Contents

1 README

```
roigreenberg,inbaravni

================================================================================

Roi Greenberg, ID 30557123, roi.greenberg@mail.huji.ac.il
Inbar Avni, ID 201131760, inbar.avni@mail.huji.ac.il

================================================================================



                            Project 6 - The Assembler

                            ------------------------




Submitted Files

---------------


Parser.java - a module that reads an assembly language command,
              parses it, and provides convenient access to the commands
              components.

Code.java - a module that translates Hack assembly language mnemonics into
            binary codes.

SymbolTable - a module that keeps a correspondence between symbolic labels
             and numeric addresses.

Assembler.java - the main part of our program, resposiple to run all parts of the
            assembler together.



Makefile.
Assembler - a script for execute the correct command.
README - This file.


Remarks

-------

* No remarks for that time.
```

2 Assembler

```
1  #!/bin/sh
2  java Assembler $1
```

3 Assembler.java

```
1  import java.io.BufferedWriter;
2  import java.io.File;
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.util.Arrays;
6  import java.util.regex.*;
7
8  public class Assembler {
9      public static void findSymbols(File input, SymbolTable st)
10     {
11         Parser p = new Parser(input);
12         String s = new String();
13         int index = 0;
14         while (p.hasMoreCommands())
15         {
16             p.advance();
17             switch (p.commandType()){
18                 case A_COMMAND:
19                     index++;
20                     break;
21                 case C_COMMAND:
22                     index++;
23                     break;
24
25                 case L_COMMAND:
26                     s = p.symbol();
27                     st.addEntry(s, index);
28                     break;
29             }
30
31
32         }
33     }
34     public static void asmToHack(File input, SymbolTable st)
35     {
36         Parser p = new Parser(input);
37         String fileName = input.getPath().split("\\.")[0];
38         Code b = new Code();
39
40         String s = new String();
41         char[] c = new char[16];
42         int varIndex = 16;
43         File outputFile = new File(fileName+".hack");
44         if (!outputFile.exists()) {
45             try {
46                 outputFile.createNewFile();
47             } catch (IOException e) {
48                 e.printStackTrace();
49             }
50         }
51
52         FileWriter fw = null;
53         try {
54             fw = new FileWriter(outputFile.getAbsoluteFile());
55         } catch (IOException e) {
56             e.printStackTrace();
57         }
58         BufferedWriter bw = new BufferedWriter(fw);
59
```

```
60          while (p.hasMoreCommands())
61          {
62              p.advance();
63              Arrays.fill(c, '0');
64              switch (p.commandType()){
65                  case A_COMMAND:
66                      s = p.symbol();
67                      if (isInteger(s))
68                      {
69                          s = Integer.toBinaryString(Integer.parseInt(s));
70                          fillChar(c, s, 16 - s.length());
71                      } else {
72                          if (!st.contains(s))
73                          {
74                              st.addEntry(s, varIndex++);
75                          }
76                          s = Integer.toBinaryString(st.getAddress(s));
77                          fillChar(c, s, 16 - s.length());                    }
78                      break;
79                  case L_COMMAND:
80                      continue;
81
82                  case C_COMMAND:
83                      s = b.jump(p.jump());
84                      fillChar(c, s, 13);
85                      s = b.dest(p.dest());
86                      fillChar(c, s, 10);
87                      s = b.comp(p.comp());
88                      fillChar(c, s, 1);
89                      c[0] = '1';
90              }
91              try {
92                  bw.write(c);
93                  bw.newLine();
94              } catch (IOException e) {
95                  e.printStackTrace();
96              }
97
98          }
99          try {
100             bw.close();
101         } catch (IOException e) {
102             e.printStackTrace();
103         }
104
105     }
106
107     public static boolean isInteger(String s) {
108         boolean isValidInteger = false;
109         try
110         {
111             Integer.parseInt(s);
112
113             // s is a valid integer
114
115             isValidInteger = true;
116         }
117         catch (NumberFormatException ex)
118         {
119             // s is not an integer
120         }
121
122         return isValidInteger;
123     }
124
125     public static void fillChar(char[] dst, String src, int startIndex)
126     {
127         for (int i = 0; i < src.length() && i < 16; i++)
```

```
128             {
129                 dst[startIndex + i] = src.charAt(i);
130             }
131         }
132 
133         public static void main(String[] args) {
134 
135             File input = new File(args[0]);
136             if (input.isFile())
137             {
138 
139                 if (input.getPath().split("\\.")[1].equals("asm"))
140                 {
141                     SymbolTable st = new SymbolTable();
142                     findSymbols(input, st);
143                     asmToHack(input, st);
144                 }
145 
146             }
147             else if(input.isDirectory())
148             {
149                 for (File file : input.listFiles())
150                 {
151                     if (file.getPath().split("\\.")[1].equals("asm"))
152                     {
153                         SymbolTable st = new SymbolTable();
154                         findSymbols(file, st);
155                         asmToHack(file, st);
156 
157                     }
158                 }
159             }
160         }
161     }
```

