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 8 - Virtual Machine II - Control

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




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.
```

2 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=project8.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 *~
```

3 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('.')[-2]
      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;
       self.lines_words = self.curLine.split();


   def commandType(self):



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

       elif (self.lines_words[0]) == 'push':
```

```
60              return Command.C_PUSH;
61
62          elif (self.lines_words[0]) == 'pop':
63              return Command.C_POP;
64
65          elif (self.lines_words[0]) == 'call':
66              return Command.C_CALL;
67          elif (self.lines_words[0]) == 'function':
68              return Command.C_FUNCTION;
69          elif (self.lines_words[0]) == 'label':
70              return Command.C_LABEL;
71          elif (self.lines_words[0]) == 'goto':
72              return Command.C_GOTO;
73          elif (self.lines_words[0]) == 'if-goto':
74              return Command.C_IF;
75          elif (self.lines_words[0]) == 'return':
76              return Command.C_RETURN;
77
78      @property
79      def arg1(self):
80
81          if (self.commandType() == Command.C_ARITHMETIC):
82              return self.lines_words[0];
83          else:
84              return self.lines_words[1];
85
86
87
88
89      def arg2(self):
90
91          return self.lines_words[2];
92          # if (self.commandType() == Command.C_PUSH) or (self.commandType() == Command.C_POP):
93          #
```

4 VMtranslator

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

5 VMtranslator.py

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

def parse(w, parser):
    while (parser.hasMoreCommands()):
        parser.advance();
        # arithmetic
        if parser.commandType() is Command.C_ARITHMETIC:
            w.writeArithmetic(parser.arg1);
        # pop
        elif parser.commandType() == Command.C_POP:
            w.writePushPop(Command.C_POP, parser.arg1, parser.arg2());
        # push
        elif parser.commandType() == Command.C_PUSH:
            w.writePushPop(Command.C_PUSH, parser.arg1, parser.arg2());
        # function
        elif parser.commandType() == Command.C_FUNCTION:
            w.writeFunction(parser.arg1, parser.arg2());
        # return
        elif parser.commandType() == Command.C_RETURN:
            w.writeReturn();
        # call
        elif parser.commandType() == Command.C_CALL:
            w.writeCall(parser.arg1, parser.arg2());
        # label
        elif parser.commandType() == Command.C_LABEL:
            w.writeLabel(parser.arg1);
        # goto
        elif parser.commandType() == Command.C_GOTO:
            w.writeGoto(parser.arg1);
        # if-goto
        elif parser.commandType() == Command.C_IF:
            w.writeIf(parser.arg1);

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])

        parse(w, parser)


        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]):
```

```
60              if each_file.endswith(".vm"):
61
62                  parser = Parser(path+each_file);
63                  w.setFileName(parser.name.split('/')[-1])
64
65                  parse(w, parser)
66
67          w.close();
68
69
70
71
72
73  if __name__ == "__main__":
74      main(sys.argv[1:])
```

6 codeWriter.py

```
from io import *
from Parser import *

__author__ = 'roigreenberg'

class CodeWriter:

    fileName = ''
    functionName =''
    labelIndex = 0
    returnIndex = 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')
        self.writeCall('Sys.init', 0)

    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
```

