Computer System Design Lab
Design Experiment 6: Assembler Design

Indian Institute of Technology Tirupati

Nov 5, 2019

1 Objective and Problem Statement:

Assembler plays an important role as an interface between the hardware and software. In the hierarchy of computer system where human has to interact the digital computer system, there are multiple components who are responsible for translating the human thought finally into the machine language. Assembler is a simple and clean design which just translate the assembly code to machine code.

The objective of this particular exercise is to design an assembler which translate the H-assembly language into H-machine language for N2T (Nand 2 Tetris) design. The assembler must check for the syntax error and must generate the appropriate machine code.

Problem statement:

Design an assembler that translate the H-assembly code into corresponding H-machine code such that the machine code should exactly give the same state as that of the H-assembly code.

2 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 1 and Table 2 enumerates all possible **compute** and **jump** instructions of C-class.

Table 1: 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 2: An instruction in this table to be read as *comp;jump*.

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

3 Design and Testing

All the instructions listed in Table 1 and Table 2 are to be considered for the assembler design.

1. Design:
   (a) Programming Language: Any language of your choice. Preferred is C/C++.
   (b) Two phase parsing: Make use of two phase parsing mechanism which was discussed during the class. Also, the detail of it is nicely provided in the lecture presentation and in Chapter 6 of the text book.
   (c) Robustness: Should be able to check the syntax errors of any possible type. Accurate error message is expected. Support of variable and label to be of any combination of character and numeric and support of range in constant to be considered. Any other additional feature such as *comment line* will be appreciated.
   (d) All the instructions to be considered.
   (e) The assembler would take **.asm** file as input and results in a **.hack** as output.
2. Testing:
   (a) Prepare three .asm files where each file would contain all the instructions from address class, alu class and jump class respectively. Your assembler should be generating a correct .hack for each file. To check the correctness either you may prepare a test file or manually compare it with the correct out of the tool.
   (b) Prepare an erroneous .asm file to check all the possible type of errors that your assembler could detect.
   (c) Finally, pass the assembly program that you have written as part of Assembly programing exercise and test this with inbuilt assembler simulator.
3. 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.

4 Experimental Flow

1. The experimental flow will follow the steps of Design and Test. Your assembler will be compared with the inbuilt one from the N2T simulator.

5 Tools:

- Language: The Nand2Tetris HDL and TSL (test scripting language)
  Refer: Appendix A and B of text book.
- For instruction format refer Chapter 4 and 6 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).

6 Reporting and Evaluation

Prepare a user manual of your assembler. The user manual should contain a description about the feature of your assembler. More detail such as the process of parsing etc is expected.

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