4 Code.java

```java
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by inbaravni on 3/29/16.
 */
public class Code {

    private Pattern destPattern = Pattern.compile("(A)?(M)?(D)?");
    private Matcher codeMatcher;
    private char[] destBit = new char[3];
    private Map<String, String> compMap = new HashMap<String, String>();
    private Map<String, String> jumpMap = new HashMap<String, String>();

    Code(){
        // init the jumpMap
        jumpMap.put(null, "000");
        jumpMap.put("JGT", "001");
        jumpMap.put("JEQ", "010");
        jumpMap.put("JGE", "011");
        jumpMap.put("JLT", "100");
        jumpMap.put("JNE", "101");
        jumpMap.put("JLE", "110");
        jumpMap.put("JMP", "111");

        // init the compMap
        compMap.put("0", "110101010");
        compMap.put("1", "110111111");
        compMap.put("-1", "110111010");
        compMap.put("D", "110001100");
        compMap.put("A", "110110000");
        compMap.put("!D", "110001101");
        compMap.put("!A", "110110001");
        compMap.put("-D", "110001111");
        compMap.put("-A", "110110011");
        compMap.put("D+1", "110011111");
        compMap.put("A+1", "110110111");
        compMap.put("D-1", "110001110");
        compMap.put("A-1", "110110010");
        compMap.put("D+A", "110000010");
        compMap.put("D-A", "110010011");
        compMap.put("A-D", "110000111");
        compMap.put("D&A", "110000000");
        compMap.put("D|A", "110010101");
        compMap.put("M", "111110000");
        compMap.put("!M", "111110001");
        compMap.put("-M", "111110011");
        compMap.put("M+1", "111110111");
        compMap.put("M-1", "111110010");
        compMap.put("D+M", "111000010");
        compMap.put("D-M", "111010011");
        compMap.put("M-D", "111000111");
        compMap.put("D&M", "111000000");
        compMap.put("D|M", "111010101");
        compMap.put("D<<", "010110000");
        compMap.put("D>>", "010010000");
```

```
60          compMap.put("A<<", "010100000");
61          compMap.put("A>>", "010000000");
62          compMap.put("M<<", "011100000");
63          compMap.put("M>>", "011000000");
64
65      }
66
67      // Returns the binary code of the dest mnemonic.
68      String dest(String mnemonic) {
69          Arrays.fill(destBit, '0');
70          if (mnemonic == null)
71              return new String(destBit);
72          codeMatcher = destPattern.matcher(mnemonic);
73          codeMatcher.find();
74
75
76              if (codeMatcher.group(1) != null)
77                  destBit[0] = '1';
78              if (codeMatcher.group(2) != null)
79                  destBit[2] = '1';
80              if (codeMatcher.group(3) != null)
81                  destBit[1] = '1';
82
83          return new String(destBit);
84      }
85
86
87      // Returns the binary code of the jump mnemonic.
88      String jump(String mnemonic) {
89
90          return jumpMap.get(mnemonic);
91      }
92
93      // Returns the binary code of the comp mnemonic.
94      String comp(String mnemonic) {
95
96          return compMap.get(mnemonic);
97      }
98  }
```

5 Makefile

```
1  ###############################################################################
2  #
3  # Makefile for Java project
4  #
5  # Students:
6  # Roi Greenberg, ID 30557123, roi.greenberg@mail.huji.ac.il
7  # Inbar Avni, ID 201131760, inbar.avni@mail.huji.ac.il
8  #
9  ###############################################################################
10
11 JAVAC=javac
12 JAVACFLAGS=-g
13
14 SRCS=*.java
15 EXEC=Assembler
16
17 TAR=tar
18 TARFLAGS=cvf
19 TARNAME=project6.tar
20 TARSRCS=$(SRCS) $(EXEC) README Makefile
21
22 all: compile
23
24 compile:
25     $(JAVAC) $(JAVACFLAGS) $(SRCS)
26     chmod +x $(EXEC)
27
28 tar:
29     $(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
30
31 clean:
32     rm -f *.class *~
```

