Assignment-My Virtual Machine-Python-DESCRIPTION

Version: Refer to the version in the file name

Contents

1 Problem to Solve

Develop a virtual machine (Vm) (i.e., a simulator or interpreter) in Python for the **myVM instruction set**. Call this virtual machine **myVM**.

Appendix A contains the description of the **myVM programming language**.

myVM consists of the following two phases:

1.1 Load Phase

Loads the myVM program from an external file called “**myVM_Prog.txt**” into the memory. Memory is represented by an array of size MAX_MEMORY_SIZE = 500.

1.2 Fetch-execute-cycle Phase

Executes the myVM program using the following steps:

```
initialize the program counter to the address of the first instruction
repeat
    fetch the instruction located at memory[program counter]
    increment the program counter
    decode the fetched instruction
    execute the fetched instruction
end when operation = HALT
```

2 Output

a. Display

   i. Your name and class information,

   ii. A line of 46 asterisks (*) and, finally,

   iii. The result of the myVM execution.

b. The output must be displayed on the screen, and also stored on an external file called **myVM_Output.txt.**

3 Incremental Development

Develop and test the virtual machine incrementally and thoroughly. You may, for example, develop and test your program on the following simple programs, first:

Program 1:

OUT 234
HALT

myVM displays 234.

Program 2:

STO number 234
OUT number
HALT

myVM stores 234 in the variable number, then displays the value of number, 234.

myVM expects the myVM source program to be stored in a text file called "**myVM_Prog.txt**". myVM will display the result on the monitor, and will also store it in a text file called "**myVM_Output.txt**"

Please note that I will test your program on my own sample myVM programs.

4 An Example Program in myVM (Stored in myVM_Prog.txt)

```
; Display a sequence of numbers in reverse order
;
; Initialize the values
STO increment 2
OUT "Enter an initial integer number:"
IN number
;
; Display the values
OUT "The values in reverse order are:"
SUB tempNum number 1
loopStart
BRn tempNum loopEnd
OUT number
SUB number number increment
SUB tempNum number 1
JMP loopStart
loopEnd
;
; Test the last value of number
OUT "The last value of number is"
OUT number
BRzp number ifEnd
OUT "The last value of number is negative."
JMP ifEnd
OUT "The last value of number is not negative."
ifEnd
;
OUT "Have a nice day!"
HALT
```

5 The result of the execution of the above program

5.1 As shown on the display monitor

```
John & Jane Doe, CSCI 4200, Spring 2025
*********************************************
Enter an initial integer number:
7
The values in reverse order are:
7
5
3
1
The last value of number is
```

```
-1
The last value of number is negative.
Have a nice day!
```

5.2 As stored in myVM_Output.txt – Note: The input value is not displayed!

```
John & Jane Doe, CSCI 4200, Spring 2025
*********************************************
Enter an initial integer number:
The values in reverse order are:
7
5
3
1
The last value of number is
-1
The last value of number is negative.
Have a nice day!
```

6 Naming of the Program Modules

Use the following names for your program modules:

Module	Name
Python script file	**myVM.py**
Input program source file	**myVM_Prog.txt** This file must be stored in the same directory as the main Python program. That is, to open the file, you must not specify a path, but the file name itself.
Output file containing the result of Vm execution	**myVM_Output.txt** This file must be stored in the same directory as the main Python program. That is, to open the file, you must not specify a path, but the file name itself.

7 What to turn in

Name	Module
1. myVM.Py	The virtual machine Python program
2. myVM_Prog.txt	Input program source file containing the Example myVM Program shown in Section 4, above.
3. myVM_Output.txt	Output file containing the result of the Vm execution

Appendix A – myVM Programming Language

8 Introduction to myVM

9 Introduction to myVM

The myVM instruction set is divided into three different categories: operation, data movement, and control.

Each instruction except for the comment line (which starts with a semicolon) and a label line (which is an identifier) has the following format:

Operator Operands

Hence, a line which does not start with either a semicolon or an operator is a label.

10 Operation Instructions

10.1 ADD Destination Source1 Source2

ADD accepts two integer values, Source1 and Source2, and stores their sum in the *Destination* variable, Destination.

Each Source can be either a variable or an integer constant.

Examples:

Code:
; Add 3 to the value of num1 and store the result in sum

```
ADD sum num1 3
```

Code:
; Add the values of num1 and num2 and store the result in sum

```
ADD sum num1 num2
```

Code:
; Add 40 and 6 and store the result in sum

```
ADD sum 40 6
```

10.2 SUB Destination Source1 Source2

SUB accepts two integer values, Source1 and Source2, and stores the result of (Source1 – Source2) in the *Destination* variable, Destination.

Each Source can be either a variable or an integer constant.

10.3 MUL Destination Source1 Source2

MUL accepts two integer values, Source1 and Source2, and stores the result of (Source1 * Source2) in the *Destination* variable, Destination.

Each Source can be either a variable or an integer constant.

