ICS LABA 实验报告

Name：张展翔

Student Number：PB20111669

代码分析

assembler.h

函数LeftTrim是去除左端空格，RightTrim是去除右端空格，Trim函数则是去除两端空格，

故根据函数功能可以补充函数为：

```
inline std::string &LeftTrim(std::string &s, const char *t = "
\t\n\r\f\v") {
    // TO BE DONE
    for (int i = 0; i < s.length(); i++)
        if (s[0] == '\t' || s[0] == '\n' || s[0] == '\r' ||
s[0] == '\f' || s[0] == '\v')
            s.erase(0);
        else break;
    return s;
}


// trim from right
inline std::string &RightTrim(std::string &s, const char *t = "
\t\n\r\f\v") {
    // TO BE DONE
    for (int i = s.length()-1; i >= 0 ; i--)
        if (*--s.end() == '\t' || *--s.end() == '\n' || *--
s.end() == '\r' || *--s.end() == '\f' || *--s.end() == '\v')
            s.pop_back();
        else break;
```

```
    return s;
}
```

assembler.cpp

主要分析主体函数assemble的功能

- 变量的创建
- 预处理
- 第一遍扫描
- 第二遍扫描
- 关键函数的分析

变量创建部分

```
// store the original string
    std::vector<std::string> file_content;
    std::vector<std::string> origin_file;
    // store the tag for line
    std::vector<LineStatusType> file_tag;
    std::vector<std::string> file_comment;
    std::vector<int> file_address;
    int orig_address = -1;
    std::string line;

    std::ifstream input_file(input_filename);
    std::string comment_str;
    std::string content_str;
```

预处理部分

逐行读取.asm文件，利用Trim函数去除空格，如果是空行就跳过，

如果不为空行，由注释可知，`// Convert line into upper case`指令，故可由std::transform函数来实现:

```
std::transform(line.begin(), line.end(), line.begin(),
::toupper);
```

然后要去注释与汇编语言的分离，故采取寻找';'的方式如果没有找到';'，即`if (comment_position == std::string::npos)`，证明没有注释，故直接将原汇编语句存入即可；若找到了';'，则需将注释与实际汇编语句分割开，然后再去判断此语句的情况。

第一遍扫描

主要涉及.ORIG命令和.END命令的处理，jump指令数值指令和注释的处理

如果改行为注释的话，就跳过

如果该行第一个字符为'.'，让改行标签值定位伪指令

接下来分别分析该伪指令的具体情况(`.ORIG`，`.END`，`.STRINGZ`，`.FILL`，`.BLKW`)

需要我们补充的是`.FILL`和`.BLKW`指令，我们可以仿照剩下三个指令的形式来进行类比填写

```
else if (pseudo_command == ".FILL")
            {
                // TO BE DONE
                file_address[line_index] = line_address;
                line_address++;
                std::string word;
                line_stringstream >> word;
                auto num_temp = RecognizeNumberValue(word);
                if (num_temp ==
std::numeric_limits<int>::max())
                {
                    return -4;
                }
                if (num_temp > 65535 || num_temp < -65536)
                {
                    return -5;
                }
            }
            else if (pseudo_command == ".BLKW")
            {
                // TO BE DONE
                std::string word;
                line_stringstream >> word;
                auto num_temp = RecognizeNumberValue(word);
                if (num_temp ==
std::numeric_limits<int>::max())
                {
                    return -6;
                }
                if (num_temp > 65535 || num_temp < -65536)
                {
                    return -7;
                }
```

```
            }
```

下面则是关于其余指令的处理，由注释可知，该if条件语句即为判断该语句是否带标签，若带则继续进行下一步的处理。

故所填代码应为:

```
  file_tag[line_index] = lOperation;
  continue;
```

如果语句带有跳转和分支标签，则需要如下处理

```
file_tag[line_index] = lOperation;
            label_map.AddLabel(label_name, value_tp(vAddress,
file_address[line_index]));
```

标签后语句为伪指令的情景可以仿照 `.FILL` 语句进行处理

```
if (word == ".BLKW")
            {
                // modify label map
                // modify line address
                // TO BE DONE
                std::string word;
                label_map.AddLabel(label_name,
value_tp(vAddress, line_address - 1));
                line_stringstream >> word;
                auto num_temp = RecognizeNumberValue(word);
                if (num_temp ==
std::numeric_limits<int>::max())
                {
                    return -6;
                }
                if (num_temp > 65535 || num_temp < -65536)
                {
                    return -7;
                }
                line_address += num_temp - 2;
            }
            if (word == ".STRINGZ")
            {
                // modify label map
                // modify line address
                // TO BE DONE
```

```
                label_map.AddLabel(label_name,
value_tp(vAddress, line_address - 1));
                std::string word;
                line_stringstream >> word;
                if (word[0] != '\"' || word.back() != '\"')
                    return -6;
                auto num_temp = word.back() - 1;
                line_address += num_temp - 1;
            }
```

第二遍扫描

首先确定输出名称，若未指定输出名称，则输出格式设置为"输入名称+.bin"

