Project 1 Report: MIPS Assembler

Author: Xuanyu Ding
SID: 116010036
Date: 2019.3.3

Purpose:
The project takes a file with MIPS assembly code as input, and output the binary assembled result of that file.

Method:
Using a typical divided and conquer methodology, the programmer divided the whole process to two procedure, namely phase1 and phase2. The reason of the division is that for some specific instructions (for example: bne, j), it will call the address to a label in the following line. The program can not fetch that address unless has all the label stored previously. That's also the main reason of creating a data structure to store information of these labels, named LabelTable.

Procedure(shown in figure1):
Phase1
(1) Read in the given MIPS file by line and find all the labels.
(2) Store label name and address to a data structure called ***LabelTable***.
Phase2
(3) Read in the given file again.
(4) Translate every instruction. If there is a call for label address, pass the label name to ***LabelTable*** and fetch the address of it.

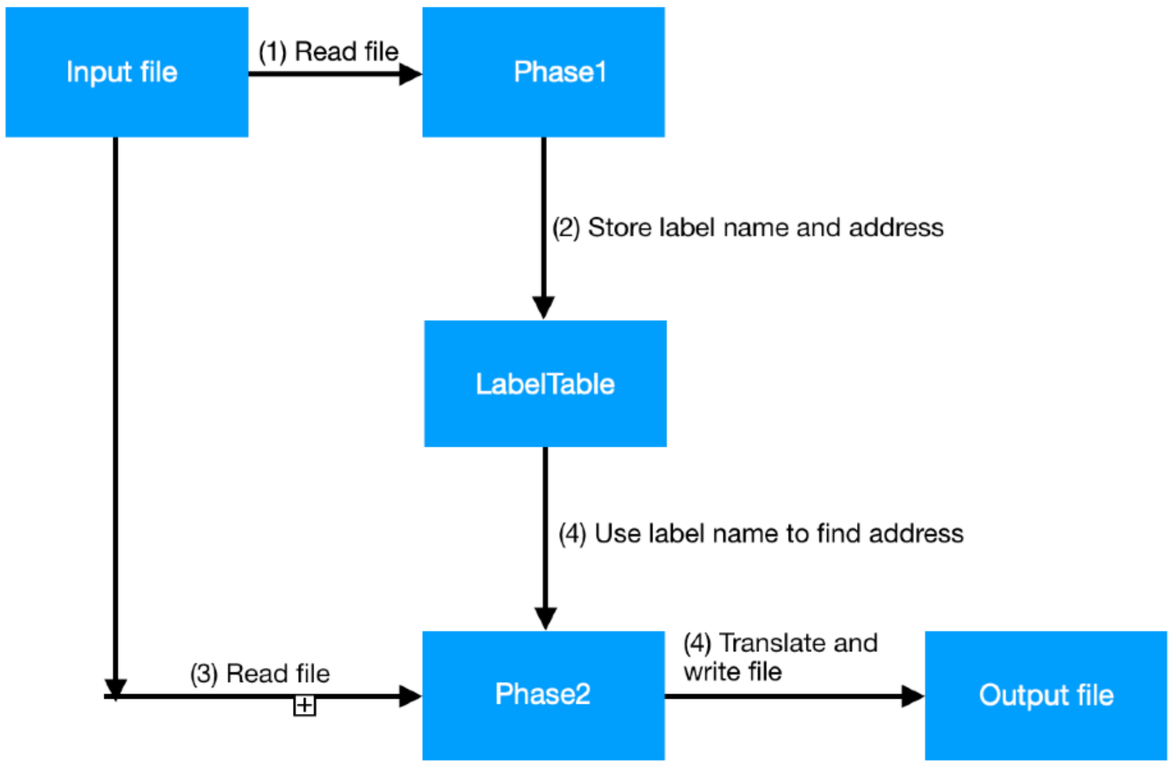
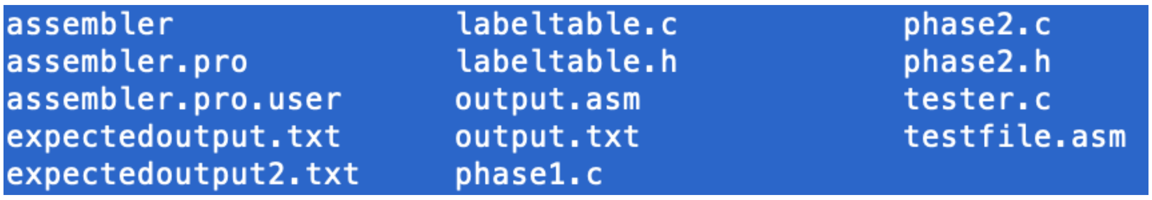

