Contents

1 Basic Test Results

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

  preparing sub.tar
dos2unix: converting file /tmp/bodek.sV5yOt/nand2tet/Project07/ransha/presubmission/testdir/stud/sub.tar/README to Unix form
  checking sub.tar
make: Nothing to be done for 'all'.
 testing
\
      Execution_PointerTest(max) {Problem in execution of PointerTest.asm: Comparison failure at line 2} \
      Execution_BasicTest(max) {Problem in execution of BasicTest.asm: Comparison failure at line 2}
********** TEST END **********
```

2 README

```
nivkeren,ransha
================================================================================
Niv Keren, ID 201478351, niv.keren@mail.huji.ac.il
Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
================================================================================

                    Project 7- Virtual Machine I - Stack Arithmetic
                            ------------------------


Submitted Files
---------------
README        - This file.
Makefile    - An empty file. no need to compile in Python
VMtranslator- A shell script that runs the python script with the given the
              asm file name (or folder) as an argument
VMtranslator.py    - The main program
CodeWriter.py- Translate the Commands from VM to assembly
Command.py    - A class represanting a command line
Parser.py    - Parses the given file to commands list

Run command
-----------

./VMtranslator <file_name>

Remarks
-------
* Our implementation followed the design given in the lectures
* "It's a trap"
```

3 CodeWriter.py

```
from Command import Command
import Parser


# Constants
DEF_ENCODING = "utf-8"
INIT_STR = '@END\n0;JMP\n(WRITE_EQ)\n@R15\nM=D\n@SP\nAM=M-1\nD=M\nA=A-1\nD=M-D\nM=0\n@END_EQ\nD;JNE\n@SP\nA=M-1\nM=-1\n(END_
    '0;JMP\n(WRITE_GT)\n@R15\nM=D\n@SP\nAM=M-1\nD=M\nA=A-1\nD=M-D\nM=0\n@END_GT\nD;JLE\n@SP\nA=M-1\nM=-1\n(END_GT)\n' + \
    '@R15\nA=M\n0;JMP\n(WRITE_LT)\n@R15\nM=D\n@SP\nAM=M-1\nD=M\nA=A-1\nD=M-D\nM=0\n@END_LT\nD;JGE\n@SP\nA=M-1\nM=-1\n' + \
    '(END_LT)\n@R15\nA=M\n0;JMP\n(END)\n@END\n0;JMP\n'

VM_SUFF = ".vm"

SEG_CONSTANT = 'constant'
SEG_ARGUMENT = 'argument'
SEG_LOCAL = 'local'
SEG_STATIC = 'static'
SEG_THIS = 'this'
SEG_THAT = 'that'
SEG_POINTER = 'pointer'
SEG_TEMP = 'temp'

# Variables
line_num = 0
jmp_counter = 0
static_counter = 0
vm_file = ''
asm_file_name = ''
content = []


def set_asm_file(filename):
    global asm_file_name
    global content
    asm_file_name = filename
    content = []
    with open(asm_file_name, mode='w', encoding=DEF_ENCODING) as asm_file:
        i=0


def write_asm():
    with open(asm_file_name, mode='a', encoding=DEF_ENCODING) as asm_file:
        for command in content:
            asm_file.write('%s\n' % command)
        asm_file.write(INIT_STR)


def set_vm_file(filename):
    global content
    global static_counter
    global vm_file
    with open(asm_file_name, mode='a', encoding=DEF_ENCODING) as asm_file:
        for command in content:
            asm_file.write('%s\n' % command)
    static_counter = 0
    content = []
    vm_file = filename


```

```
60  def write_unary_op():
61      content.append('@SP')
62      content.append('A=M-1')
63
64
65  def write_binary_op():
66      content.append('@SP')
67      content.append('AM=M-1')
68      content.append('D=M')
69      content.append('A=A-1')
70
71
72  def write_arithmetic(command):
73      global jmp_counter
74      global content
75      if command == Command.A_ADD:
76          write_binary_op()
77          content.append('M=D+M')
78      elif command == Command.A_SUB:
79          write_binary_op()
80          content.append('M=M-D')
81      elif command == Command.A_EQ:
82          content.append('@RET_ADDRESS' + str(jmp_counter))
83          content.append('D=A')
84          content.append('@WRITE_EQ')
85          content.append('0;JMP')
86          content.append('(RET_ADDRESS' + str(jmp_counter) + ')')
87          jmp_counter += 1
88      elif command == Command.A_LT:
89          content.append('@RET_ADDRESS' + str(jmp_counter))
90          content.append('D=A')
91          content.append('@WRITE_LT')
92          content.append('0;JMP')
93          content.append('(RET_ADDRESS' + str(jmp_counter) + ')')
94          jmp_counter += 1
95      elif command == Command.A_GT:
96          content.append('@RET_ADDRESS' + str(jmp_counter))
97          content.append('D=A')
98          content.append('@WRITE_GT')
99          content.append('0;JMP')
100         content.append('(RET_ADDRESS' + str(jmp_counter) + ')')
101         jmp_counter += 1
102     elif command == Command.A_NEG:
103         write_unary_op()
104         content.append('M=-M')
105     elif command == Command.A_AND:
106         write_binary_op()
107         content.append('M=D&M')
108     elif command == Command.A_OR:
109         write_binary_op()
110         content.append('M=D|M')
111     elif command == Command.A_NOT:
112         write_unary_op()
113         content.append('M=!M')
114
115 def write_push_pop(push_pop, segment, index):
116     global content
117     static_name = vm_file[:-len(VM_SUFF)]
118     if push_pop == Parser.CommandType.C_PUSH:
119         if segment == SEG_CONSTANT:
120             content.append('@' + str(index))
121             content.append('D=A')
122         elif segment == SEG_LOCAL:
123             content.append('@LCL')
124             content.append('D=M')
125             content.append('@' + str(index))
126             content.append('A=D+A')
127             content.append('D=M')
```

