Mips Disassembler
Kulneet Singh

Build Instructions:

This code for myDisassembler was written in C++. The source code file is named myDisassembler.cpp, this file can be compiled and built by any C++ 11 compiler. In the command prompt you must change directories in to the folder where the file is contained. Then the compile command along with the source file name should be executed in the command prompt. The example for the Visual Studio 2017 developer command prompt is: “cl /EHsc myDisassembler.cpp”. Then the compiler should compile without errors and create an executable file in the same directory named myDisassembler.exe. In order to run the program, the object file should be in the same directory, the command to run it is illustrated best with and example with test_case1.obj as follows: “myDisassembler test_case1.obj”. This command should execute the program and create a test_case1.s output file (assuming no errors are found in the object file).

High-level Code Description:

The program takes in one command line argument, which is the name of the object file that needs to be disassembled. If there isn’t exactly one command line argument and if the command line argument is not a .obj file, the program prints an error to standard output and then the program is ended. If all those error tests are passed, then the program continues to execute. The first thing that is done that input and output file streams are opened based on the *.obj filename that was passed to the program. Then the first thing that the program does it does one sweep through the program and see is there are any branch instructions in the object code and then the branch targets are calculated and are stored in a list data structure from the standard C++ library. After all of these targets are calculated, a couple algorithms from the list class are used on that list. The first algorithm is the sort algorithm is the sort and then the unique algorithm which essentially just makes sure there is not more than one target label printed even if there are multiple branches that are pointing to one instruction. Then there is a second pass through the object file which is the main pass that decodes all the instructions in the object file. First, before every line is decoded, the program checks if the next line is a branch target, and if it is then a tag with the address is printed. Then, each instruction is read in one at a time, next the instruction is converted to a 32-bit unsigned integer, so that is can be parsed properly. After this is done, then a helper function helps isolate just the opcode to help determine what kind of instruction it is. This opcode is then put into a switch case to determine if the instruction is a R-

type or a I-type instruction. If the instruction is an R-type then there is another helper function that retrieves the function code to determine what function it is and once the correct function is found the function is printed to the output file with the appropriate register names and signed immediate values. The I-type instructions are found just based on opcode. However, if there is an error found with any of the instructions, there is an error statement that is made for each error and it is printed to standard output. After all of the instructions are decoded, then the output file is deleted because there was an error found. If there is no error found, there is an output file that contains the assembly code (one for each instruction), and it is stored in the same folder as the input file and the program executable.