Contents

1 Basic Test Results

```
********** TEST START **********

  preparing sub.tar
dos2unix: converting file /tmp/bodek._MLOGg/nand2tet/Project07/roigreenberg/presubmission/testdir/stud/sub.tar/README to Uni
  checking sub.tar
chmod a+x VMtranslator

********** TEST END **********
```

2 README

```
roigreenberg,inbaravni

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

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

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



             Project 7 - Virtual Machine I - Stack Arithmetic

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




Submitted Files

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

README - This file.

VMtranslator.py - the main.
Parser.py - parse the file
codeWriter.py - write the assembly code
VMtranslator - An executable.
Makefile - a makefile.

Remarks

-------

* No remarks for that time.
```

3 Makefile

```
1  ##############################################################################
2  #
3  # Makefile for Python 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
12
13 SRCS=*.py
14 EXEC=VMtranslator
15
16 TAR=tar
17 TARFLAGS=cvf
18 TARNAME=project7.tar
19 TARSRCS=$(SRCS) $(EXEC) README Makefile
20
21 all:
22     chmod a+x $(EXEC)
23 tar:
24     $(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
25
26 clean:
27     rm -f *~
```

4 Parser.py

```
__author__ = 'inbaravni'

import os


class Command:
    C_ARITHMETIC = 0
    C_PUSH = 1
    C_POP = 2
    C_LABEL = 3
    C_GOTO = 4
    C_IF = 5
    C_FUNCTION = 6
    C_RETURN = 7
    C_CALL = 8

class Parser:

   arithmetic = ['add','sub','neg','eq','gt','lt','and','or','not'];


   def __init__(self, input_file):

      self.inputFile = input_file;
      self.file = open(input_file, "r");
      #self.name = os.path.abspath(input_file).split('.')[0];
      self.name = (input_file).split('.')[0]
      self.curLine = '';
      self.nextLine = '';

   def hasMoreCommands(self):

       while True:
           self.nextLine = self.file.readline();
           # EOF case
           if not self.nextLine:
               return False;
           self.nextLine = self.nextLine.strip().split('/')[0]
           # the line is only whitespaces or a comment
           if not (self.nextLine) or (self.nextLine[0] == '/'):
               continue;
           else:
               return True;


   def advance(self):

       self.curLine = self.nextLine;


   def commandType(self):

       lines_words = self.curLine.split();

       if lines_words[0] in self.arithmetic:
           return Command.C_ARITHMETIC;

       elif (lines_words[0]) == 'push':
           return Command.C_PUSH;
```

```
60
61          elif (lines_words[0]) == 'pop':
62              return Command.C_POP;
63
64
65
66
67      def arg1(self):
68
69          lines_words = self.curLine.split();
70
71          if self.commandType() == Command.C_ARITHMETIC:
72              return lines_words[0];
73
74          elif (self.commandType() == Command.C_PUSH) or (self.commandType() == Command.C_POP):
75              return lines_words[1];
76
77
78
79
80      def arg2(self):
81
82          lines_words = self.curLine.split();
83
84          if (self.commandType() == Command.C_PUSH) or (self.commandType() == Command.C_POP):
85              return lines_words[2];
```

5 VMtranslator

```
#!/bin/sh
python3 VMtranslator.py $1
```

6 VMtranslator.py

```
__author__ = 'inbaravni'
from Parser import *
from codeWriter import *
import sys
import os



def main(argv):
    # file given
    if (os.path.isfile(argv[0])):
        parser = Parser(argv[0]);
        w = CodeWriter(parser.name);
        w.setFileName(parser.name.split('/')[-1])


        while (parser.hasMoreCommands()):
            parser.advance();
            # arithmetic
            if parser.commandType() is Command.C_ARITHMETIC:
                w.writeArithmetic(parser.arg1());
            # pop
            if parser.commandType() == Command.C_POP:
                w.writePushPop(Command.C_POP, parser.arg1(), parser.arg2());
            # push
            if parser.commandType() == Command.C_PUSH:
                w.writePushPop(Command.C_PUSH, parser.arg1(), parser.arg2());

        w.close();

    # directory given
    else:
        #path = os.path.abspath(argv[0])+'/'
        path = argv[0]
        if path[-1] != '/':
            path = path+'/'
        name = path + path.split('/')[-2];
        w = CodeWriter(name);

        for each_file in  os.listdir(argv[0]):
            if each_file.endswith(".vm"):

                parser = Parser(path+each_file);
                w.setFileName(parser.name.split('/')[-1])

                while (parser.hasMoreCommands()):
                    parser.advance();
                    # arithmetic
                    if parser.commandType() == Command.C_ARITHMETIC:
                        w.writeArithmetic(parser.arg1());
                    # pop
                    if parser.commandType() == Command.C_POP:
                        w.writePushPop(Command.C_POP, parser.arg1(), parser.arg2());

                    # push
                    if parser.commandType() == Command.C_PUSH:
                        w.writePushPop(Command.C_PUSH, parser.arg1(), parser.arg2());

        w.close();
```

```
60
61
62
63
64
65  if __name__ == "__main__":
66      main(sys.argv[1:])
```