Input file
(1) Read file
Phase1
(2) Store label name and address
LabelTable
(4) Use label name to find address
(3) Read file
Phase2
(4) Translate and write file
Output file

figure 1. Procedure Line of the Assembler

Files and Functions:

As stated before, the project will contain three core source file: **phase1.c, phase2.c** and **labeltable.c**, *figure 2* gives an overview of these files altogether with their header files and test files.



```
assembler                labeltable.c             phase2.c
assembler.pro            labeltable.h             phase2.h
assembler.pro.user       output.asm               tester.c
expectedoutput.txt       output.txt               testfile.asm
expectedoutput2.txt      phase1.c
```

figure 2. Files Overview in Assembler

To further understand how these files works, one have to reach in the function level. Thus all functions implemented has been display here with their purpose. <u>More information could be found in the comments in corresponding files.</u>

Phase1

LabelTable phase1(char *filename);

Read in the file and find all the labels, then pass the name and the address of the label. Take the filename as the parameter and return a LabelTable generated.

void eliminateComments(char *filename);

Eliminate the content after “#". Called in every iteration while the program read file in each line. Call by phase1() and phase2().

LabelTable

It’s a table structure to store entries which contain name and address of labels.

LabelTable *createTable();

Initialize a table. Call by phase1().

int addLabel(LabelTable *table, char *name, int address);

Add a label to a table, including the name and the address.
Call by phase1().

int getAddressFromTable(LabelTable *table, const char *name);

Use the name of a label to get the address of it.
Call by phase2().

Phase2

int phase2(LabelTable *table, char *filename);

Read the file again and translate the file. Then write the result to output.txt.

void translateType(char *name, char **inst, char **args, LabelTable *table);

Translate each line of MIPS instructions in to binary machine code.
Call by phase2().

char *getReg(const char *arg);

Get the corresponding binary code for a Register(e.g. $a0).
Call by translateType().

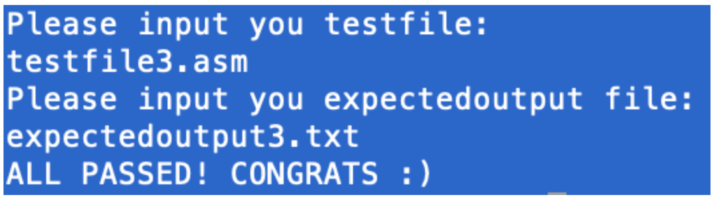
char *getNum(const char *num);

Get the corresponding binary code for a number(0 to 31).
Call by translateType().

char *decimalToBinary(int n, int bit);

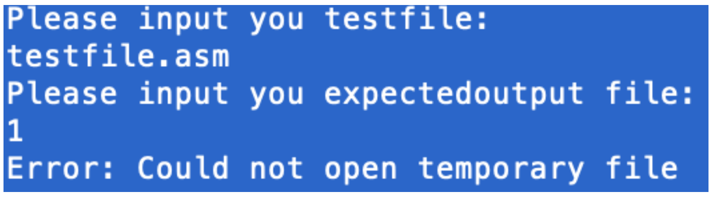
Get a "bit"-bit binary representation of a positive or negative number.
Get more details from comments in phase2.h
Call by translateType().

Test file and Result sample:
The major part of tester.c is remain unchanged, and several changes made for adjustment is declare clearly in comments in tester.c.
After call “gcc tester.c phase1.c phase2.c labelTable.c -o assembler” and “./assembler”, please enter your testfile and eo file correctly following the instruction.
Only if your input file is valid, the test will run:



```
Please input you testfile:
testfile3.asm
Please input you expectedoutput file:
expectedoutput3.txt
ALL PASSED! CONGRATS :)
```