6 Parser.java

```java
/**
 * Created by inbaravni on 3/29/16.
 */


import java.nio.*;
import java.nio.file.*;
import java.io.*;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class Parser {

    private File inputFile;
    private BufferedReader bufReader;
    private boolean toAdvance = true;
    private String currentLine = "";
    private String nextLine = "";
    private Boolean MoreCommands;
    public enum Command {A_COMMAND, C_COMMAND, L_COMMAND};
    private Pattern cCommandPattern = Pattern.compile("(([^=/]*)=)?([^/=;]*)(;([^/=;]*))?(//.*)?");
    private Matcher matcher;

    // Constructor
    // opens the input file/stream and gets ready to parse it.
    Parser(File inputFile) {

        try {
            this.inputFile = inputFile;
            FileReader fileReader = new FileReader(inputFile.getAbsolutePath());
            this.bufReader = new BufferedReader(fileReader);
            if (this.bufReader == null)
            {
                System.out.println("No file");
            }
        } catch (IOException ioexc) {
            //throw new IOException("Problem reading the file");
        }
    }


    // are there more commands in the input?
    Boolean hasMoreCommands() {

        if (toAdvance) {
            toAdvance = false;
            try {
                while ((nextLine = this.bufReader.readLine()).trim().isEmpty() || nextLine.trim().charAt(0) == '/') {
                }
                if (nextLine != null) {
                    this.MoreCommands = true;
                    return true;
                }

                this.MoreCommands = false;
                return false;

```

```
            } catch  (Exception e) {
                this.MoreCommands = false;
                return false;
            }
        } else {
            return this.MoreCommands;
        }
    }

    // the next command from the input and makes it the current
    // command. Should be called only if hasMoreCommands() is true.
    void advance() {

        if (this.hasMoreCommands()) {
            currentLine = nextLine.replaceAll("\\s", "");
        }
        toAdvance = true;
    }

    // Returns the type of the current command
    Command commandType() {

        switch (currentLine.charAt(0)) {
            // A_COMMAND
            case '@':
                return Command.A_COMMAND;

            // L_COMMAND
            case '(':
                return Command.L_COMMAND;

            // C_COMMAND
            default:
                return Command.C_COMMAND;
        }
    }

    // Returns the symbol or decimal Xxx of the current command @Xxx or (Xxx).
    String symbol() {


        switch (commandType()) {

            case A_COMMAND:
                return currentLine.split("@")[1];

            case L_COMMAND:
                return currentLine.split("[\\(\\)]")[1];

            default:
                return null;
        }
    }


    // Returns the dest mnemonic in the current C-command
    String dest() {

        matcher = cCommandPattern.matcher(currentLine);
        matcher.find();
        return matcher.group(2);
    }


    // Returns the comp mnemonic in the current C-command
    String comp() {

        matcher = cCommandPattern.matcher(currentLine);
```

```
128             matcher.find();
129             return matcher.group(3);
130         }
131 
132         // Returns the jump mnemonic in the current C-command
133         String jump() {
134 
135             matcher = cCommandPattern.matcher(currentLine);
136             matcher.find();
137             return matcher.group(5);
138         }
139 
140     }
```

7 SymbolTable.java

```java
import java.util.HashMap;
import java.util.Map;

/**
 * Created by roigreenberg on 3/29/16.
 */
public class SymbolTable {
    private Map<String, Integer> symbolTable = new HashMap<String, Integer>();

    SymbolTable()
    {
        symbolTable.put("SP", 0);
        symbolTable.put("LCL", 1);
        symbolTable.put("ARG", 2);
        symbolTable.put("THIS", 3);
        symbolTable.put("THAT", 4);
        symbolTable.put("SCREEN", 16384);
        symbolTable.put("KBD", 24576);
        for (int i = 0; i < 16; i++)
        {
            symbolTable.put("R"+ i, i);

        }
    }
    void addEntry(String symbol, int address)
    {
        symbolTable.put(symbol, address);
    }

    boolean contains(String symbol)
    {
        return symbolTable.containsKey(symbol);
    }

    int getAddress(String symbol)
    {
        return symbolTable.get(symbol);
    }

}
```