

Contents

1	Basic Test Results	2
2	README	3
3	CodeWriter.py	4
4	Command.py	8
5	Makefile	9
6	Parser.py	10
7	VMtranslator	12
8	VMtranslator.py	13

1 Basic Test Results

```
1 ***** TEST START *****
2
3   preparing sub.tar
4 dos2unix: converting file /tmp/bodek.sV5y0t/nand2tet/Project07/ransha/presubmission/testdir/stud/sub.tar/README to Unix form
5   checking sub.tar
6 make: Nothing to be done for 'all'.
7   testing
8
9 ***** TEST END *****
```

2 README

```
1  nivkeren,ransha
2  =====
3  Niv Keren, ID 201478351, niv.keren@mail.huji.ac.il
4  Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
5  =====
6
7          Project 7- Virtual Machine I - Stack Arithmetic
8          -----
9
10
11 Submitted Files
12 -----
13 README          - This file.
14 Makefile        - An empty file. no need to compile in Python
15 VMtranslator- A shell script that runs the python script with the given the
16                asm file name (or folder) as an argument
17 VMtranslator.py  - The main program
18 CodeWriter.py- Translate the Commands from VM to assembly
19 Command.py      - A class representing a command line
20 Parser.py       - Parses the given file to commands list
21
22 Run command
23 -----
24
25 ./VMtranslator <file_name>
26
27 Remarks
28 -----
29 * Our implementation followed the design given in the lectures
30 * "It's a trap"
```

3 CodeWriter.py

```
1  from Command import Command
2  import Parser
3
4
5  # Constants
6  DEF_ENCODING = "utf-8"
7  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(EN
8  '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' + \
9  '@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' + \
10 '@(END_LT)\n@R15\nA=M\n0; JMP\n(EN)\n@END\n0; JMP\n'
11
12 VM_SUFF = ".vm"
13
14 SEG_CONSTANT = 'constant'
15 SEG_ARGUMENT = 'argument'
16 SEG_LOCAL = 'local'
17 SEG_STATIC = 'static'
18 SEG_THIS = 'this'
19 SEG_THAT = 'that'
20 SEG_POINTER = 'pointer'
21 SEG_TEMP = 'temp'
22
23 # Variables
24 line_num = 0
25 jmp_counter = 0
26 static_counter = 0
27 vm_file = ''
28 asm_file_name = ''
29 content = []
30
31
32 def set_asm_file(filename):
33     global asm_file_name
34     global content
35     asm_file_name = filename
36     content = []
37     with open(asm_file_name, mode='w', encoding=DEF_ENCODING) as asm_file:
38         i=0
39
40
41 def write_asm():
42     with open(asm_file_name, mode='a', encoding=DEF_ENCODING) as asm_file:
43         for command in content:
44             asm_file.write('%s\n' % command)
45         asm_file.write(INIT_STR)
46
47
48 def set_vm_file(filename):
49     global content
50     global static_counter
51     global vm_file
52     with open(asm_file_name, mode='a', encoding=DEF_ENCODING) as asm_file:
53         for command in content:
54             asm_file.write('%s\n' % command)
55     static_counter = 0
56     content = []
57     vm_file = filename.split('/')[1]
58
59
```

```

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     index = int(index)
117     global content
118     static_name = vm_file[:-len(VM_SUFF)]
119     if push_pop == Parser.CommandType.C_PUSH:
120         if segment == SEG_CONSTANT:
121             content.append('@' + str(index))
122             content.append('D=A')
123         elif segment == SEG_LOCAL:
124             content.append('@LCL')
125             content.append('D=M')
126             content.append('@' + str(index))
127             content.append('A=D+A')

```

```

128         content.append('D=M')
129     elif segment == SEG_ARGUMENT:
130         content.append('@ARG')
131         content.append('D=M')
132         content.append('@' + str(index))
133         content.append('A=D+A')
134         content.append('D=M')
135     elif segment == SEG_THIS:
136         content.append('@THIS')
137         content.append('D=M')
138         content.append('@' + str(index))
139         content.append('A=D+A')
140         content.append('D=M')
141     elif segment == SEG_THAT:
142         content.append('@THAT')
143         content.append('D=M')
144         content.append('@' + str(index))
145         content.append('A=D+A')
146         content.append('D=M')
147     elif segment == SEG_POINTER:
148         content.append('@R' + str(3 + index))
149         content.append('D=M')
150     elif segment == SEG_TEMP:
151         content.append('@R' + str(5 + index))
152         content.append('D=M')
153     elif segment == SEG_STATIC:
154         content.append('@' + static_name + '.' + str(index))
155         content.append('D=M')
156         content.append('@SP')
157         content.append('A=M')
158         content.append('M=D')
159         content.append('@SP')
160         content.append('M=M+1')
161
162 elif push_pop == Parser.CommandType.C_POP:
163     if segment == SEG_LOCAL:
164         content.append('@LCL')
165         content.append('D=M')
166         content.append('@' + str(index))
167         content.append('D=D+A')
168     elif segment == SEG_ARGUMENT:
169         content.append('@ARG')
170         content.append('D=M')
171         content.append('@' + str(index))
172         content.append('D=D+A')
173     elif segment == SEG_THIS:
174         content.append('@THIS')
175         content.append('D=M')
176         content.append('@' + str(index))
177         content.append('D=D+A')
178     elif segment == SEG_THAT:
179         content.append('@THAT')
180         content.append('D=M')
181         content.append('@' + str(index))
182         content.append('D=D+A')
183     elif segment == SEG_POINTER:
184         content.append('@R' + str(3 + index))
185         content.append('D=A')
186     elif segment == SEG_TEMP:
187         content.append('@R' + str(5 + index))
188         content.append('D=A')
189     elif segment == SEG_STATIC:
190         content.append('@' + static_name + '.' + str(index))
191         content.append('D=A')
192         content.append('@R13')
193         content.append('M=D')
194         content.append('@SP')
195         content.append('AM=M-1')

