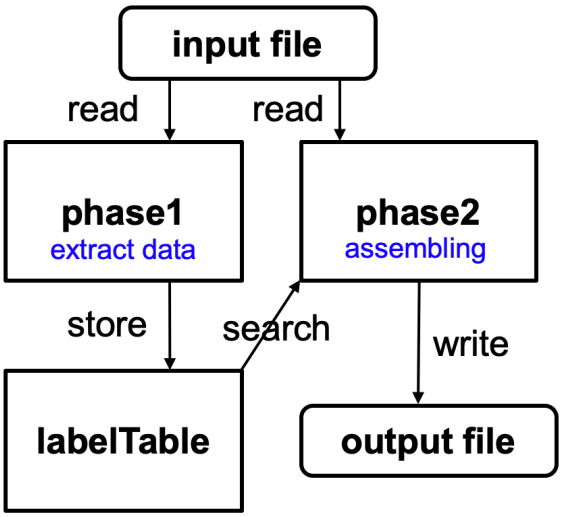
Project Report

1. Objective

The objective of this project is to write an assembler for MIPS assembly language. This assembler can convert the MIPS instructions from input file into machine code and write them on the output file.

2. Structure

As what provided by project instructions, there are three parts in my assembler: “labelTable”, “phase1”, “phase2”. They work together and do the job of assembler. The figure above simply shows how they work:



“phase1”:

1. Read the file of MIPS instructions (skip the blank line);
2. Extract the information of tags —— tag name and tag address;
3. Store the data of tags into label Table;

```
// Scan through the input file, find out where are the tags and store into lableTable.
labelTable phase1(char* filename);
```

“labelTable”:

1. Construct a container to store the data;

```
// The container of a single info of tags.
typedef struct labelInformation
{
    char *label;
    int address;

}labelInfo;

// The container of all tag information.
typedef labelInfo *labelTable;
```

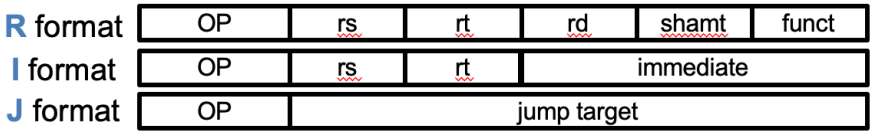
2. Provide a function for user to get the tag address when tag name is known;

```
// Get the char* type of address of the label inputed through labelTable.
char* getAddress(char* label, labelTable table);
```

“phase 2”:

0. Prepare functions for later use, which is shown below

For every single type of instructions(R or I or J), it has its own fixed machine code format：



If we can know the concrete value of its content, we can get machine code right. So we prepare functions for later use:

```
// Write 3 types of instruction into output file in the form of machine code.
void writeR(FILE* file, char* opcode, char*rs, char* rt, char* rd, char* sa, char* function);
void writeI(FILE* file, char* opcode, char* rs, char* rt, char* immediate);
void writeJ(FILE* file, char* opcode, char* target);
```

However, as shown below, although the instructions in the same type has the same machine code format, their MIPS instruction format may still be quite different. For example, in the R type, “add” is followed by “rd”, “rs”, “rt” ; but “jr” is only followed by “rs”. Therefore, I divided instruction into 11 types —— 5 for R type, 5 for I type and 1 for J type, for each type, they have the same order of things like “rs” , “rt”.

```
// According to the different type and format of instruction,
// we divided into 11 types in total.
// THe function bellow is used to identify the type of instruction.
int isRType1(char* str);
int isRType2(char* str);
int isRType3(char* str);
int isRType4(char* str);
int isRType5(char* str);

int isIType1(char* str);
int isIType2(char* str);
int isIType3(char* str);
int isIType4(char* str);
int isIType5(char* str);

int isJType(char* str);

// To grab the key word of instruction like 'add', 'sw', etc.
int isInstruction(char* str);
```

After we get the correct value of the component of instructions like “rs”, “rt”, “immediate”, we need to represent them in binary form, which is what we expect to output.

```
// Convert decimal into binary with different number of bits,
// used for interprete things like label or immediate into binary.
char* dec2bin5(char* decimal);
char* dec2bin16(char* decimal);
char* dec2bin26(char* decimal);

// Convert register symbol into binary code.
char* reg2bin5(char* reg);
```

After defining these function, phase2 can work easier:

1. Read the file of MIPS instructions (skip the blank line);
2. For each valid instruction line, catch its key instruction like “add”, “sw”, etc.

```
char* token = strtok(line, abandoned);
if (!isInstruction(token)){
    token = strtok(NULL, abandoned); // In case we got label.
} // Here we have gotten instruction we will do.
```

3. Check which small type the instruction is, and for instructions of each type, discuss their common features like the number of tokens should be extract, fixed “rs”, etc. Also discussed independent feature like opcode, so that they can correctly write machine code on the output file by the function defined before.

Example:

```
else if (isIType2(token)){
    for (int i = 0; i < 3; i++){
        Instruction[i] = strtok(NULL, abandoned);
    }
    char* opcode;
    if (!strcmp(token, "beq")){opcode = "000100";}
    else {opcode = "000101";}
    writeI(output, opcode, reg2bin5(Instruction[0]), reg2bin5(Instruction[1]),
           dec2bin16(getAddress(Instruction[2], table)));
}
```

4. After read all of the lines, the job is done.

3. Test Result

I put the three “testfile” files and three “expectedoutput” files under the directory of the project, and compile it , and test it. Fortunately, it works.