Computer System Design Lab
Design Experiment 7: VM Translator Design

Indian Institute of Technology Tirupati

Nov 13, 2019

1 Objective and Problem Statement:

Code portability has been a genuine concern for high-level programmers. Since the underlying hardware platform in the current time comes with variety of ISA, it is intelligent to think of a common platform just above the real hardware on which everyone agree upon and design their code accordingly. The programming language such as Java and .net has this concept inbuilt in them. In Java it is called as Java virtual machine and in .net it is called as common language runtime. The concept of virtual machine has now advanced to much extend that now we have the virtual machine like LLVM.

The objective of this exercise is to design such a virtual machine for the hardware under consideration. In this exercise a VM translator would be designed considering the H-hardware platform.

Problem statement:

Design a VM translator that would take a VM code and translate it to corresponding assembly code while detecting syntax errors. The input to VM translator would be .vm file and the output would be .asm file.

2 VM Commands

The virtual machine that is considered in N2T is the stack based VM. There are four different type of VM commands. Following Table list all the possible VM commands.

Table 1: List of VM commands.

Memory Access	ALU Command	Program Flow	Function Call
push *segment index*	add, sub, neg, eq, gt,	goto *symbol*	function *name nl*
pop *segment index*	lt, and, or, not	if-goto *symbol*	call *name na*
		label *symbol*	return

In the Table 1 *segment* is one of seven segments, *index* is a positive integer number within a range, *symbol* is a string of character and digit without any special character, *name* is name of the function (function name is also a string without any special character), *nl* is number of local variables, and *na* is number of arguments.

3 Assembly Instruction Set

All the instructions in the H-computer ISA are to be considered for assembler design. Following table lists all the possible instructions in the H-computer ISA.

The ISA consists of two class of instructions: A-class and C-class.

A-class:

Format: @*value*, where the *value* represents either a constant or a variable.
Example: @12, here 12 is a constant in the range of 0 to $2^{14} - 1$.
@*var*, here *var* is a variable which can be of string type without any special char.

C-class:

The C-class instructions are rest of all of the instruction set which includes ALU and jump type.

ALU instructions:

Format: *destination* = *computation*, where *destination* is one of the seven possible destinations and *computation* is one of the 28 ALU instructions.
Example: AMD = M - D, where destinations are A, M, and D and the computation is M - D

Jump instructions

Format: *comp;jump*, where *comp* is any of the valid compute type mnemonic and *jump* is the jump type mnemonic.
Example: D;JMP
0;JMP
D-A;JMP
D-A;JEQ
D-M;JEQ

The Table 2 and Table 3 enumerates all possible **compute** and **jump** instructions of C-class.

Table 2: In this table an instruction can be formed by reading as *dest = comp*.

comp→ dest↓	0, 1, -1	D	A	!D	!A	-D	-A	D + 1	A + 1	D - 1	A - 1	D + A	D - A	A - D	D&A	D | A
M	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
D	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
MD	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
A	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
AM	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
AD	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
AMD	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
dest↑ comp→			M		!M		-M		M + 1		M - 1	D + M	D - M	M - D	D & M	D | M

Table 3: An instruction in this table to be read as *comp;jump*.

jump → comp ↓	JGT	JEQ	JGE	JLT	JNE	JLE	JMP
any compute instruction	;	;	;	;	;	;	;

4 Design and Testing

The primary function of VM translator is to translate the command given Table 1 into the instruction set listed in Table 2 and Table 3.

1. Design:
   (a) Programming Language: Any language of your choice. Preferred is C/C++.
   (b) Design Methodology: the design could be done according to the guidelines provided in Chapter 7 and 8 of the text book. It is suggested that do the design in two stages. In the first stage implement all the commands from memory and alu. In the second stage extend the translator to consider the program flow and function call commands.

(c) The translator would take .vm code and produce .asm as output.

(d) Consider the as robust as possible which includes detection of syntax error among other features.

2. Testing:

   (a) Prepare three separate .vm files which would contain the vm commands of memory type, alu type, and program flow & function call type in the respective file. Translate these three files using your translator. Cross check your .asm output with that of the inbuilt translator.

   (b) Prepare an erroneous .vm file to check all the possible type of errors that your translator could detect.

   (c) Write two programs in vm: one to solve a polynomial equation $ax^2 + bx + c = 0$, and the other to perform insertion of bubble sort on array of size 5. Translate these two program into .asm code using your translator and run it on the inbuilt assembler simulator and cross check the output manually.

3. Wherever needed, try to automate as much as possible, I mean, you can write python, .tst etc to avoid manual intervention to the tool. Just run one script and the user should be able to see the output.

5 Experimental Flow

1. The experimental flow will follow the steps of Design and Testing. For design, follow the guidelines provided in text book and lecture presentation.

2. Testing need to be performed rigorously to highlight the pros and cons of your VM translator.

6 Tools:

- Language: The Nand2Tetris HDL and TSL (test scripting language)
  Refer: Appendix A and B of text book.
- For instruction format refer Chapter 4, 6, 7, and 8 of The Element of Computing System.
- Tools: Hardware Simulator of Nand2Tetris.
  `https://www.nand2tetris.org/software`
- Machine and OS: x86_64 machines with any distribution of Linux (Ubuntu or CentOS).

7 Reporting and Evaluation

Prepare a user manual of your translator. The user manual should contain a description about the feature of your translator. You are encouraged to include more detail such as the process of parsing.

The evaluation will be based on two factor: successful passing of Testing and the elegance of design that include the Robustness feature and the number of command that is supported.