10.4 DIV Destination Source1 Source2

DIV accepts two integer values, Source1 and Source2, and stores the result of (Source1 / Source2) in the *Destination* variable, Destination.

Each Source can be either a variable or an integer constant.

10.5 IN Variable

Inputs an integer value and stores it in Variable.

Example:

Code:
; Input an integer value and store it in the variable num

```
IN num
```

10.6 OUT Value

Display Value.

Value can be either an integer variable or a string of characters enclosed in quotation marks (" ")

Example:

Code:
; Display the value of the variable sum

```
OUT sum
```

If the value of sum is 1234, the output will be

```
1234
```

Code:
; Display “Hello World!”

```
OUT "Hello World!"
```

11 Data Movement Instructions

11.1 STO Destination Source

The STO instruction stores the value of Source in Destination variable.

Source can be either a variable or an integer constant.

Example:

Code:
; Store 23 in age

```
STO age 23
```

Code:
; Store the value of num2 in num1

```
STO num1 num2
```

12 Control Instructions

12.1 BRn Variable Label

If the value of Variable is negative, jump to Label.

Example:

Code:
; If the value of age is negative, jump to sumLabel

```
BRn age sumLabel
; Other statements
sumLabel
; Other statements
```

12.2 BRz Variable Label

If the value of Variable is zero, jump to Label.

Example:

Code:
; If the value of age is zero, jump to sumLabel

```
BRz age sumLabel
; Other statements
sumLabel
; Other statements
```

12.3 BRp Variable Label

If the value of Variable is positive, jump to Label.

Example:

Code:
; If the value of age is positive, jump to sumLabel

```
BRp age sumLabel
; Other statements
sumLabel
; Other statements
```

12.4 BRzp Variable Label

If the value of Variable is zero or positive, jump to Label.

Example:

Code:
; If the value of age is zero or positive, jump to sumLabel

```
BRzp age sumLabel
; Other statements
sumLabel
; Other statements
```

12.5 BRzn Variable Label

If the value of Variable is zero or negative, jump to Label.

Example:

Code:
; If the value of age is zero or negative, jump to sumLabel

```
BRzn age sumLabel
; Other statements
sumLabel
; Other statements
```

12.6 JMP Label

Jump to Label.

Example:

Code:
; Jump to sumLabel

```
JMP sumLabel
; Other statements
sumLabel
; Other statements
```

12.7 HALT

End the program execution.

---

13 Comments, Labels and Identifiers

Comment:

Semicolon (;) at the beginning of a line indicates a comment line, which will be ignored by myVMVM.

13.1 Identifier:

The name of an identifier starts with a letter, followed by a sequence of zero or more letters, digits or underscores.

An identifier may represent either a variable (of type integer) or a label.

13.2 Label:

Label is an identifier, which indicates a location within the myVM program. A label is a destination for a JMP or BRx instruction.

14 Summary of myVM Statements

Statement	Description
ADD Destination Source1	ADD accepts two integer values, Source1 and Source2, and stores their sum in the *Destination* variable, Destination.

	Each Source can be either a variable or an integer constant.
BRn Variable Label	If the value of Variable is negative, jump to Label.
BRp Variable Label	If the value of Variable is positive, jump to Label.
BRz Variable Label	If the value of Variable is negative, jump to Label.
BRzn Variable Label	If the value of Variable is zero or negative, jump to Label.
BRzp Variable Label	If the value of Variable is zero or positive, jump to Label.
Comment	Semicolon (;) at the beginning of a line indicates a comment line, which will be ignored by myVMVM.
DIV Destination Source1 Source2	DIV accepts two integer values, Source1 and Source2, and stores the result of (Source1 / Source2) in the *Destination* variable, Destination. Each Source can be either a variable or an integer constant.
HALT	End the program execution.
Identifier	The name of an identifier starts with a letter, followed by a sequence of zero or more letters, digits or underscores. An identifier may represent either a variable (of type integer) or a label.
IN Variable	Inputs an integer value and stores it in Variable.
JMP Label	Jump to Label.
Label	Label is an identifier, which indicates a location within the myVM program. It is used as a destination for a JMP or BRx instruction.
MUL Destination Source1 Source2	MUL accepts two integer values, Source1 and Source2, and stores the result of (Source1 * Source2) in the *Destination* variable, Destination. Each Source can be either a variable or an integer constant.
OUT Value	Display Value. Value can be either an integer variable or a string of characters enclosed in quotation marks (" ")
STO Destination Source	The STO instruction stores the value of Source in Destination variable. Source can be either a variable or an integer constant.
SUB Destination Source1 Source2	SUB accepts two integer values, Source1 and Source2, and stores the result of (Source1 – Source2) in the *Destination* variable, Destination. Each Source can be either a variable or an integer constant.

15 Reference

These instructions are adapted from the LC3 instruction set:

http://www.lc3help.com/tutorials.htm?article=Basic_LC-3_Instructions/