California State University Stanislaus
Department of Computer Science

Programming Project 2

Modifying the Stack Machine Simulator to Add a New Instruction

We have introduced the computer architecture simulators for different instruction formats. In this project, you will modify the Stack Machine simulator to add a new Instruction

TOP Var

This instruction will save the value on top of stack into the variable Var without changing the stack.

In order to complete this project, you need to read the simulator source code, which is on the webpage of computer simulators: https://www.cs.csustan.edu/~xliang/Courses/SimulatorWeb. You will need to find out three places in the source to add your code according to a special comment there. Let me give you some details about each task on these three places.

In the first place, you will find out the special comment:

//--(1) You need to add the instruction name "TOP" at the end of this array

This array of strings is *operations*. Please note that the array index is the opcode of an instruction that is stored here. For example, the opcode of the instruction ADD is 0 because *operators*[0] = "ADD". So, you can figure out what is the opcode of the instruction TOP after adding "TOP" to the end of the array. In order to find out the opcode for a given instruction name, you will use the lookup table *opcodes*. For example, *opcodes*.get("ADD") = 0.

In the second place, you will find out the special comment:

//--(2) You need to encode the operand of TOP here

The operand of the instruction TOP Var is the memory address of the variable Var, which is stored in the symbol table *variables*. Please note that each assembly instruction is represented by an array of strings. For example, TOP Var is represented by the array ("TOP", "Var"). When encoding the instruction obj = ("TOP", "Var"), *variables*.get(obj[1]) will give you the operand, i.e. the memory address of Var.

In the third place, you will find out the special comment

//--(3) You need to add a case clause to execute TOP instruction

Here, you can refer the execution of the instruction POP Var. The difference between POP and TOP is that the POP will decrease the stack pointer after returning the value on top of the stack, while TOP will not change the stack but only return the value on top of stack. Please note that in my implementation of this simulator, the stack uses the data memory and the data memory is simulated by using an array called *data*.

The entire source code contains about 350 lines. The *initialize* routine is to build a lookup table called *opcodes* to get the opcode for a given instruction name. The scanner *routine* is to eliminate extra white

spaces and all comments in user assembly language source code. The *loader* routine has two stages. The first stage is to build a symbol table of variables. It has two phases to deal with the forward references. The second stage is to build a label table and to translate assembly instruction to its equivalent machine instruction. It also has two phases to deal with the forward references. If there is any pseudo-instruction, the pseudo-instruction should be break down to equivalent assembly instructions in the first phase. Note that there is no pseudo-instruction implemented for the stack machine yet. In fact, we could implement TOP as a pseudo-instruction. Note that POP A and PUSH A is equivalent with TOP A. Finally, the interpreter routine is to execute each machine instruction of a program.

You will need to have Java installed on your computer. The software I used was Eclipse Java Photon. Please write your name and date as comments in your source code. You can verify your code by running it and taking as input your assembly language program in project 1. You need to replace two instructions POP W and PUSH W with one instruction TOP W in your assembly language program.

Compiling and running your Java source code

You can use Eclipse Java to create a new Java Project. For JRE, select “Use an execution environment JRE: JavaSE-1.8 or above”. For Project layout, select “Create separate folders for sources and class files”. You can use any project name you like. Then click Finish button. Whin your project in Eclipse, you can import your source code under the src directory. Your input file (assembly language code) to your simulator should be put under your project directory. Now, you are ready to compile and run your simulator program. (I will show you these steps in a video.)

Testing your simulator

You can use your stack machine code from the project #1 for the testing. It should work correctly without any changes if your code was working in the project #1. Then you will replace two instructions POP immediately followed by PUSH with one TOP instruction in stack machine code. Run your simulator again with the new stack machine code. If you get the same result, congratulation! You are done.

What and how to turn in your project

You will submit your simulator source code in Canvas.