");
        incrementStackPointer();
        break;
case "neg":
        decrementStackPointer();
        loadStackPointerToA();
        output.println("M=-M");
        incrementStackPointer();
        break;
case "eq":
        writeCompareLogic("JEQ");
        break;
case "gt":
        writeCompareLogic("JGT");
        break;
case "lt":
        writeCompareLogic("JLT");
        break;
case "and":
        popStackToD();
        decrementStackPointer();
        loadStackPointerToA();
        output.println("M=D&M");
        incrementStackPointer();
        break;
case "or":
        popStackToD();
        decrementStackPointer();
        loadStackPointerToA();
        output.println("M=D|M");
```

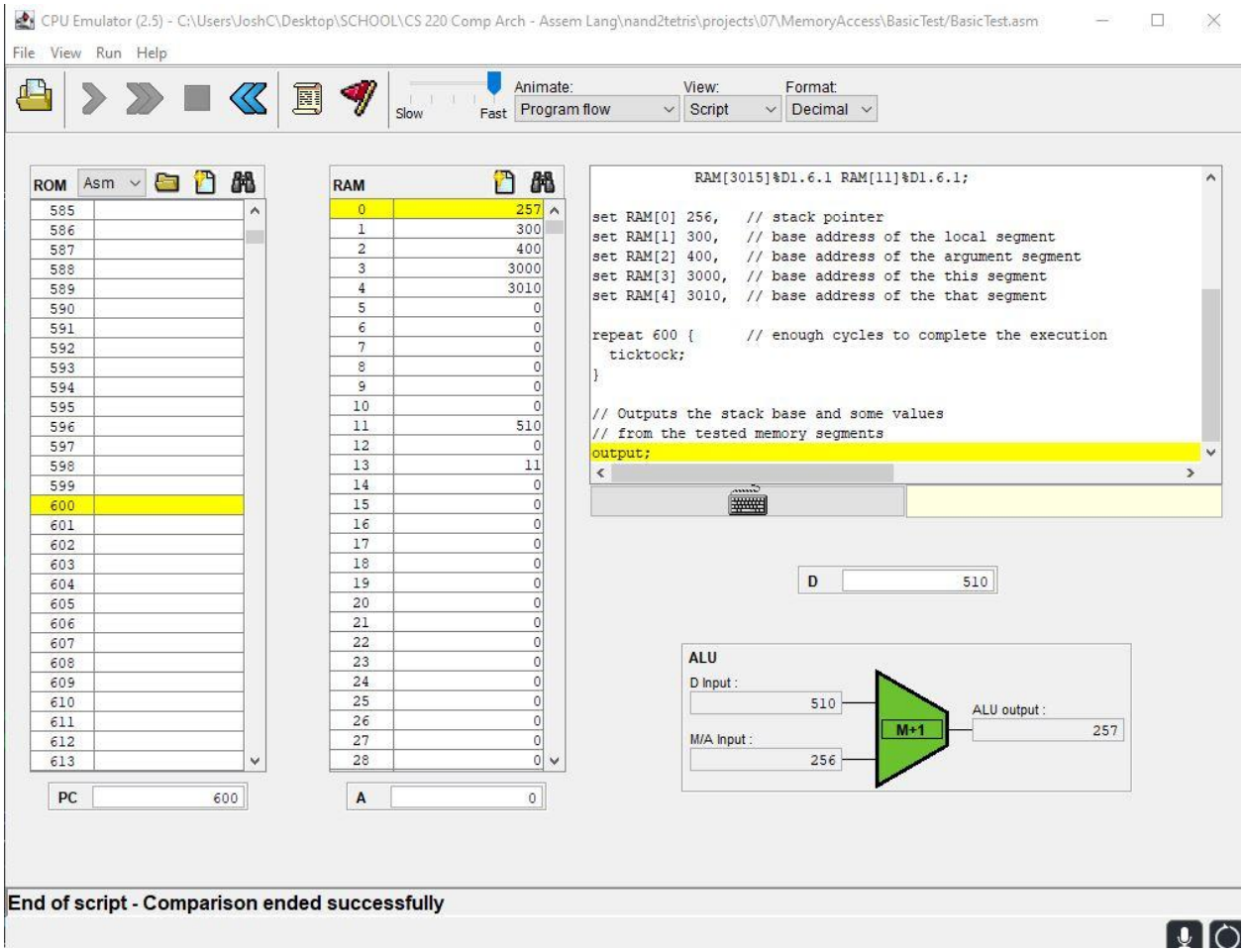
```
            incrementStackPointer();
            break;
case "not":
            decrementStackPointer();
            loadStackPointerToA();
            output.println("M=!M");
            incrementStackPointer();
            break;
}
}