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

```
128             self.labelIndex+=1
129
130         def pushFrom(self, seg, i):
131             self.w.write('@'+str(i)+'\n')
132             if seg != '':
133                 self.w.write('D=A\n')
134                 self.w.write('@'+seg+'\n')
135                 self.w.write('A=D+M\n')
136             self.w.write('D=M\n')
137
138         def popTo(self, seg, i):
139             if seg != '':
140                 self.w.write('@R13\n')
141                 self.w.write('M=D\n')
142
143             self.w.write('@'+str(i)+'\n') #find and save memory address
144             if seg != '':
145                 self.w.write('D=A\n')
146                 self.w.write('@'+seg+'\n')
147                 self.w.write('D=D+M\n')
148                 self.w.write('@R14\n')
149                 self.w.write('M=D\n')
150
151                 self.w.write('@R13\n') #set to  memory address
152                 self.w.write('D=M\n')
153                 self.w.write('@R14\n')
154                 self.w.write('A=M\n')
155             self.w.write('M=D\n')
156
157         def writeArithmetic(self, command):
158
159             if command == 'add':
160                 self.pop2()
161                 self.w.write('@R13\n')
162                 self.w.write('D=D+M\n')
163                 self.push()
164             elif command == 'sub':
165                 self.pop2()
166                 self.w.write('@R13\n')
167                 self.w.write('D=D-M\n')
168                 self.push()
169             elif command == 'neg':
170                 self.pop()
171                 self.w.write('D=-D\n')
172                 self.push()
173             elif command == 'eq':
174                 self.compare('JEQ')
175
176             elif command == 'gt':
177                 self.compare('JGT')
178             elif command == 'lt':
179                 self.compare('JLT')
180             elif command == 'and':
181                 self.pop2()
182                 self.w.write('@R13\n')
183                 self.w.write('D=D&M\n')
184                 self.push()
185             elif command == 'or':
186                 self.pop2()
187                 self.w.write('@R13\n')
188                 self.w.write('D=D|M\n')
189                 self.push()
190             elif command == 'not':
191                 self.pop()
192                 self.w.write('D=!D\n')
193                 self.push()
194
195         def writePushPop(self, command, segment, index):
```

```
196             if command == Command.C_POP:
197                 self.pop()
198                 if segment == 'local':
199                     self.popTo('LCL', index)
200                 elif segment == 'argument':
201                     self.popTo('ARG', index)
202                 elif segment == 'this':
203                     self.popTo('THIS', index)
204                 elif segment == 'that':
205                     self.popTo('THAT', index)
206                 elif segment == 'static':
207                     self.popTo('',self.fileName+'.'+ str(index))
208                     #self.popTo(self.fileName+'.',int(index))
209                 elif segment == 'pointer':
210                     self.popTo('',3+int(index))
211                 elif segment == 'temp':
212                     self.popTo('',5+int(index))
213             elif command == Command.C_PUSH:
214 
215                 if segment == 'constant':
216                     self.w.write("@"+str(index)+"\n")
217                     self.w.write('D=A\n')
218                 elif segment == 'local':
219                     self.pushFrom('LCL', index)
220                 elif segment == 'argument':
221                     self.pushFrom('ARG', index)
222                 elif segment == 'this':
223                     self.pushFrom('THIS', index)
224                 elif segment == 'that':
225                     self.pushFrom('THAT', index)
226                 elif segment == 'static':
227                     self.pushFrom('',self.fileName+'.'+ str(index))
228                     #self.pushFrom(self.fileName+'.',int(index))
229                 elif segment == 'pointer':
230                     self.pushFrom('',3+int(index))
231                 elif segment == 'temp':
232                     self.pushFrom('',5+int(index))
233 
234                 self.push()
235 
236         def writeLabel(self, label):
237             self.w.write('(' + self.functionName +'$' + str(label) + ')\n')
