****

**Report on Simple Virtual Machine (SVM) Implementation**

Assignment - Dec 2023

# Saghar Fadaei

202329649

## **Reflection Usage in Opcode Type Identification**

## Our approach to matching SML opcodes with corresponding C# types leverages Reflection. The process involves scanning loaded assemblies for classes implementing the `IInstruction` interface. Each class's name is compared (case-insensitively) to the opcode from the SML file. If a match is found, an instance of the corresponding class is created using `Activator.CreateInstance`

* Reflection is pivotal in dynamically identifying and instantiating appropriate instruction types at runtime, providing flexibility and extensibility to the SVM's instruction set. It means Reflection enables the SVM to scan through loaded assemblies, including external DLLs, to find classes that implement the IInstruction interface. This process is dynamic, meaning the SVM doesn't need prior knowledge of the instruction classes at compile time.

## **SvmVirtualMachine.Run() Method for Task 2**

* The `Run` method orchestrates the execution of compiled SML programs. It iterates through the list of `IInstruction` instances, setting each instruction's `VirtualMachine` property and executing them in sequence. The method also handles breakpoints and updates the program counter appropriately. Also in Run() I assigned the `this` reference (which refers to the current instance of the SvmVirtualMachine Class) to the VirtualMachine property of each instructions object. This allows the instructions to interact with the virtual machine’s stack and perform their operations.

## **Additional DLL Searching Mechanism**

* The SVM extends its instruction set by searching for instruction implementations in external DLLs located in the SVM executable's directory. The search avoids loading unnecessary assemblies by only considering those that contain relevant instruction types, thus optimizing memory usage.
* In this implementation, I have two methods SearchInExternalAssemblies and SearchInExternalAssembliesWithOperand (in JITCompiler.cs) that handle the dynamic loading and instantiation of instructions from external assemblies. These methods use reflection to load assemblies from the SVM executable directory, search for types that match the opcode, and create an instance of the found instruction type. A screen shot of a computer program

  Description automatically generated

A screen shot of a computer code

Description automatically generated

## **Refactoring SvmVirtualMachine and IVirtualMachine Interface Introduction**

* Introducing the `IVirtualMachine` interface and refactoring `SvmVirtualMachine` to implement this interface was a strategic decision. It enhances modularity, allowing different implementations of the virtual machine to be used interchangeably. This change is particularly beneficial for unit testing, where a mock implementation of `IVirtualMachine` can be employed. the introduction of the IVirtualMachine interface and the refactoring of the SvmVirtualMachine class represent a pivotal step in enhancing the SVM's architecture. This transformation not only improves modularity and testability but also ensures the SVM's readiness for future expansions or modifications, affirming its relevance and adaptability in evolving software landscapes.

## **Providing IVirtualMachine Instance to Incr and Decr Instructions**

* During unit testing, the `Incr` and `Decr` instructions require an `IVirtualMachine` instance. We achieved this by creating a mock implementation of `IVirtualMachine`, populating it with necessary data, and associating it with the instruction instances before running them.
* A screen shot of a computer program

  Description automatically generated

## **Screenshot Inclusions**

* Visual Studio Test Explorer/Test Results showing pass/fail status of unit tests.

A screenshot of a computer

Description automatically generated

* Debugger window displaying code and stack contents at a breakpoint.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A child in a white outfit

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

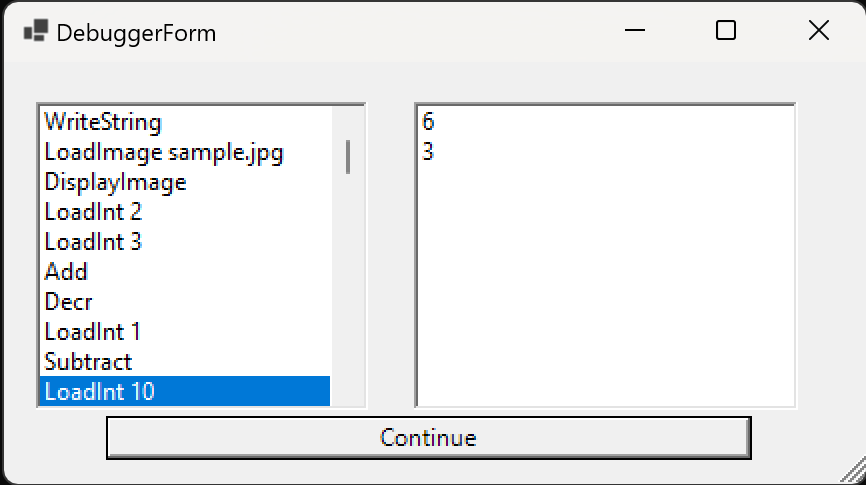
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

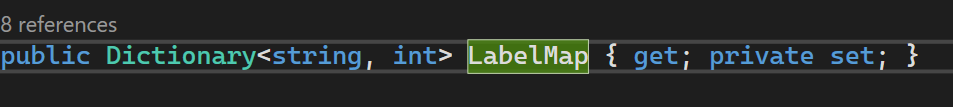
Description automatically generated

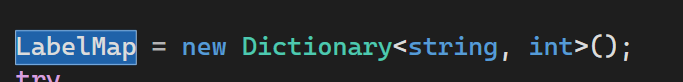
It’s a sample sml file I wrote, you can also add `inc`, `bgrint`, `decr`, `equint`, `notequ` intruction to test it.

## **Mechanism for Instruction Labeling and Execution**

* The SVM maintains a mapping (`LabelMap`) of labels to instruction indices. During the compilation phase, labels are extracted and stored in this map.   
  As the SVM reads the SML file, it identifies lines prefixed with % signifying labels. The ExtractLabel method accurately retrieves the label name from these lines, and the associated line number is recorded in the LabelMap. For instance, the label %loop% is extracted and mapped to its corresponding line number. During instruction execution, particularly in Goto instructions, the SVM refers to the LabelMap to locate the targeted label's associated instruction index. When encountering a Goto instruction, the SVM retrieves the index of the label from the LabelMap and sets the program counter to this index, enabling a direct jump to the labeled instruction.

Screenshot of part of my code :





A screenshot of a computer

Description automatically generated

In for example Bgrint run() :   
A computer screen shot of a math equation

Description automatically generated

## The ModifiedProgramCounter is a boolean property utilized within the SVM to track whether a specific type of instruction has altered the program counter's value during execution. This property is essential for controlling the flow of the program and keeping track of jumps or branches to other parts of the code.

## **Task 8 SML Source Code and Output**

* SML Code:

A screenshot of a computer program

Description automatically generated

I added screenshots of its functionality before.