```

```
196     content.append('D=M')
197     content.append('@R13')
198     content.append('A=M')
199     content.append('M=D')
```

4 Command.py

```
1 class Command:
2     """This class represents an assembly command."""
3
4     # Constants:
5     A_ADD = 'add'
6     A_SUB = 'sub'
7     A_EQ = 'eq'
8     A_LT = 'lt'
9     A_GT = 'gt'
10    A_NEG = 'neg'
11    A_AND = 'and'
12    A_OR = 'or'
13    A_NOT = 'not'
14
15    # The type of the command
16    type = -1
17
18    # The content of the command - i.e. The label (without the parentheses), the address (without @) or the
19    # command.
20    content = ''
21
22    def __init__(self, type, content):
23        """ Basic Constructor, Initializes the command variables
24        Input type - the type of the command (L,A or C)
25        content - the command content
26        """
27        self.type = type
28        self.content = content
```


5 Makefile

```
1  # --- Empty Makefile ---
2  all: ;
3
4  TAR_FILES=README Makefile VMtranslator VMtranslator.py CodeWriter.py Command.py Parser.py
5  TAR_FLAGS=-cvf
6  TAR_NAME=project7.tar
7  TAR=tar
8
9  tar:
10     $(TAR) $(TAR_FLAGS) $(TAR_NAME) $(TAR_FILES)
11
12  .PHONY: all tar
```

6 Parser.py

```
1  from enum import Enum
2  from Command import Command
3  import os
4
5  """ The parser module for the assembler.
6  """
7
8  # Constants.
9  COMMENT_PREFIX = '//'
10 READ_ONLY = 'r'
11 DEF_ENCODING = 'utf-8'
12 EMPTY_LINE = ''
13
14 STR_ARITHMETIC = ['add', 'sub', 'neg', 'eq', 'gt', 'lt', 'and', 'or', 'not']
15 STR_PUSH = 'push'
16 STR_POP = 'pop'
17 STR_LABEL = 'label'
18 STR_GOTO = 'goto'
19 STR_IF = 'if'
20 STR_FUNCTION = 'function'
21 STR_RETURN = 'return'
22 STR_CALL = 'call'
23
24 class CommandType(Enum):
25     ''' Enum for the command type.
26     '''
27     NO_COMMAND = -1
28     C_ARITHMETIC = 0
29     C_PUSH = 1
30     C_POP = 2
31     C_LABEL = 3
32     C_GOTO = 4
33     C_IF = 5
34     C_FUNCTION = 6
35     C_RETURN = 7
36     C_CALL = 8
37
38
39 content = []
40
41 def parse(file_name):
42     """ Parse a given assembly language file.
43     """
44     # Clean up when parsing a new file
45     global content
46     content = []
47     current_command = None
48     # Read the file and parse lines
49     with open(file_name, mode=READ_ONLY, encoding=DEF_ENCODING) as vm_file:
50         for line in vm_file:
51             # Ignore whitespace & comments in the start and end of the line
52             found_comment = line.find(COMMENT_PREFIX)
53             if found_comment != -1:
54                 line = line[:found_comment]
55
56             line = line.strip().split(' ')
57             if line[0] == EMPTY_LINE:
58                 continue
59             # 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

```
1  #!/bin/sh
2
3  # Runs the python script with the given argument
4  python3 VMtranslator.py $*
```

8 VMtranslator.py

```
1  import Parser,CodeWriter,sys,os
2  from Command import Command
3
4  # Constants:
5  VM_SUFF = ".vm"
6  ASM_SUFF = ".asm"
7  W_FILE_MODE = "w"
8  DEF_ENCODING = "utf-8"
9
10 """ The vmtranslator main file.
11 """
12 def parse_vm_file(file_name):
13     """ Gets the commands list using the parser and scans it twice
14         first time searching for labels, second time uses the code to translate
15         the A and C commands to machine code.
16         Adds the machine code to a new .asm file
17         Input: file_name - the .vm file needed to be translated
18         Output: the translated file_name.asm file
19     """
20     Parser.parse(file_name)
21     CodeWriter.set_vm_file(file_name)
22     for command in Parser.get_commands():
23         if command.type == Parser.CommandType.C_ARITHMETIC:
24             CodeWriter.write_arithmetic(command.content[0])
25         elif command.type == Parser.CommandType.C_PUSH or \
26             command.type == Parser.CommandType.C_POP:
27             CodeWriter.write_push_pop(command.type, command.content[1], command.content[2])
28
29 def main():
30     """ runs the assembler on the given argument (Assembler.py <file_name>)
31     """
32     file_name = sys.argv[1]
33     if os.path.isfile(file_name):
34         CodeWriter.set_asm_file(file_name[:-len(VM_SUFF)] + ASM_SUFF)
35         parse_vm_file(file_name)
36     elif os.path.isdir(file_name):
37         if file_name.endswith('/'):
38             file_name = file_name[:-1]
39         dir_name = file_name.split("/")[-1]
40         CodeWriter.set_asm_file(os.path.abspath(file_name) + "/" + dir_name + ASM_SUFF)
41         os.chdir(file_name)
42         for f in os.listdir():
43             if f.endswith(VM_SUFF):
44                 parse_vm_file(f)
45         os.chdir('.')
46     CodeWriter.write_asm()
47
48 if __name__ == "__main__":
49     main()
```