19AIE112 Elements of Computing System -2
S_2 B.Tech CSE(AI)

Project Report

Submitted by
Nithin Sylesh

AM.EN.U4AIE19044

June 2020
Department of Computer science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Department of Computer Science and Engineering

Certified that this is the bonafide project report for the course 19AIE112 Elements of Computing System -2 submitted on its completion by NITHIN SYLESH AM.EN.U4AIE19044, a second semester student of B.Tech CSE(AI)

Name and Signature
of the student

Nithin Sylesh

Name and Signature
of the examiner

Abstract

In this project we will build a vm translator .The vm translator Translates a file containing VM (Virtual Machine) commands into a Hack assembly language file. The VM specification can can be found in the section 7.2 of Nisan, Noam. "The Elements of Computing Systems: Building a Modern Computer from First Principles." The assembler can be invoked via command line , where the string fileName.vm is the translator's input, i.e. the name of a text file containing VM commands. The translator creates an output text file named fileName.asm, containing Hack assembly commands. The output file is stored in the same directory of the input file. The name of the input file may contain a file path.

Contents

1. INTRODUCTION

In this project-centered course you will build a modern software hierarchy, designed to enable the translation and execution of object-based, high-level languages on a bare-bone computer hardware platform. In particular, you will implement a virtual machine and a compiler for a simple, Java-like programming language, and you will develop a basic operating system that closes gaps between the high-level language and the underlying hardware platform. In the process, you will gain a deep, hands-on understanding of numerous topics in applied computer science, e.g. stack processing, parsing, code generation, and classical algorithms and data structures for memory management, vector graphics, input-output handling, and various other topics that lie at the very core of every modern computer system.

This is a self-contained course: all the knowledge necessary to succeed in the course and build the various systems will be given as part of the learning experience. The only prerequisite is knowledge of programming at the level acquired in introduction to computer science courses. All the software tools and materials that are necessary to complete the course will be supplied freely after you enrol in the course. This course is accompanied by the textbook "The Elements of Computing Systems" (Nisan and Schocken, MIT Press).

2. SOFTWARE AND HARDWARE REQUIREMENTS

Language used: Java Java is a high-level programming language developed by Sun Microsystems. It was originally designed for developing programs for set-top boxes and handheld devices, but later became a popular choice for creating web applications. The Java syntax is similar to C++, but is strictly an object-oriented programming language.

The Nand to Tetris Software Suite contains all the tools and files necessary for completing all the projects described in this site, and in the book *The Elements of Computing Systems*. Once you download the software suite to your PC, there is no need to download anything else throughout your Nand to Tetris learning experience.

The software runs as is on Windows, Unix, and Mac OS.

The software can be used freely under the terms of the GNU GPL (General Public License). The software is open source.

In order to use the nand2tetris software tools, your computer must be equipped with a Java Run-time Environment.

About the software

The Nand2tetris Software Suite consists of two directories: *projects*, and *tools*.

The projects directory is divided into 14 project directories named 00, 01, ..., 13 (of which project 00 is relevant only to learners who take the course in Coursera, and project 13 is open-ended). These directories contain files that you have to modify and complete as you work on various nand2tetris projects.

The tools directory contains the nand2tetris software tools. It's a collection of programs and files that will be explained as you follow the various projects.

The remainder of this section should be used as reference; there is no need to read what follows until you will be asked to use a particular software tool.

The .bat and .sh files are batch and script files, used to invoke the nand2tetris software tools. These files are explained in detail below.

The bin directory contains the code of the nand2tetris software tools. It consists of several subdirectories containing Java class files and supporting files.

The builtInChips and the builtInVMCode directories contain files that are used by the supplied Hardware Simulator and VM Emulator, respectively.

The OS directory contains a compiled version of the Jack operating system.

