Project Overview: Assembler

Lucy Post & Vincent Li

Description:

The goal of our program, assembler.c, is to translate the assembly-language into machine code (decimal value) and store that value in an output file. First, the program reads in an input file with assembly-language. The program passes through all lines of the code three times. The first time, it counts the number of lines in the input file so that we could know the maximum capacity for the array of labels. The second time, it captured which addresses contained a label and placed the label and its address into an array. The third time, the program starts to translate each line from the beginning of the input file.

During the process of translating each line, the program calls the translateLine function. In this function, it first identifies the opcode and calls a function corresponding to what type the opcode is (r-type, j-type, i-type, or o-type). In those functions, the program separates the line and places each segment of the line into the proper format (according to the type). This is done by using bit shifting and masking.

We wrote tests for our TestSuite by testing each type of error check in our program. In addition, to error checking tests, we tested the example included on our project description and we also tested the opcodes not included in the example (nand, sw, and jalr). We tested for different types of whitespace used, including space and tab.

Challenges:

We spent a lot of time in fragmenting each line of the input file. We overcame this by using a string tokenizer. When we were comparing two strings to determine the opcode, it took us a while to figure out that strings end in a \0 character. Once this knowledge was known, it was an easy fix.

Issues Remaining:

We don’t know of any issues remaining. Our tests passed. An improvement we could make is a more efficient way to store the labels and their values instead of using an array.