Joan Marin-Romero and Nancy Yang

Dr. Joseph Myre

CISC 340

October 15, 2017

Assembler

Directory Structure:

**project1\_john7770\_yang0120**

**assembler**

assembler.c

Makefile

README

**Test Suite**

test1

test2

test3

test4

test5

test6

test7

test8

test9

test10

assembler.c

We created a two-pass assembler. We decided to create two functions, so we could decipher the two passes from each other. In the first pass, we calculated the address for every label. In order to do so, we used a struct to define the {label, location} pairs, and then we used an array of the struct data type to store each pair encountered. Subsequently, we looped through each line of assembly code to fill in the array. First, we split each line into a series of tokens using whitespace as the delimiter. Then, we looked at the first token to determine if there was a label. If there was a label, we did some error checking – we checked for the use of undefined labels (begins with a number, includes a special character, greater than six characters) and duplicate labels. If these checks passed, then the label was stored into the array and the location was set to the appropriate line number. The first pass returns the number of labels in the array so it could be used in the second pass.

In the second pass, we generated the machine code for each line of assembly code. In order to do so, we looped through each line of assembly code, and we split each line into a series of tokens using whitespace as the delimiter. If the first token was not an instruction, we error check to see if the next token is a valid instruction. If the check passes, we move on to the next token to determine which instruction is being used. Depending on the instruction type, there will be different required fields. We also error check each field for undefined labels (not in the label array). Then, we bit shift the values in each field into their appropriate bit location for each instruction type and bitwise OR each section together. For I-type instructions, we also error check to see if the offset is between -32768 and 32767. Lastly, we write the decimal values to the screen or another file.

The main method sets all the global variables we need for the code. The main file where the assembly code is stored in and the file that would be created if the user specifies, numOfLabels is the number of labels in labelTable, where labelTable is all the labels in the assembly code.

The main problem we had was the first pass. We failed many times in trying to get the labels and store their position without changing the file. We tried 4 different data structures and mainly used file in and out until you told us to use string tokens, which made the process easier. Another problem was that sometimes the code would run on our computer but not CISC-hank. We had to review our code and open the debugger many times on both ends to try to get the right results. The second pas we hit a wall in trying to convert the code into a 32-bit unsigned integer. We did not understand the logic at first and tried to get a string that had all the binary parts and convert it to an integer at the end, but it did not work and then used bit shifting.

Makefile

This is the Makefile. It includes a clean rule. The MakeFile compiles the code into the proper files.

test1:

The first test case assesses the format of a label. The assembler should return an error because one of the labels begins with a number.

test2:

The second test case assesses the format of a label. The assembler should return an error because there is a special character in one of the label.

test3:

The third test case assesses the format of a label. The assembler should return an error because there are more than six characters in one of the label.

test4:

The fourth test case assesses if labels are defined. The assembler should return an error because there is a label used in a sw instruction that was not previously defined.

test5:

The fifth test case assesses if labels are duplicated. The assembler should return an error because there is a label that is reused in the label field.

test6:

The sixth test case assesses if the offset value is a valid number. The assembler should return an error because there is an offset value that is greater than 32767.

test7:

The seventh test case assesses if the offset value is a valid number. The assembler should return an error because there is an offset value that is less than -32768.

test8:

The eighth test case assesses if the given instruction is valid. The assembler should return an error because there is an invalid instruction.

test9:

The ninth test case assesses the jalr instruction. The assembler should output the decimal values of each line of instruction.

test10:

The tenth test case assesses the add, lw, beq, noop, and halt instructions. The assembler should output the decimal values of each line of instruction.