

## 1.Main

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

CPU Emulator (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\nand2tetris\projects\07\MemoryAccess\BasicTest\BasicTest.asm

File View Run Help

Animate: Program flow View: Script Format: Decimal

ROM Asm

585	
586	
587	
588	
589	
590	
591	
592	
593	
594	
595	
596	
597	
598	
599	
600	
601	
602	
603	
604	
605	
606	
607	
608	
609	
610	
611	
612	
613	

PC 600

RAM

0	257
1	300
2	400
3	3000
4	3010
5	0
6	0
7	0
8	0
9	0
10	0
11	510
12	0
13	11
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 0

```
RAM[3015]%D1.6.1 RAM[11]%D1.6.1;

set RAM[0] 256, // stack pointer
set RAM[1] 300, // base address of the local segment
set RAM[2] 400, // base address of the argument segment
set RAM[3] 3000, // base address of the this segment
set RAM[4] 3010, // base address of the that segment

repeat 600 { // enough cycles to complete the execution
    ticktock;
}

// Outputs the stack base and some values
// from the tested memory segments
output;
```

D 510

ALU

D Input: 510

M/A Input: 256

ALU output: 257

End of script - Comparison ended successfully

## 7.PointerTest screenshot comparison

CPU Emulator (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\nand2tetris\projects\07\MemoryAccess\PointerTest\PointerTest.asm

File View Run Help

Animate: Program flow View: Script Format: Decimal

ROM Asm

435	
436	
437	
438	
439	
440	
441	
442	
443	
444	
445	
446	
447	
448	
449	
450	
451	
452	
453	
454	
455	
456	
457	
458	
459	
460	
461	
462	
463	

PC 450

RAM

0	257
1	300
2	400
3	3030
4	3040
5	0
6	0
7	0
8	0
9	0
10	0
11	510
12	0
13	3046
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 0

```
load PointerTest.asm,
output-file PointerTest.out,
compare-to PointerTest.cmp,
output-list RAM[256]%D1.6.1 RAM[3]%D1.6.1
          RAM[4]%D1.6.1 RAM[3032]%D1.6.1 RAM[3046]%D1.6.1;

set RAM[0] 256, // initializes the stack pointer

repeat 450 { // enough cycles to complete the execution
    ticktock;
}

// outputs the stack base, this, that, and
// some values from the the this and that segments
output;
```

D 46

ALU

D Input: 46

M/A Input: 256

ALU output: 257

End of script - Comparison ended successfully

## 8.SimpleAdd screenshot comparison

CPU Emulator (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\nand2tetris\projects\07\StackArithmetic\SimpleAdd\SimpleAdd.asm

File View Run Help

Animate: Program flow View: Script Format: Decimal

ROM Asm

45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	

PC 60

RAM

0	257
1	300
2	400
3	3030
4	3040
5	0
6	0
7	0
8	0
9	0
10	0
11	510
12	0
13	18
14	0
15	0
16	888
17	333
18	111
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 0

```
// by Nisan and Schocken, MIT Press.
// File name: projects/07/StackArithmetic/SimpleAdd/SimpleAdd.tst

load SimpleAdd.asm,
output-file SimpleAdd.out,
compare-to SimpleAdd.cmp,
output-list RAM[0]%D2.6.2 RAM[256]%D2.6.2;

set RAM[0] 256, // initializes the stack pointer

repeat 60 { // enough cycles to complete the execution
    ticktock;
}

output; // the stack pointer and the stack base
```

D 8

ALU

D Input: 8

M/A Input: 256

ALU output: 257

End of script - Comparison ended successfully

## 9.StackTest screenshot comparison

CPU Emulator (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\nand2tetris\projects\07\StackArithmetic\StackTest\StackTest.asm

File View Run Help

Slow Fast Program flow Script Decimal

ROM Asm

1014	
1015	
1016	
1017	
1018	
1019	
1020	
1021	
1022	
1023	
1024	
1025	
1026	
1027	
1028	
1029	
1030	
1031	
1032	
1033	
1034	
1035	
1036	
1037	
1038	
1039	
1040	
1041	
1042	

PC 1033

RAM

0	266
1	300
2	400
3	3030
4	3040
5	0
6	0
7	0
8	0
9	0
10	0
11	510
12	0
13	18
14	0
15	0
16	888
17	333
18	111
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 0

```
compare-to StackTest.cmp,
output-list RAM[0]%D2.6.2
RAM[256]%D2.6.2 RAM[257]%D2.6.2 RAM[258]%D2.6.2 RAM[259]%D2.6.2

set RAM[0] 256, // initializes the stack pointer

repeat 1000 { // enough cycles to complete the execution
ticktock;
}

// outputs the stack pointer (RAM[0]) and
// the stack contents: RAM[256]-RAM[265]
output;
output-list RAM[261]%D2.6.2 RAM[262]%D2.6.2 RAM[263]%D2.6.2 RAM[264]%D2.6.2
output;
```

D 82

ALU

D Input: 82

M/A Input: 265

ALU output: 266

End of script - Comparison ended successfully

## 10.StaticTest screenshot comparison

CPU Emulator (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\nand2tetris\projects\07\MemoryAccess\StaticTest\StaticTest.asm

File View Run Help

Animate: Program flow View: Script Format: Decimal

ROM Asm

180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	

PC 200

RAM

0	257
1	300
2	400
3	3030
4	3040
5	0
6	0
7	0
8	0
9	0
10	0
11	510
12	0
13	18
14	0
15	0
16	888
17	333
18	111
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 0

```
// by Nisan and Schocken, MIT Press.
// File name: projects/07/MemoryAccess/StaticTest/StaticTest.tst

load StaticTest.asm,
output-file StaticTest.out,
compare-to StaticTest.cmp,
output-list RAM[256]%D1.6.1;

set RAM[0] 256, // initializes the stack pointer

repeat 200 { // enough cycles to complete the execution
    ticktock;
}

output; // the stack base
```

D 888

ALU

D Input: 888

M/A Input: 256

ALU output: 257

End of script - Comparison ended successfully