Files

opcodes.docx: describes the opcode executed by the VM part of the project

overview.pdf: this file

parserActions.pdf: describes the actions of the parser.

parserExample.pdf: describes the parsing of a program by the parser, and how the various data structures are filled in

TestCases: a directory containing test programs. For each test program x, the file x.dump will contain the output from the parser dump routine or any error messages generated while parsing x. x.output will contain the output for running the programs. **As of 7/18/2022 the .dump and .output files are not present.**

updates.pdf: gives the date that files in the directory were updated for clarity or to correct mistakes.

vmActions.pdf: describes the VM's execution of the program produced by the parser

vmExample.pdf: describes the execution of a program by the VM, and how the execution of statements affect the various data structures

binin.txt: shows how the VM reads a binary .out file

binout.txt shows how the Parser produces a binary .out file

Project overview

The project will be done in three steps.

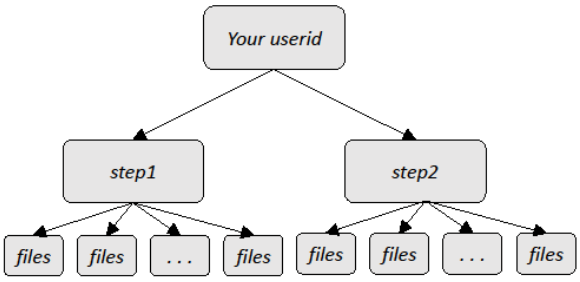
The first step will involve parsing a simple assembly language program for an artificial (virtual) machine. Statements in the language will be of the form: *opcode [operand [operand]]*. The opcode will specify what kind of statement is present, and there will be zero to two operands following it. Once the opcode is determined the number and kind of operands will be known exactly. Thus parsing basically involves getting the first set of characters from the input line, recognizing what kind of statement it is, and then getting the required operands, i.e., very simple parsing.

The input will be test cases that I supply, and the output will be like the supplied solutions in the significant parts. By "*in the significant parts*" I mean the desired information is provided and correct, the format is similar to the solution, but it doesn't have to match the provided input character-for-character. As well, your program should write, to a file, a program that is executable by the program of step 2. If the input file to the parser is *testx*, the output file produced must be *testx.out*.

The second step involves developing a VM that executes a program in the .out file produced by the first step. When the second step is tested, your first step will be executed and the file it produces will be passed to the program of the second step. Grading will be based on the output from executing the program and whether or not errors (segmentation faults, etc.) occur.

The third step has not requirements beyond those of the first two steps, but is a way to allow you to correct errors found during the grading of the second step and get credit for those corrections. If you

get a score that is acceptable to you on Step 2 you do not need to turn in Step 3 – the Step 2 score will be scaled and used for the Step 3 grade.



Projects should be turned in with the directory structure as shown above. The *step2* folder will be empty when turning in the first step, both *step1* and *step2* will be populated when turning in the second step, and both will be populated with your final code when turning in Step 3.

If your project cannot be compiled on a linux machine using g++ `g++ -std=c++11` *.cpp you must include a make file such that *make* will compile your program and produce an executable called comp for the compiler part and vm for the interpreter part.

Partners You may have a single partner for the project. To let me know your partner, send me an email with your partners name, and a subject of "ECE 39595 Partner". Your partner should be copied on the email. For each group only one partner should turn in a project step and both partners will be credited for the work.

Turn-in dates Turn in dates will be posted on Brightspace, and project will be turned in through Brightspace.