private void incrementStackPointer() {
output.println("@SP");
output.println("M=M+1");
}

private void decrementStackPointer() {
output.println("@SP");
output.println("M=M-1");
}

private void popStackToD() {
decrementStackPointer();
output.println("A=M");
output.println("D=M");
}

private void pushDToStack() {
loadStackPointerToA();
output.println("M=D");
incrementStackPointer();
}

private void loadStackPointerToA() {
output.println("@SP");
output.println("A=M");
}

private void writeCompareLogic(String jumpCommand) {
popStackToD();
decrementStackPointer();
loadStackPointerToA();
output.println("D=M-D");
output.println("@LABEL" + labelCount);
```

```
        output.println("D;"+jumpCommand);
        loadStackPointerToA();
        output.println("M=0");
        output.println("@ENDLABEL" + labelCount);
        output.println("0;JMP");
        output.println("(LABEL" + labelCount + ")");
        loadStackPointerToA();
        output.println("M=-1");
        output.println("(ENDLABEL" + labelCount + ")");
        incrementStackPointer();
        labelCount++;
        }

        private void loadSegment(String segment, int index) {
        output.println("@" + segment);
        output.println("D=M");
        output.println("@"+String.valueOf(index));
        output.println("A=D+A");
        }

        public void close() {
        output.close();
        }
}
```

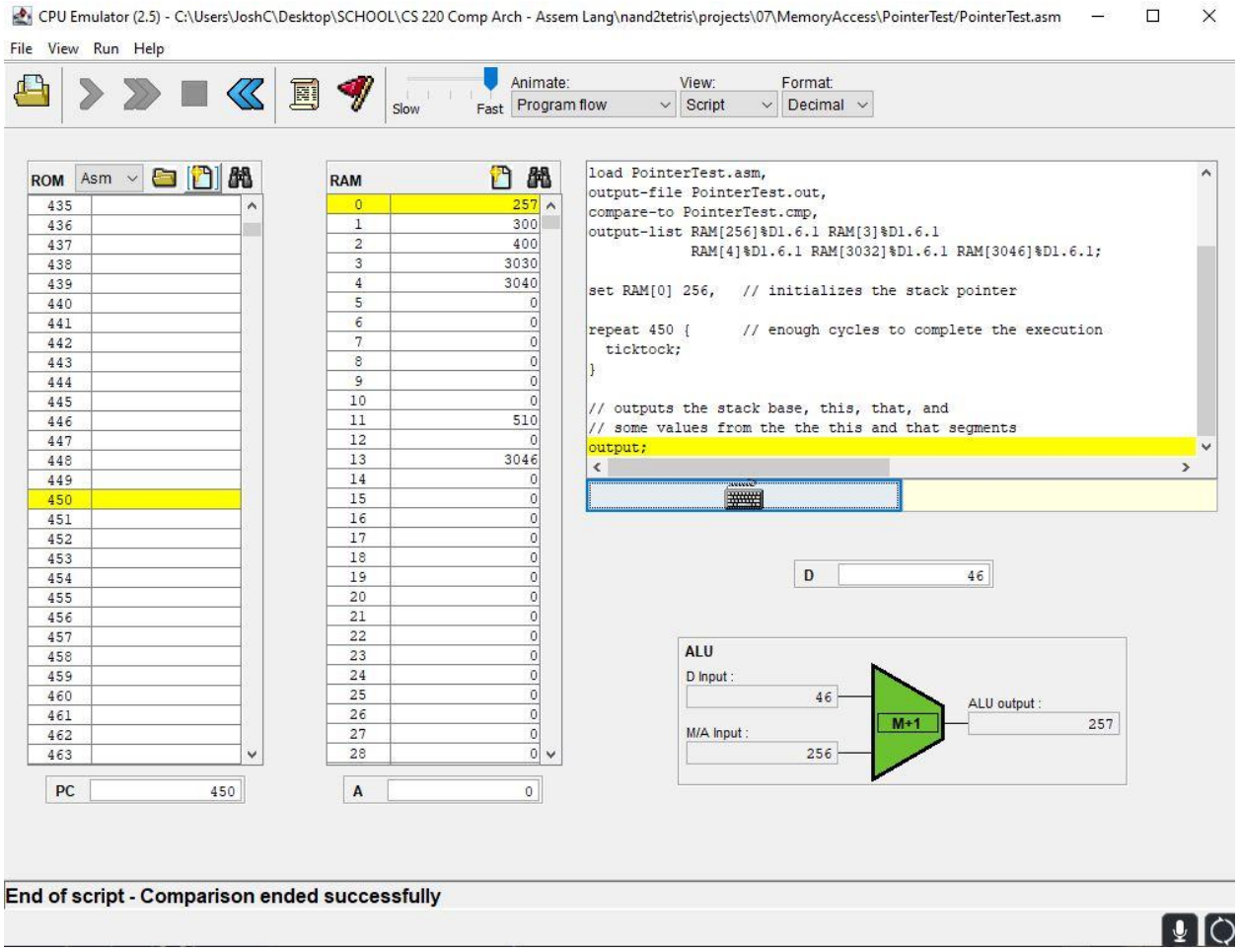
5.CommandType