7 codeWriter.py

```
from io import *
from Parser import *


__author__ = 'roigreenberg'


class CodeWriter:

    fileName = ''
    labelIndex = 0

    def __init__(self, fileName):
        self.w = open(fileName + ".asm", "w")
        self.w.write('@256\n')
        self.w.write('D=A\n')
        self.w.write('@SP\n')
        self.w.write('M=D\n')

    def setFileName(self, fileName):
        self.fileName = fileName

    def close(self):

        self.w.close()

    def pop(self):
        self.w.write('@SP\n')
        self.w.write('M=M-1\n')
        self.w.write('A=M\n')
        self.w.write('D=M\n')

    def pop2(self):
        self.pop()
        self.w.write('@R13\n')
        self.w.write('M=D\n')
        self.pop()

    def push(self):
        self.w.write('@SP\n')
        self.w.write('M=M+1\n')
        self.w.write('A=M-1\n')
        self.w.write('M=D\n')

    def compare(self, op):
        self.pop()
        self.w.write('@R14\n')
        self.w.write('M=D\n')
        self.pop()
        self.w.write('@R13\n')
        self.w.write('M=D\n')

        self.w.write('@R13\n')
        self.w.write('D=M\n')
        self.w.write('@Apos'+str(self.labelIndex)+'\n') #if A>=0
        self.w.write('D;JGE\n')
        self.w.write('@Aneg'+str(self.labelIndex)+'\n') #if A<0
        self.w.write('0;JMP\n')
        self.w.write('(Apos'+str(self.labelIndex)+')\n') #A >= 0
```

```
60
61          self.w.write('@R14\n')
62          self.w.write('D=M\n')
63          self.w.write('@AposBpos'+str(self.labelIndex)+'\n') #if A>=0 && B>=0
64          self.w.write('D;JGE\n')
65          self.w.write('@AposBneg'+str(self.labelIndex)+'\n') #if A>=0 && B<0
66          self.w.write('0;JMP\n')
67
68          self.w.write('(AposBpos'+str(self.labelIndex)+')\n') # A >= 0, B >= 0
69          self.w.write('@R13\n')
70          self.w.write('D=M\n')
71          self.w.write('@R14\n')
72          self.w.write('D=D-M\n')
73          self.w.write('@R15\n')
74          self.w.write('M=D\n')
75          self.w.write('@End'+str(self.labelIndex)+'\n') #if A<0 && B<0
76          self.w.write('0;JMP\n')
77
78          self.w.write('(AposBneg'+str(self.labelIndex)+')\n') # A >= 0, B < 0
79          self.w.write('@R15\n')
80          self.w.write('M=1\n')
81          self.w.write('@End'+str(self.labelIndex)+'\n') # end
82          self.w.write('0;JMP\n')
83
84          self.w.write('(Aneg'+str(self.labelIndex)+')\n') #A < 0
85          self.w.write('@R14\n')
86          self.w.write('D=M\n')
87          self.w.write('@AnegBpos'+str(self.labelIndex)+'\n') #if A<0 && B>=0
88          self.w.write('D;JGE\n')
89          self.w.write('@AnegBneg'+str(self.labelIndex)+'\n') #if A<0 && B<0
90          self.w.write('0;JMP\n')
91
92          self.w.write('(AnegBpos'+str(self.labelIndex)+')\n') # A < 0, B >= 0
93          self.w.write('@R15\n')
94          self.w.write('M=-1\n')
95          self.w.write('@End'+str(self.labelIndex)+'\n') # end
96          self.w.write('0;JMP\n')
97
98          self.w.write('(AnegBneg'+str(self.labelIndex)+')\n') # A < 0, B < 0
99          self.w.write('@R13\n')
100         self.w.write('D=M\n')
101         self.w.write('@R14\n')
102         self.w.write('D=D-M\n')
103         self.w.write('@R15\n')
104         self.w.write('M=D\n')
105         self.w.write('@End'+str(self.labelIndex)+'\n') # end
106         self.w.write('0;JMP\n')
107
108         self.w.write('(End'+str(self.labelIndex)+')\n') # end
109
110
111         self.w.write('@R15\n')
112         self.w.write('D=M\n')
113         self.w.write('@True'+str(self.labelIndex)+'\n')
114         self.w.write('D;'+ op +'\n')
115         self.w.write('D=0\n')
116         self.push()
117         self.w.write('@Endcmp'+str(self.labelIndex)+'\n') # end
118         self.w.write('0;JMP\n')
119         self.w.write('(True'+str(self.labelIndex)+')\n')
120         self.w.write('D=-1\n')
121         self.push()
122         self.w.write('(Endcmp'+str(self.labelIndex)+')\n') # end comparing
123
124         self.labelIndex+=1
125