```
128             elif segment == SEG_ARGUMENT:
129                 content.append('@ARG')
130                 content.append('D=M')
131                 content.append('@' + str(index))
132                 content.append('A=D+A')
133                 content.append('D=M')
134             elif segment == SEG_THIS:
135                 content.append('@THIS')
136                 content.append('D=M')
137                 content.append('@' + str(index))
138                 content.append('A=D+A')
139                 content.append('D=M')
140             elif segment == SEG_THAT:
141                 content.append('@THAT')
142                 content.append('D=M')
143                 content.append('@' + str(index))
144                 content.append('A=D+A')
145                 content.append('D=M')
146             elif segment == SEG_POINTER:
147                 content.append('@R' + str(3 + index))
148                 content.append('D=M')
149             elif segment == SEG_TEMP:
150                 content.append('@TEMP')
151                 content.append('D=A')
152                 content.append('@' + str(5 + index))
153                 content.append('A=D+A')
154                 content.append('D=M')
155             elif segment == SEG_STATIC:
156                 content.append('@' + static_name + '.' + str(index))
157                 content.append('D=M')
158             content.append('@SP')
159             content.append('A=M')
160             content.append('M=D')
161             content.append('@SP')
162             content.append('M=M+1')
163 
164         elif push_pop == Parser.CommandType.C_POP:
165             if segment == SEG_LOCAL:
166                 content.append('@LCL')
167                 content.append('D=M')
168                 content.append('@' + str(index))
169                 content.append('D=D+A')
170             elif segment == SEG_ARGUMENT:
171                 content.append('@ARG')
172                 content.append('D=M')
173                 content.append('@' + str(index))
174                 content.append('D=D+A')
175             elif segment == SEG_THIS:
176                 content.append('@THIS')
177                 content.append('D=M')
178                 content.append('@' + str(index))
179                 content.append('D=D+A')
180             elif segment == SEG_THAT:
181                 content.append('@THAT')
182                 content.append('D=M')
183                 content.append('@' + str(index))
184                 content.append('D=D+A')
185             elif segment == SEG_POINTER:
186                 content.append('@R' + str(3 + index))
187                 content.append('D=A')
188             elif segment == SEG_TEMP:
189                 content.append('@TEMP')
190                 content.append('D=A')
191                 content.append('@' + str(index))
192                 content.append('D=D+A')
193             elif segment == SEG_STATIC:
194                 content.append('@' + static_name + '.' + str(index))
195                 content.append('D=A')
```

```
196             content.append('@R13')
197             content.append('M=D')
198             content.append('@SP')
199             content.append('AM=M-1')
200             content.append('D=M')
201             content.append('@R13')
202             content.append('A=M')
203             content.append('M=D')
```

4 Command.py

```
class Command:
    """This class represents an assembly command."""

    # Constants:
    A_ADD = 'add'
    A_SUB = 'sub'
    A_EQ = 'eq'
    A_LT = 'lt'
    A_GT = 'gt'
    A_NEG = 'neg'
    A_AND = 'and'
    A_OR = 'or'
    A_NOT = 'not'

    # The type of the command
    type = -1

    # The content of the command - i.e. The label (without the parentheses), the address (without @) or the
    # command.
    content = ''

    def __init__(self, type, content):
        """ Basic Constructor, Initializes the command variables
        Input type - the type of the command (L,A or C)
           content - the command content
        """
        self.type = type
        self.content = content
```

5 Makefile

```
# --- Empty Makefile ---
all: ;

TAR_FILES=README Makefile VMtranslator VMtranslator.py CodeWriter.py Command.py Parser.py
TAR_FLAGS=-cvf
TAR_NAME=project7.tar
TAR=tar

tar:
    $(TAR) $(TAR_FLAGS) $(TAR_NAME) $(TAR_FILES)

