MIPS Disassembler

Phillip Sullivan

High Level Overview

1. My disassembler reads in each input, line by line, as a string and converts the hex number into a 32-bit number.
2. As the program reads in input it converts the 32-bit number into a MIPS instruction. It does this by going through some logic statements.
   a. If the opcode is zero, it must be a R-type instruction, so a it goes into a function lookup function. This takes the 6-bit function code and returns the function name (add, sub, etc.). It does this by using a lookup table.
   b. If the opcode is not zero, the opcode goes into an opcode lookup function which works the same way as the function lookup described above.
3. Once the command is found, the code now finds all register names, immediate values and addresses needed for the current instruction. It compiles them into a string and adds the complete command to a vector.
   a. If the function is R-type it finds necessary, register names using and array where v[i] is the name of register i. It then combines the complete command with register names and instruction into a string and adds it to a vector.
   b. If the function is I-type it finds necessary, register names using and array where v[i] is the name of register i. It also fetches the immediate operand as a signed number for signed operations and as an unsigned number for unsigned operations. It then combines the complete command with register names and instruction into a string and adds it to a vector. NOTE: branch instructions are explained in a few lines.
   c. There is also a stub for J-type instructions. I started writing this but noticed it wasn't needed in the spec.
4. If there is a branch instruction the address of the branch (current instruction + 4 + offset) is added to the instruction as well as being stored into a vector of labels I'll need to tag in the output.
5. Now the disassembler has a vector of all instructions as well as a vector of instructions I need to tag in the output.
6. I loop through the vector of instructions and output to the output file. On each loop, I check if the current instruction needs a label, if it does, I output it before he instruction.

How to Compile

I don't know what operating system you (TA) has installed, so use your favorite c++ compiler to compile myDisassembler.cpp.

You should be able to run the executable from the command line with the command **./myDisassembler.exe [Input file name] .**

Make sure the input file is in the same directory as the executable.