126     def pushFrom(self, seg, i):
127         self.w.write('@'+str(i)+'\n')
```

```
128             if seg != '':
129                 self.w.write('D=A\n')
130                 self.w.write('@'+seg+'\n')
131                 self.w.write('A=D+M\n')
132             self.w.write('D=M\n')
133 
134         def popTo(self, seg, i):
135             if seg != '':
136                 self.w.write('@R13\n')
137                 self.w.write('M=D\n')
138 
139             self.w.write('@'+str(i)+'\n') #find and save memory address
140             if seg != '':
141                 self.w.write('D=A\n')
142                 self.w.write('@'+seg+'\n')
143                 self.w.write('D=D+M\n')
144                 self.w.write('@R14\n')
145                 self.w.write('M=D\n')
146 
147                 self.w.write('@R13\n') #set to  memory address
148                 self.w.write('D=M\n')
149                 self.w.write('@R14\n')
150                 self.w.write('A=M\n')
151             self.w.write('M=D\n')
152 
153         def writeArithmetic(self, command):
154 
155             if command == 'add':
156                 self.pop2()
157                 self.w.write('@R13\n')
158                 self.w.write('D=D+M\n')
159                 self.push()
160             elif command == 'sub':
161                 self.pop2()
162                 self.w.write('@R13\n')
163                 self.w.write('D=D-M\n')
164                 self.push()
165             elif command == 'neg':
166                 self.pop()
167                 self.w.write('D=-D\n')
168                 self.push()
169             elif command == 'eq':
170                 self.compare('JEQ')
171 
172             elif command == 'gt':
173                 self.compare('JGT')
174             elif command == 'lt':
175                 self.compare('JLT')
176             elif command == 'and':
177                 self.pop2()
178                 self.w.write('@R13\n')
179                 self.w.write('D=D&M\n')
180                 self.push()
181             elif command == 'or':
182                 self.pop2()
183                 self.w.write('@R13\n')
184                 self.w.write('D=D|M\n')
185                 self.push()
186             elif command == 'not':
187                 self.pop()
188                 self.w.write('D=!D\n')
189                 self.push()
190 
191         def writePushPop(self, command, segment, index):
192             if command == Command.C_POP:
193                 self.pop()
194                 if segment == 'local':
195                     self.popTo('LCL', index)
```

```
196             elif segment == 'argument':
197                 self.popTo('ARG', index)
198             elif segment == 'this':
199                 self.popTo('THIS', index)
200             elif segment == 'that':
201                 self.popTo('THAT', index)
202             elif segment == 'static':
203                 self.popTo('',self.fileName+'.'+ str(index))
204             elif segment == 'pointer':
205                 self.popTo('',3+int(index))
206             elif segment == 'temp':
207                 self.popTo('',5+int(index))
208         elif command == Command.C_PUSH:
209             if segment == 'constant':
210                 self.w.write("@"+str(index)+"\n")
211                 self.w.write('D=A\n')
212             elif segment == 'local':
213                 self.pushFrom('LCL', index)
214             elif segment == 'argument':
215                 self.pushFrom('ARG', index)
216             elif segment == 'this':
217                 self.pushFrom('THIS', index)
218             elif segment == 'that':
219                 self.pushFrom('THAT', index)
220             elif segment == 'static':
221                 self.pushFrom('',self.fileName+'.'+ str(index))
222             elif segment == 'pointer':
223                 self.pushFrom('',3+int(index))
224             elif segment == 'temp':
225                 self.pushFrom('',5+int(index))
226 
227             self.push()
```