Report of Assignment 1

Ang, Chen / 118010009 / L23

The Chinese University of Hong Kong, Shenzhen

Report of Assignment 1

Introduction

The goal of this project was to implement a MIPS-to-machine-code toy assembler. We chose to use C++ as the programming language. The MIPS assembly language (.asm) file was passed in as the input of the program. The file consists of individual lines of MIPS assembly language instructions. Each line was processed by the toy assembler into a 4-byte machine code, which then was stored as a string in the output text (.txt) file.

Implementation

We implement the toy assembler as two major steps.

Step 1. Scanning File and Recording Label

After prompting user for the path of the input file, the toy assembler reads the MIPS file line by line. The assembler first processes the line by stripping off the comments following a '#' character. (Of course, if the line were left with empty constituting only of whitespaces our assembler would ignore the current line and go to the next.) Then the assembler scans label info on the line by searching for ':' character, which indicates the end of the label. If any label has been found, the program would store the name of the label, paired with the current `ln_idx`, into an `unordered_map` (hashmap) called `labels`. After that, the line is further shrunk down by the `strip` function, which strips off whitespaces on the both ends of the line. There are two cases here. If the whitespace-stripped line is longer than an empty string, our program will insert it into a `vector<string>` containing all valid instructions, ready for later assembling. However, if the line is merely an empty string, this means that we have come across a standalone label line. In that case, we do not insert the current line (an empty string) into the instructions vector. We also need to decrement the `ln_idx` by one, since the current line is not really an instruction. This process continues until we have reached the last line of the MIPS code. Note that, crucially, we do not assemble a line of instruction right after reading them from the MIPS file. This is because we may not have obtained the full information of labels until we scanned the entire file. If one line of instruction used a label below the current instruction, we would not have known which line the label is at when assembling.

Step 2. Reading Instructions and Assembling

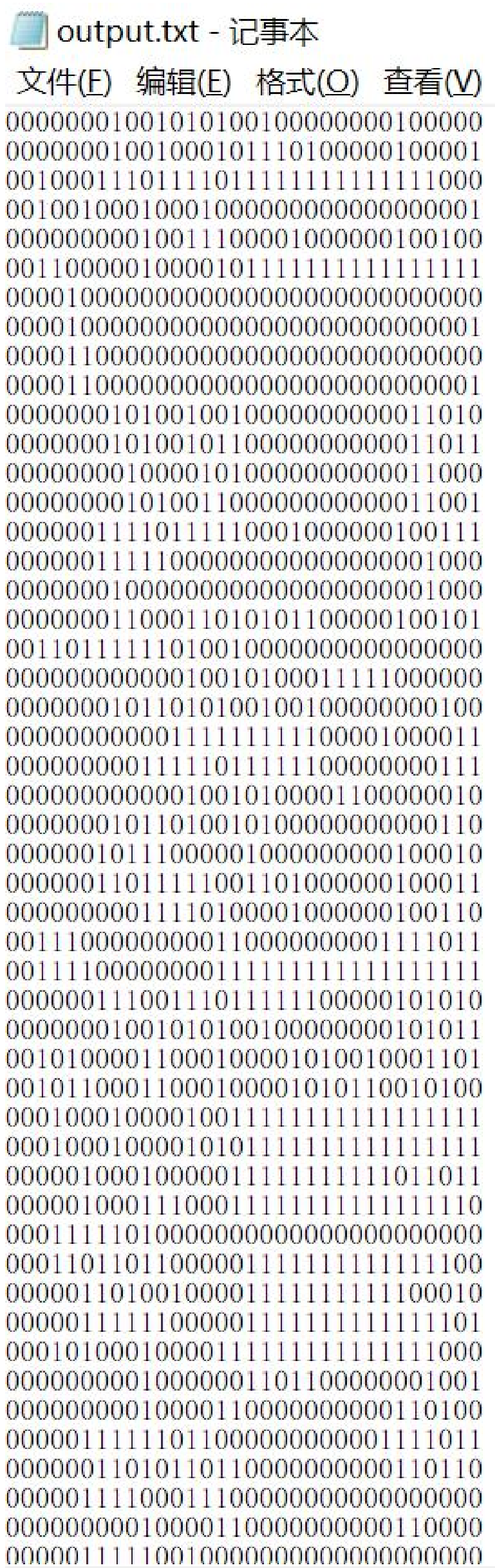
The second step the program does is to retrieve the instructions stored in the vector, one by one, and turn them into machine codes using the `make` function. In this step, the label information stored in the hashmap would be accessed whenever the assembler has found a label in one instruction. We used the `bitset<32>` and `.to_string()` to transform the `int` type machine code into a 4-byte format string, which was stored into the output file.