```
public enum CommandType {
        C_ARITHMETIC,
        C_PUSH,
        C_POP,
        C_LABEL,
        C_GOTO,
        C_IF,
        C_FUNCTION,
        C_RETURN,
        C_CALL,
        C_COMMENT
}
```

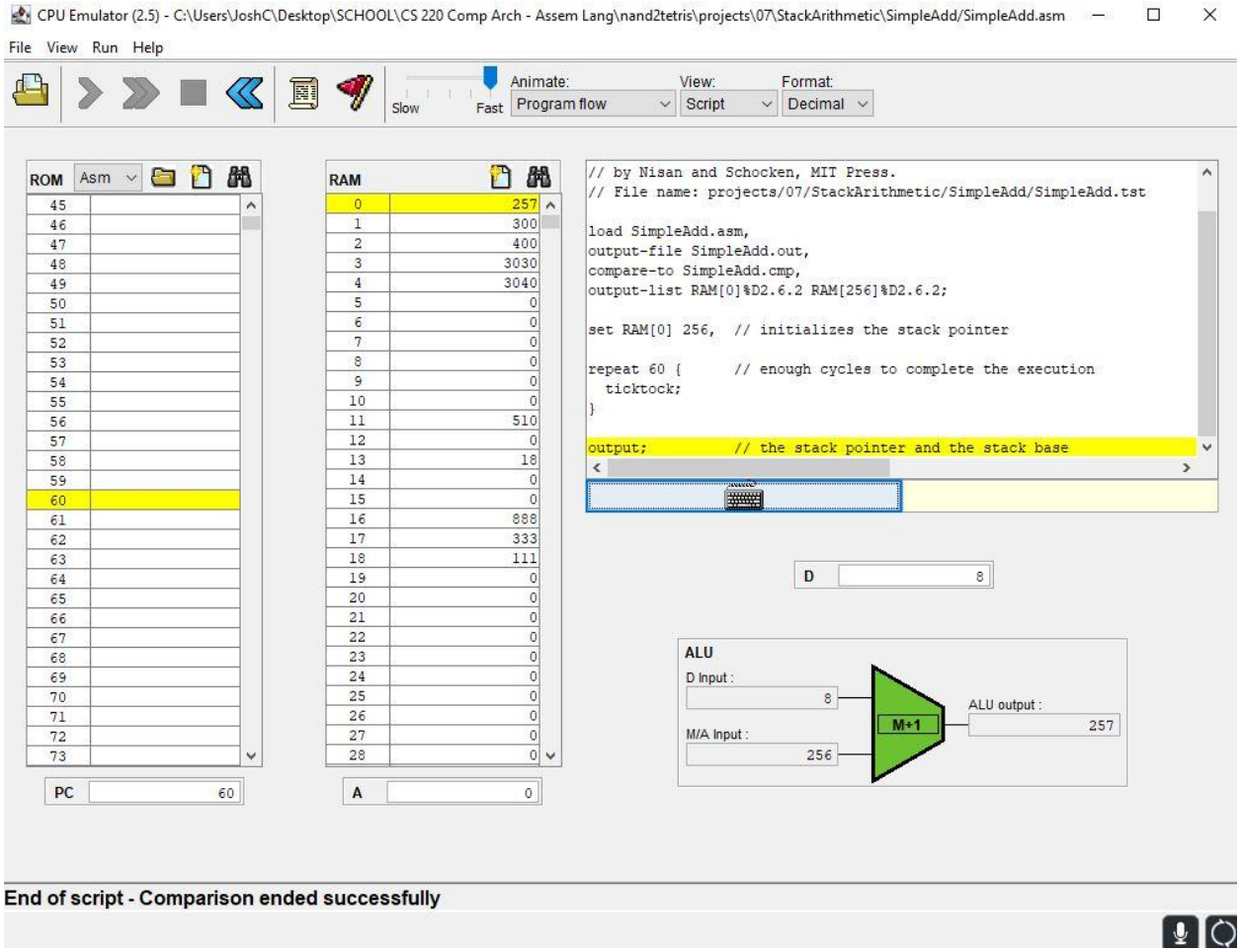
