Section I: How to reproduce the result

Section II: What has been done

Section III: What has not been done

Section IV: LLVMSharp functions

Section I: How to reproduce the result

[Steps]:

(1) Execute our code (after git clone from our shared GitHub), and the generated IR will be saved as .ll file in the IR folder.

(2) If using Windows OS, move the .ll file to any directory within WSL.

(3) At that directory of WSL, use the command: llc -march=riscv64 generated_IR.ll -o out3.s

This llc command will generate .s file for RISC-V assembly within the same directory.

(4) view the assembly: cat out3.s

If any of the above steps are unclear, please feel free to view our video from 1:43 which shows a video demonstration of the above steps.

Video (from 1:43):
https://www.youtube.com/watch?v=uV5Eei8CEzg

***NOTE**: You do not need to install anything but simply git clone our shared GitHub.

Section II: What has been done

Please see IR & RISC-V for sampleTest1.shank, sampleTest2.shank, and Fibonacci.shank

1) Expression: Any expression (mixture of both variable [load from hashmap] & number [create a new value of corresponding LLVM datatype]) of any length can be assigned to a variable through EvalExpression()
2) Statements: {Assignment, Function, For, While} statements are supported.
3) Functions: Code has been divided into functions to reduce redundancy
4) Generalizability: I have tested with a number of other .shank files to be generalized.

The custom functions (All in Expression.cs):

Exec_Assignment: used for assignment statement. Ex.) val := 1 + prev1 * 4 – prev2

Exec_Function: used for function statement. Ex.) write prev1

Exec_For: used for for loop

Exec_While: used for while loop

EvalExpression(): used to handle expression being assigned to a variable through recursion.

LLVMCompile(): the main function to generate IR

returnStatementTokens(): present at each node to return necessary tokens for the statement being processed.

Section III: What has not been done

1) If, else, else if statement
2) Function recursion
3) Array
4) Function creation other than write()
5) Nested loops
6) Data types other than integer
7) "mod" token
8) loop condition (other than "<" where the left side of the inequality increases to the right side of the inequality by 1")

Section IV: LLVMSharp functions

I have utilized all necessary functions by referencing to below 2 resources:
1) LLVMSharp.Interop 15.0.0 beta-1 API: https://www.nuget.org/packages/LLVMSharp.Interop/

2) HelloSharp: https://github.com/MoshiKoi/HelloSharp

I believe all necessary data types, functions, etc. can be found within the program and so you would not need to find much other necessary data types, functions, etc. However, if needed other necessary functions can be found from LLVMSharp.Interop version 15.0.0 beta-1 API from above link.

If you have any questions or if there is an error in reproducing of our results, please feel free to reach out to me through joshuacho370@gmail.com. (It is because I am not sure whether my @albany.edu will still work upon my graduation). Thank you!