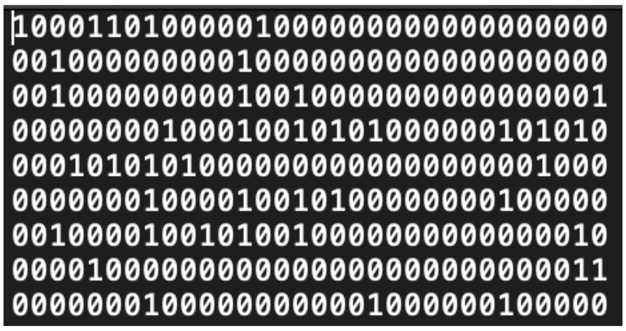
otherwise an error will be generated:



```
Please input you testfile:
testfile.asm
Please input you expectedoutput file:
1
Error: Could not open temporary file
```

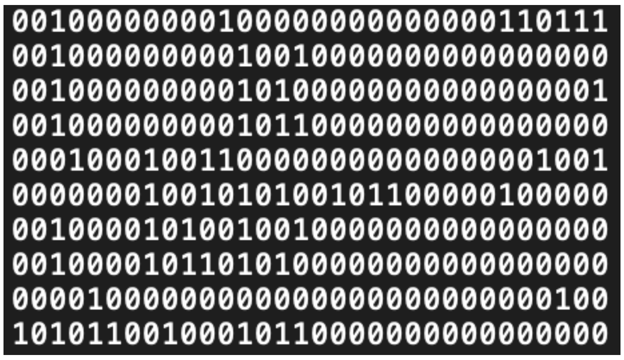
In this case you should call ./assembler again to restart the project.

This is the output of given test file:
PASS all given three test file using gcc and the edited tester.
testfile.asm
output:



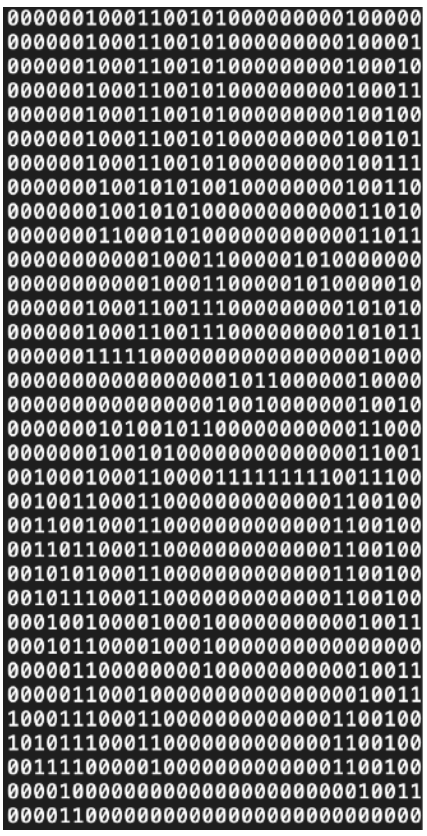
```
10001101000000100000000000000000
00100000000010000000000000000000
00100000000010010000000000000001
00000000100010010101000000101010
00010101010000000000000000001000
00000001000010010100000000100000
00100001001010010000000000000010
00001000000000000000000000000011
00000001000000000001000000100000
```

testfile2.asm
output:



```
00100000000010000000000000110111
00100000000010010000000000000000
00100000000010100000000000000001
00100000000010110000000000000000
00010001001100000000000000001001
00000001001010100101100000100000
00100001010010010000000000000000
00100001011010100000000000000000
00001000000000000000000000000100
10101100100010110000000000000000
```

testfile3.asm
output:



```
00000010001100101000000000100000
00000010001100101000000000100001
00000010001100101000000000100010
00000010001100101000000000100011
00000010001100101000000000100100
00000010001100101000000000100101
00000010001100101000000000100111
00000001001010100100000000100110
00000001001010100000000000011010
00000001100010100000000000011011
00000000000100011000001010000000
00000000000100011000001010000010
00000010001100111000000000101010
00000010001100111000000000101011
00000011111000000000000000001000
00000000000000001011000000010000
00000000000000001001000000010010
00000001010010110000000000011000
00000001001010000000000000011001
00100010001100001111111110011100
00100110001100000000000001100100
00110010001100000000000001100100
00110110001100000000000001100100
00101010001100000000000001100100
00101110001100000000000001100100
00010010000100010000000000010011
00010110000100010000000000000000
00000110000000010000000000010011
00000110001000000000000000010011
10001110001100000000000001100100
10101110001100000000000001100100
00111100000100000000000001100100
00001000000000000000000000010011
00001100000000000000000000000000
```