Simple Virtual Machine Implementation Resit Report

Overview

The following describes how different features of the simple virtual machine has been implemented with justifications.

To implement the virtual machine, the files in the working directory were first checked to know which of them a valid assembly file is. After which, C# types with IIinstruction as an interface were detected by reflection, and saved in a list. And each time a new simple machine language code needs to be executed, I compare the saved type names and the opcode and create an instance the opcode by reflection.

For memory efficiency, I saved the already found instructions in Struct based list that help me match instance of such instruction with their opcodes for easy access.

Implementing the Run Method

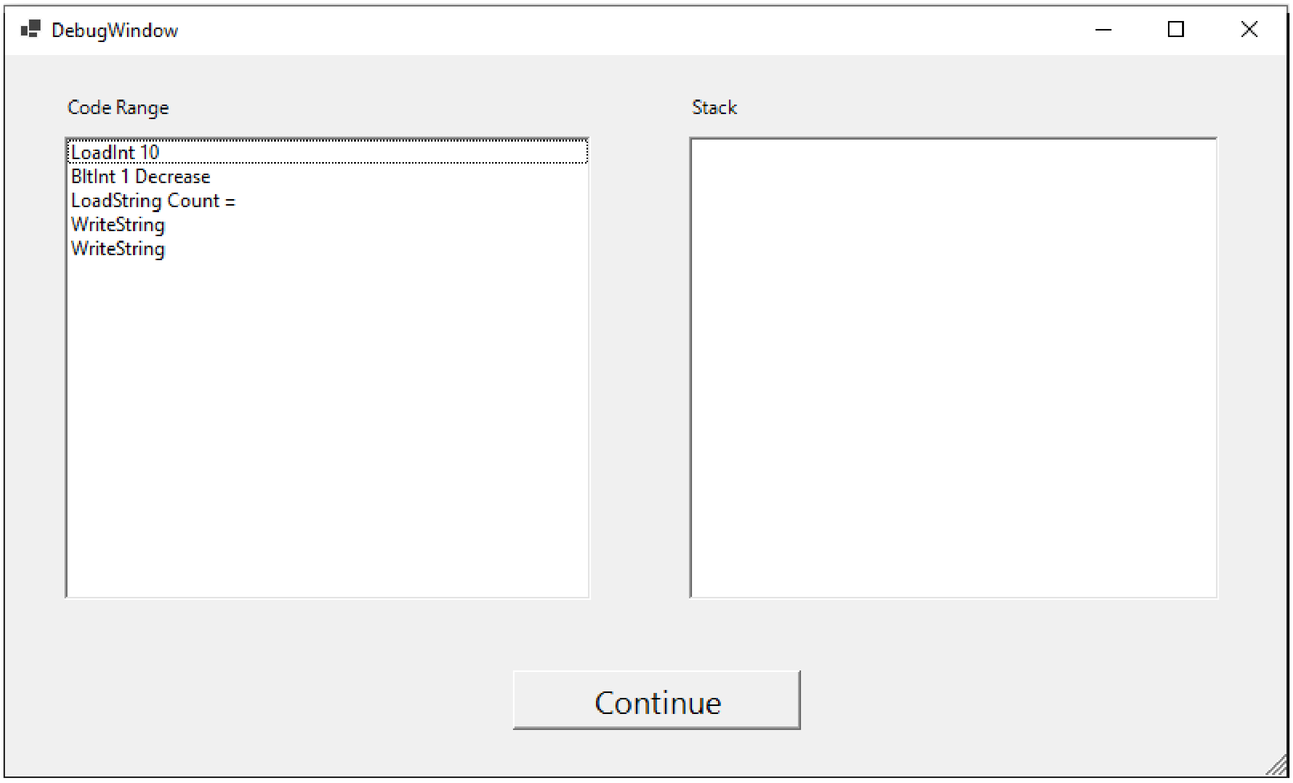
Knowing that the aim of the Run Method is to execute instances of found instructions on the virtual machine. Hence, using a while loop to go over the instruction instances, I passed the virtual machine to each before executing. The Run method is also responsible for handling the branching and starting of the debug window.

To achieve this, for each loop, if the executing instruction has its breakpoint property set to true, then, we the code frame can be generated and the debug window can be started with the most recent updates from the virtual machine.

Why it made sense to refactor the SvmVirtualMachine class

For general security of the virtual machine, introducing an interface class helps to conceal some important properties of the SvmVirtualMachine class from the public.

Screenshot of debug window

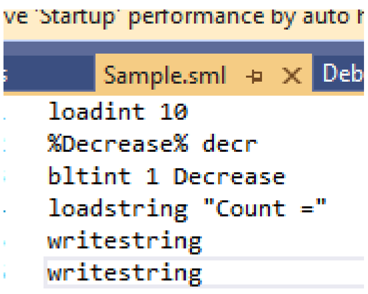


```
LoadInt 10
BltInt 1 Decrease
LoadString Count =
WriteString
WriteString
```

Branching of instructions

For branching to work, I created an if statement in the parseinstruction method to identity labels and store in a list with their matching instructions. As a result, each time a conditional instruction request for a branch, the list is looked up and the matching instruction is returned and made the next instruction.

Task 8



```
loadint 10
%Decrease% decr
bltint 1 Decrease
loadstring "Count ="
writestring
writestring
```

Further work.

I would recommend implementing strong names on the assembly files and writing unit test cases to increase the usability of the system.