Sample Output

```
C:\Qt\Qt5.14.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
MIPS ASSEMBLER for CSC3050 Assignment 1

Please specify the absolute path of input file (e.g. /usr/local/input.asm):c:/users/chen1/desktop/input.asm
Save output file where input file is? [Y/N]y
Successfully assembled!
File path: c:/users/chen1/desktop/output.txt
Press <RETURN> to close this window...
```

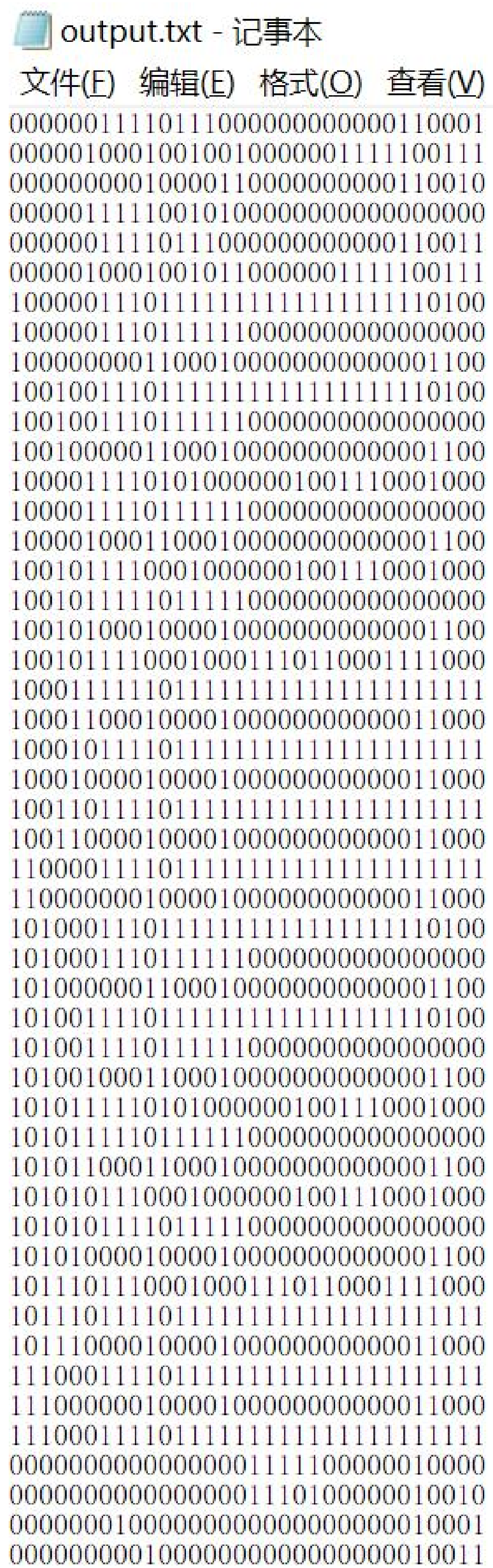
(see next page)



output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

```
00000001001010100100000000100000
00000001001000101110100000100001
00100011101111011111111111111000
00100100010001000000000000000001
00000000010011100001000000100100
00110000010000101111111111111111
00001000000000000000000000000000
00001000000000000000000000000001
00001100000000000000000000000000
00001100000000000000000000000001
00000001010010010000000000011010
00000001010010110000000000011011
00000000100001010000000000011000
00000000101001100000000000011001
00000011110111110001000000100111
00000011111000000000000000001000
00000001000000000000000000001000
00000001100011010101100000100101
00110111111010010000000000000000
00000000000010010100011111000000
00000001011010100100100000000100
00000000000111111111100001000011
00000000011111011111100000000111
00000000000010010100001100000010
00000001011010010100000000000110
00000010111000000100000000100010
00000011011111001101000000100011
00000000011110100001000000100110
00111000000000011000000000111011
00111100000000011111111111111111
00000011100111011111100000101010
00000001001010100100000000101011
00101000011000100001010010001101
00101100011000100001010110010100
00010001000010011111111111111111
00010001000010101111111111111111
00000100010000011111111111011011
00000100011100011111111111111110
00011111010000000000000000000000
00011011011000001111111111111100
00000110100100001111111111100010
00000111111000001111111111111101
00010100010000111111111111111000
00000000010000001101100000001001
00000000010000110000000000110100
00000111111011000000000000111011
00000011010110110000000000110110
00000111100011100000000000000000
00000000010000110000000000110000
00000111110010000000000000000000
```



output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

```
00000011110111000000000000110001
00000100010010010000000111100111
00000000010000110000000000110010
00000111110010100000000000000000
00000011110111000000000000110011
00000100010010110000000111100111
10000011101111111111111111110100
10000011101111110000000000000000
10000000011000100000000000001100
10010011101111111111111111110100
10010011101111110000000000000000
10010000011000100000000000001100
10000111101010000000100111001000
10000111101111110000000000000000
10000100011000100000000000001100
10010111100010000000100111001000
10010111101111110000000000000000
10010100010000100000000000001100
10010111100010001110110001111000
10001111110111111111111111111111
10001100010000100000000000011000
10001011110111111111111111111111
10001000010000100000000000011000
10011011110111111111111111111111
10011000010000100000000000011000
11000011110111111111111111111111
11000000010000100000000000011000
10100011101111111111111111110100
10100011101111110000000000000000
10100000011000100000000000001100
10100111101111111111111111110100
10100111101111110000000000000000
10100100011000100000000000001100
10101111101010000000100111001000
10101111101111110000000000000000
10101100011000100000000000001100
10101011100010000000100111001000
10101011101111110000000000000000
10101000010000100000000000001100
10111011100010001110110001111000
10111011110111111111111111111111
10111000010000100000000000011000
11100011110111111111111111111111
11100000010000100000000000011000
11100011110111111111111111111111
00000000000000011111000000010000
00000000000000011101000000010010
00000001000000000000000000010001
00000000010000000000000000010011
```