CISC 340 – Assembler

James Allender
Mitchell Nelson

October 17, 2018

1 Description:

Project 1 is a simple assembler to assemble the UST-3400 (Rip Van saWinkle) ISA into machine code. The program uses a single procedural C program to read in a given file containing assembly code and assemble that into machine code and either output it to the screen or to a file specified by the user.

2 Overview:

The program takes one or two arguments: an input file alone, or an input file and an output file. Any deviation from this will lead to a failure and exit from the program.

One input example:

```
$./assembler -i [assembler code file]
```

Two input example:

```
$./assembler -i [assembler code file] -o [out file]
```

The program can also have its output redirected to a file with:

```
$./assembler -i [assembler code file] > [out file]
```

The program requires that an input file is given directly after a "-i" flag, or it will fail and close. When an input file is provided, the program checks that the input file exists in main, and if so, it will start processing the file.

The file is read twice by the program. On the first pass, the program iterates over each line and records any labels that it finds and puts them in a hash map with the corresponding line number of the label. The program validates that the provided labels meet the requirement for a valid label.

On the second pass over the file, the program reads in all of the arguments on a line into an array to be used to construct the machine code for an instruction. Each Instruction type has a separate builder to interpret the instruction into a decimal number depending on the unique field layout for that instruction.

In each of these machine code creations, each parameter of an instruction is passed to a parameter handler function that validates it and returns its numeric value. If it is a label, it returns the corresponding decimal value for that label. If it is a string that contains purely digits, it returns the decimal value for that string.

For I type instructions, the machine code builder also calculates the offset for labels. It does this by first getting the corresponding decimal value for the label (provided by the parameter handler function). It then either subtracts the current line number + 1 from that value, if the label is for a location prior to the current line, or subtracts the current line number, if the label location is after the current location.

The assembler then takes the interpreted parameters and packs them into their appropriate position in a 32bit integer. It does this by shifting them an amount specified in the program and then bitwise or'ing all the pieces together. These decimal machine codes generated by the assembler are then printed to the screen or the specified output file.

3 Challenges:

Some challenges that were encountered when implementing the assembler were finding a data structure in C to store the labels and corresponding line addresses. The solution to this was using the Glib library and using the hash table provided in the library. This data structure took some studying from the online Glib API to learn how to implement the hash table. It was also a challenge to figure out how to best cast the input strings into integers, when needed. Not every string needed to be cast because some of the input would be kept as a string and used as a label. The solution to this challenge was to write some helper functions that first check if the input contained only digits using strtol. If the string passed this test, the program would then cast it. Otherwise, it would check the hash table for that string.