.PHONY: all tar
```

6 Parser.py

```
from enum import Enum
from Command import Command
import os

""" The parser module for the assembler.
"""

# Constants.
COMMENT_PREFIX = '//'
READ_ONLY = 'r'
DEF_ENCODING = 'utf-8'
EMPTY_LINE = ''

STR_ARITHMETIC = ['add', 'sub', 'neg', 'eq', 'gt', 'lt', 'and', 'or', 'not']
STR_PUSH = 'push'
STR_POP = 'pop'
STR_LABEL = 'label'
STR_GOTO = 'goto'
STR_IF = 'if'
STR_FUNCTION = 'function'
STR_RETURN = 'return'
STR_CALL = 'call'

class CommandType(Enum):
    ''' Enum for the command type.
    '''
    NO_COMMAND = -1
    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


content = []

def parse(file_name):
    """ Parse a given assembly language file.
    """
    # Clean up when parsing a new file
    global content
    content = []
    current_command = None
    # Read the file and parse lines
    with open(file_name, mode=READ_ONLY, encoding=DEF_ENCODING) as vm_file:
        for line in vm_file:
            # Ignore whitespace & comments in the start and end of the line
            found_comment = line.find(COMMENT_PREFIX)
            if found_comment != -1:
                line = line[:found_comment]

            line = line.strip().split(' ')
            if line[0] == EMPTY_LINE:
                continue
            # Determine whether current line is A/L/C CommandType (L for Label)
```

```
60              elif line[0] in STR_ARITHMETIC:
61                  current_command = CommandType.C_ARITHMETIC
62              elif line[0] == STR_PUSH:
63                  current_command = CommandType.C_PUSH
64              elif line[0] == STR_POP:
65                  current_command = CommandType.C_POP
66              elif line[0] == STR_LABEL:
67                  current_command = CommandType.C_LABEL
68              elif line[0] == STR_GOTO:
69                  current_command = CommandType.C_GOTO
70              elif line[0] == STR_IF:
71                  current_command = CommandType.C_IF
72              elif line[0] == STR_FUNCTION:
73                  current_command = CommandType.C_FUNCTION
74              elif line[0] == STR_RETURN:
75                  current_command = CommandType.C_RETURN
76              elif line[0] == STR_CALL:
77                  current_command = CommandType.C_CALL
78
79              # Add the created command to the content list
80              content.append(Command(current_command, line))
81
82          # For loop ends here.
83
84      # File is closed here
85
86  def get_commands():
87      """ Get all commands in a parsed file - use this after running the parse function.
88      This is a generator, thus running 'for command in get_commands()' will yield
89      all commands in the parsed file in the correct order.
90      """
91      for command in content:
92          yield command
```

7 VMtranslator

```
#!/bin/sh

# Runs the python script with the given argument
python3 VMtranslator.py $*
```

8 VMtranslator.py

```
import Parser,CodeWriter,sys,os
from Command import Command

# Constants:
VM_SUFF = ".vm"
ASM_SUFF = ".asm"
W_FILE_MODE = "w"
DEF_ENCODING = "utf-8"

""" The vmtranslator main file.
"""
def parse_vm_file(file_name):
    """ Gets the commands list using the parser and scans it twice
        first time searching for labels, second time uses the code to translate
        the A and C commands to machine code.
        Adds the machine code to a new .asm file
        Input: file_name - the .vm file needed to be translated
        Output: the translated file_name.asm file
    """
    Parser.parse(file_name)
    CodeWriter.set_vm_file(file_name)
    for command in Parser.get_commands():
        if command.type == Parser.CommandType.C_ARITHMETIC:
            CodeWriter.write_arithmetic(command.content[0])
        elif command.type == Parser.CommandType.C_PUSH or \
              command.type == Parser.CommandType.C_POP:
            CodeWriter.write_push_pop(command.type, command.content[1], command.content[2])

def main():
    """ runs the assembler on the given argument (Assembler.py <file_name>)
    """
    file_name = sys.argv[1]
    if os.path.isfile(file_name):
        CodeWriter.set_asm_file(file_name[:-len(VM_SUFF)] + ASM_SUFF)
        parse_vm_file(file_name)
    elif os.path.isdir(file_name):
        if file_name.endswith('/'):
            file_name = file_name[:-1]
        dir_name = file_name.split("/")[-1]
        CodeWriter.set_asm_file(os.path.abspath(file_name) + "/" + dir_name + ASM_SUFF)
        os.chdir(file_name)
        for f in os.listdir():
            if f.endswith(VM_SUFF):
                parse_vm_file(f)
        os.chdir('..')
    CodeWriter.write_asm()

if __name__ == "__main__":
    main()
```