RUNNING THE SOFTWARE

The supplied software tools are designed to be run from your computer's command-line environment (also known as "terminal", or "shell"). Command-line environments vary from one operating system to another, and working in them requires some knowledge of various OS shell commands.
In order to eliminate this overhead, we supply batch files (for Windows) and scripts (for Unix and Mac OS), developed by Mark Armbrust. These batch and script files enable invoking the supplied nand2tetris tools from the command line on your computer, painlessly. They can be used from any working directory on your computer, without requiring full paths to the files on which they operate. Further, they accept spaces in directory and file names, so they will work if nand2tetris is installed under a directory named, say, "My Documents".
Mac and Linux users:

Before running the scripts, you must first change their file attributes to include "executable". You can then run the scripts by typing their name, as well as the .sh extension, in the terminal environment.

If you want to avoid typing the 'sh' extensions, you can create (once and for all) symbolic links in your ~/bin directory. Here is an example how to do it for, say, the HardwareSimulator tool:

ln -s ~/nand2tetris/tools/HardwareSimulator.sh HardwareSimulator
chmod +x HardwareSimulator
Windows users:
For the batch files to work from the command line, you must add (once and for all) the nand2tetris/toolsdirectory to your PATH variable.
To run a batch file from command-line, type its name, without the .bat extension.
If you use Windows 7 64-bit you need to install the 64-bit version of Java so that 64-bit cmdexe can run Java commands in batch files. If you get the output "'java' is not recognized..." you likely only have the 32-bit Java installed on your computer.
You can create desktop icons and use them to invoke the interactive versions of the following supplied tools: HardwareSimulator, Assembler, CPUEmulator and VMEmulator. This can be done by finding the disk locations of the respective batch files, right-clicking on them and picking "Send to > Desktop." Edit the shortcuts' properties and set "Run" to "minimized."

3. PROBLEM STATEMENT

Build a basic VM translator, focusing on the implementation of the VM language's stack arithmetic and memory accesscommands

Write a VM-to-Hack translator, conforming to the VM Specification. Use your VM translator to translate the VM files supplied below, yielding corresponding programs written in the Hack assembly language. When executed on the supplied CPU emulator, the translated code generated by your translator should deliver the results mandated by the test scripts .

4. IMPLEMENTATION DETAILS

We propose implementing the basic VM translator API described in chapter 7 in two stages. This will allow you to unit-test your implementation incrementally, using the test programs supplied below. In what follows, when we say "your VM translator should implement some VM command" we mean "your VM translator should translate the given VM command into a sequence of Hack assembly commands that accomplish the same task".

Stage I: Handling stack arithmetic commands: The first version of your basic VM translator should implement the nine arithmetic / logical commands of the VM language as well as the VM command push constant x.