238 
239         def writeGoto(self, label):
240             self.w.write('@' + self.functionName +'$' + str(label) + '\n')
241             self.w.write('0;JMP\n')
242 
243         def writeIf(self, label):
244             self.pop()
245             self.w.write('@' + self.functionName +'$' + str(label) + '\n')
246             self.w.write('D;JNE\n')
247 
248         def writeCall(self, functionName, numArg):
249             self.w.write('@Return$$' + str(self.returnIndex) + '\n')
250             self.w.write('D=A\n')
251             self.push()
252             self.w.write('@LCL\n')
253             self.w.write('D=M\n')
254             self.push()
255             self.w.write('@ARG\n')
256             self.w.write('D=M\n')
257             self.push()
258             self.w.write('@THIS\n')
259             self.w.write('D=M\n')
260             self.push()
261             self.w.write('@THAT\n')
262             self.w.write('D=M\n')
263             self.push()
```

```
264             self.w.write('@SP\n')
265             self.w.write('D=M\n')
266             self.w.write('@LCL\n')
267             self.w.write('M=D\n')
268             self.w.write('@'+ str(int(numArg) + 5)+'\n')
269             self.w.write('D=D-A\n')
270             self.w.write('@ARG\n')
271             self.w.write('M=D\n')
272             self.w.write('@' + str(functionName) + '\n') # MISSING SOMETHING?
273             self.w.write('0;JMP\n')
274             self.w.write('(Return$$' + str(self.returnIndex) + ')\n')
275             self.returnIndex+=1
276 
277         def writeReturn(self):
278 
279             self.pop()
280             self.w.write('@R15\n')
281             self.w.write('M=D\n')
282             self.w.write('@LCL\n')  # set SP to LCL
283             self.w.write('D=M\n')
284             self.w.write('@SP\n')
285             self.w.write('M=D\n')
286             self.pop()                  # pop THAT
287             self.w.write('@THAT\n')     # set THAT
288             self.w.write('M=D\n')
289             self.pop()                  # pop THIS
290             self.w.write('@THIS\n')     # set THIS
291             self.w.write('M=D\n')
292             self.pop()                  # pop ARG
293             self.w.write('@R13\n')      # save ARG in R13
294             self.w.write('M=D\n')
295             self.w.write('@ARG\n')      # save this ARG
296             self.w.write('D=M\n')
297             self.w.write('@R14\n')
298             self.w.write('M=D\n')
299             self.w.write('@R13\n')      # take ARG
300             self.w.write('D=M\n')
301             self.w.write('@ARG\n')      # set ARG
302             self.w.write('M=D\n')
303 
304             self.pop()                  # pop LCL
305             self.w.write('@LCL\n')      # set LCL
306             self.w.write('M=D\n')
307             self.pop()                  # pop ret
308             self.w.write('@R13\n')      # save ret
309             self.w.write('M=D\n')
310 
311             self.w.write('@R14\n')      # set SP to this ARG
312             self.w.write('D=M\n')
313             self.w.write('@SP\n')
314             self.w.write('M=D\n')
315             self.w.write('@R15\n')
316             self.w.write('D=M\n')
317             self.push()
318             self.w.write('@R13\n')
319             self.w.write('A=M\n')
320             self.w.write('0;JMP\n')
321             return
322 
323         def writeFunction(self, functionName, numLocals):
324             self.functionName = functionName
325             self.w.write('(' + str(functionName) + ')\n') # MISSING SOMETHING?
326             for i in range(int(numLocals)):
327                 self.writePushPop(Command.C_PUSH, 'constant', 0)
328     #             self.writePushPop(Command.C_POP, 'local', i)
329     #
330     #
331     # w = CodeWriter('test')
```

```
# w.w.write('@START\n')
# w.w.write('0;JMP\n')
# w.writeFunction('z', '0')
# w.writePushPop(Command.C_PUSH, 'constant', 80)
# w.writePushPop(Command.C_PUSH, 'constant', 90)
# w.writePushPop(Command.C_POP, 'pointer', 0)
# w.writePushPop(Command.C_POP, 'pointer', 1)
# w.writePushPop(Command.C_PUSH, 'constant', 999)
#
# w.writeReturn()
# w.w.write('(START)\n')
# w.writePushPop(Command.C_PUSH, 'constant', 789)
# w.writePushPop(Command.C_PUSH, 'constant', 456)
# w.writePushPop(Command.C_PUSH, 'constant', 123)
# w.writeCall('z', '0')
# w.writePushPop(Command.C_PUSH, 'constant', 951)
#
#
#
#
# w.close()
```