6.BasicTest screenshot comparison



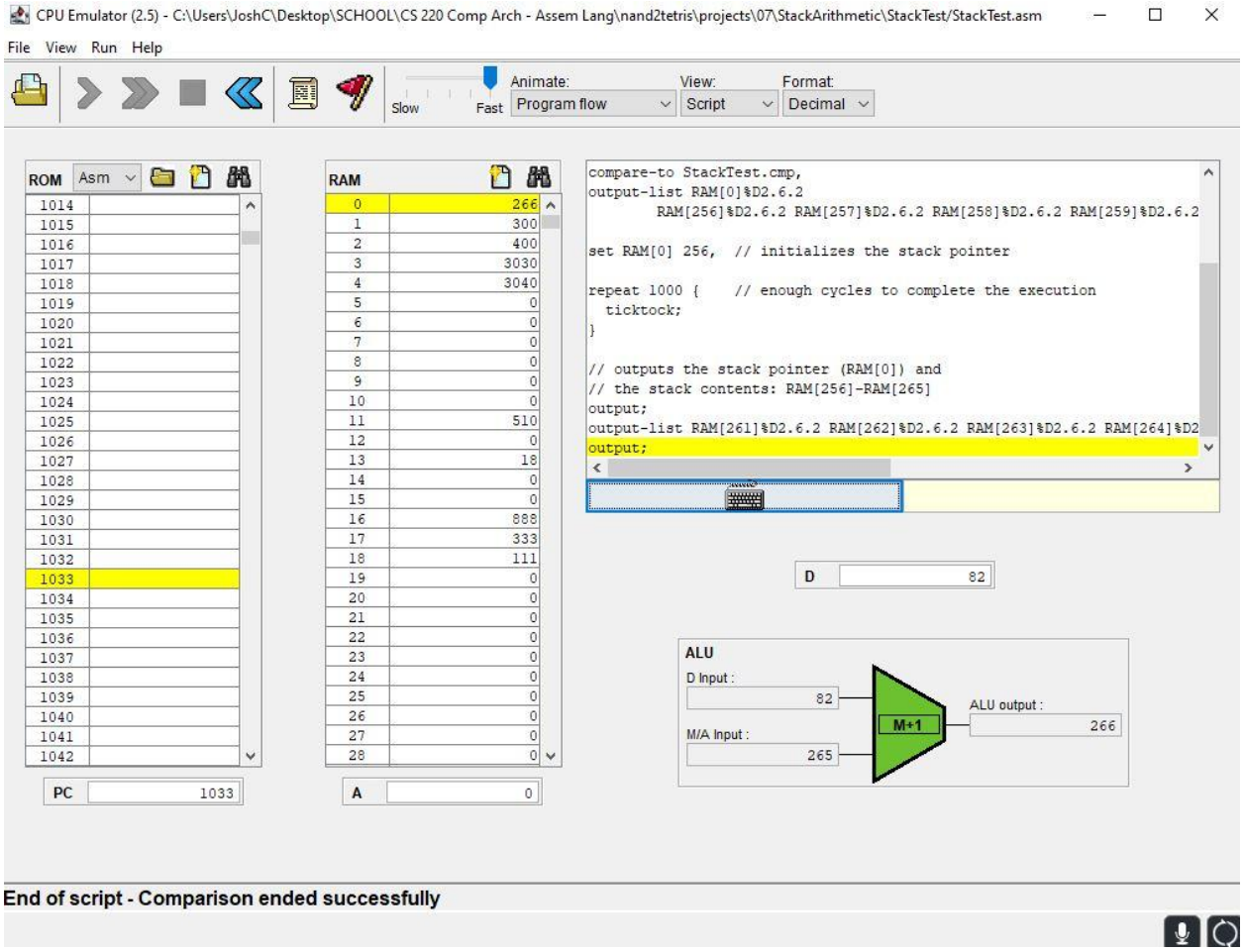
7.PointerTest screenshot comparison



8.SimpleAdd screenshot comparison



9.StackTest screenshot comparison



10.StaticTest screenshot comparison