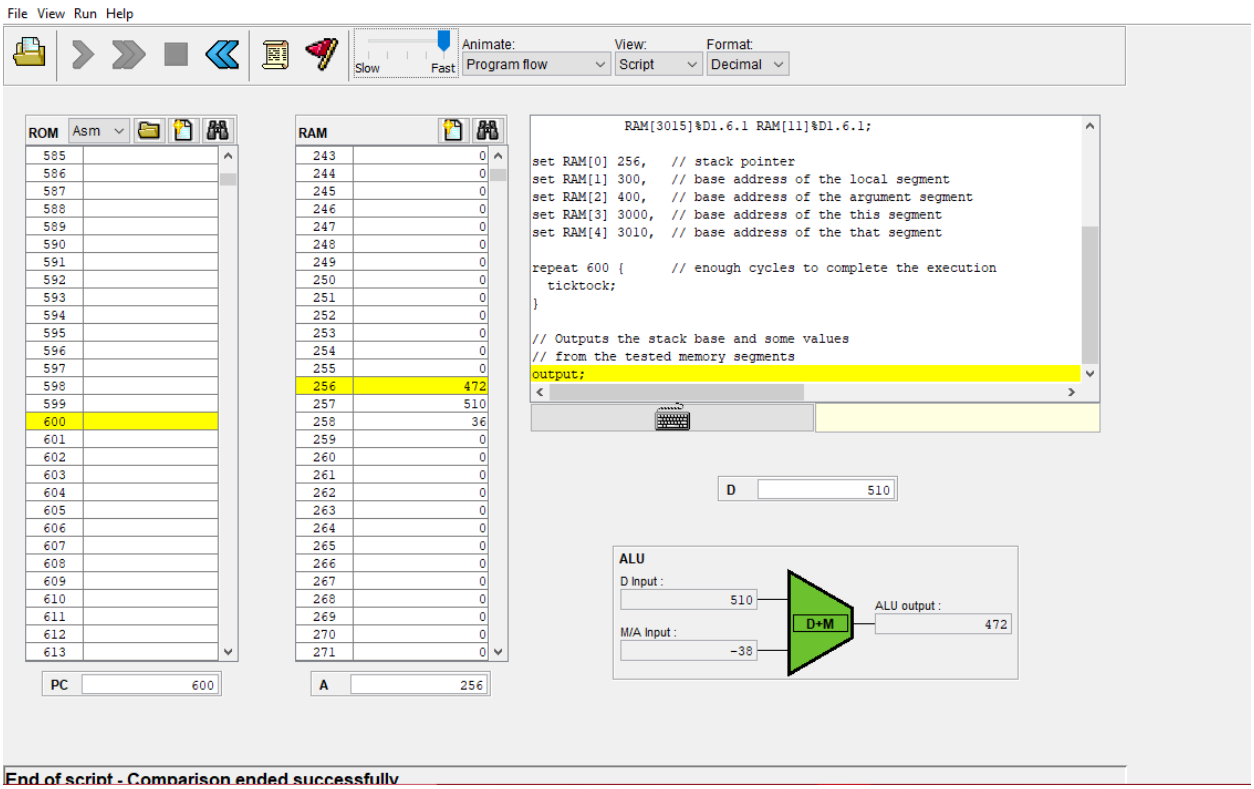
The latter command is the generic push command for which the first argument is constant and the second argument is some non-negative integer x. This command comes handy at this early stage, since it helps provide values for testing the implementation of the arithmetic / logical VM commands. For example, in order to test how your VM translator handles the VM add command, we can test how it handles the VM code push constant 3, pushconstant 5, add. The other arithmetic and logical commands are tested similarly.

Stage II: Handling memory access commands: The next version of your basic VM translator should include a full implementation of the VM language's push and pop commands, handling the eight memory segments described in chapter 7. We suggest breaking this stage into the following sub-stages:

1. You have already handled the constant segment;
2. Next, handle the segments local, argument, this, and that;
3. Next, handle the pointer and temp segments, in particular allowing modification of the bases of the this and that segments.
4. Finally, handle the static segment.

The supplied test programs were carefully planned to test the incremental features introduced by each development stage of your basic VM translator. Therefore, it's important to implement your translator in the proposed order, and to test it using the appropriate test programs at each stage. Implementing a later stage before an early one may cause the test programs to fail.

5. EXPERIMENTATION RESULT



6. CONCLUSION

The purpose of this project is to walk through the steps that you have to go through when you work on Project 7. So we have here my nand2tetris folder. Let's go into Project 7. And within Project 7 we'll go into memory access, and into basic test. And we see that what we have here are four files, a compare file. A test script design for the CPU emulator then we have the vehicle that you have to translate into assembly. And finally we have yet another test script design for the VM emulator. The main process here is basictest.vm, this is the program that we have to translate. we see that we have here a set of VM commands, which are designed to carry out various operations on the stack and the virtual segments, constant, local, argument, this, that and so on. Our problem is to translate this VM into assembly code. And then use the CPU emulator to run the assembly code and ascertain that resulting code actually carries out what the VM code was supposed to do. execution continues because there's nothing that tells it to stop in the code. And at some point, right here, execution has stopped. And the execution stopped because the supplied test script is programmed to execute 600 clock cycles. And therefore at some point it kind of forces the program to stop. The translated assembly code has passed the test that was mandated by the supplied test scripts

7. REFERENCE

a. www.nand2tetris.org

b. www.coursera.org

c. Noam Nisan and Shimon Schocken. 2008. The Elements of Computing Systems: Building a Modern Computer from First Principles (History of Computing S.). The MIT Press.