然后开始扫描，若注释行，则跳过本行处理；

然后读取每一行的内容，

如果为伪指令

首先读取该伪指令的内容，

.FILL

把FILL的数值转换为二进制填入输出文件中

.BLKW指令和.STRINGZ指令可以类比.FILL指令填写

代码如下

```
else if (word == ".BLKW")
            {
                // Fill 0 here
                // TO BE DONE
                std::string number_str;
                std::string output_line = "0000000000000000";
                if (gIsHexMode)
                    output_line = ConvertBin2Hex(output_line);
                line_stringstream >> number_str;
                int num = RecognizeNumberValue(number_str);
                for (int i = 0; i < num; i++)
                {
                    output_file << output_line << std::endl;
                }
            }
            else if (word == ".STRINGZ")
```

```
            {
                // Fill string here
                // TO BE DONE
                std::string word;
                line_stringstream >> word;
                for (int i = 1; i < word.size() - 2; i++)
                {
                    auto output_line = NumberToAssemble((int
&)word[i]);
                    if (gIsHexMode)
                        output_line =
ConvertBin2Hex(output_line);
                    output_file << output_line << std::endl;
                }
                std::string output_line = "0000000000000000";
                if (gIsHexMode)
                    output_line = ConvertBin2Hex(output_line);
                output_file << output_line << std::endl;
            }
```

如果为操作指令

首先判断操作指令是否带标签并进行处理

由所需补充代码部分所给注释`// Convert comma into space for splitting`,可知所需补充代码为将','替换为'**\space**'.

故补充代码为`std::replace(parameter_str.begin(), parameter_str.end(), ',', ' ');`

~~接下来就是匹配操作指令与对应的操作码啦！~~

~~太多了就不详细介绍了~~

关键函数的分析

RecognizeNumberValue函数

该函数主要实现将一个字符串转换为一个数字

故需分情况讨论,

- #开头的10进制数字
- X or x开头的16进制数字

- .BLKW指令后的直接的数字

故所填写代码如下

```
int RecognizeNumberValue(std::string s)
{
    //Convert string s into a number
    //TO BE DONE
    if (s[0] == '#')
        return std::stoi(s.substr(1));
    else if (s[0] == 'X' || s[0] == 'x')
        return std::stoi(s.substr(1), 0, 16);
    else if(s[0]<='9'&&s[0]>='0')
        return std::stoi(s);
    else return std::numeric_limits<int>::max();
}
```

NumberToAssemble函数

该函数主要实现了将一个数(可能为int型或string型)转换为一个16位的二进制字符串

易写出如下代码

```
std::string NumberToAssemble(const int &number)
{
    //Convert the number into a 16 bit binary string
    //TO BE DONE
    std::string AssembleNum ;
    int num = number;
    while (num)
    {
        if (num % 2 == 0)
            AssembleNum += '0';
        else
            AssembleNum += '1';
        num /= 2;
    }
    int length=AssembleNum.size();
    for (int i = 0; i < 16 -length; i++)
    {
        AssembleNum += '0';
    }
    std::reverse(AssembleNum.begin(), AssembleNum.end());
    return AssembleNum;
}
std::string NumberToAssemble(const std::string &number)
```

```
{
    // Convert the number into a 16 bit binary string
    // You might use `RecognizeNumberValue` in this function
    // TO BE DONE
    return NumberToAssemble(RecognizeNumberValue(number));
}
```

ConvertBin2Hex函数

该函数实现了将一个二进制字符串转变为一个16进制字符串

函数实现如下:

```
std::string ConvertBin2Hex(std::string bin)
{
    //Convert the binary string into a hex string
    //TO BE DONE
    std::string s = {};
    for (int i = 0; i < 4; i++)
    {
        static const int bin2num[4] = {8, 4, 2, 1};
        int num;
        for (int j = i * 4; j < i * 4 + 4; j++)
        {
            if (bin[j] == 1)
                num += bin2num[j % 4];
        }
        if (num > 9)
            s += num - 10 + 'A';
        else
            s += "" + num;
    }
    return s;
}
```

TanslateOprand函数

该函数主要是将寄存器or立即数or标签翻译为二进制操作码

实现代码如下所示

```
std::string assembler::TranslateOprand(int current_address,
std::string str, int opcode_length)
{
    // Translate the oprand
    str = Trim(str);
```

```
    auto item = label_map.GetValue(str);
    if (!(item.getType() == vAddress && item.getVal() == -1))
    {
        // str is a label
        // TO BE DONE
        int label_address = item.getVal();
        int label_offset = label_address - current_address - 1;
        std::string comple_str;
        for (int i = 0; i < opcode_length - 1; i++)
        {
            int tmp = label_offset & (1<<i);
            comple_str += tmp ? '1' : '0';
        }
        comple_str += (label_offset >= 0 ? '0' : '1');
        std::reverse(comple_str.begin(), comple_str.end());
        return comple_str;
    }
    if (str[0] == 'R')
    {
        // str is a register
        // TO BE DONE
        int register_num = str[1] - '0';
        std::string regis_table[8] = {"000", "001", "010",
"011", "100", "101", "110", "111"};
        return regis_table[register_num];
    }
    else
    {
        // str is an immediate number
        // TO BE DONE
        int num = RecognizeNumberValue(str);
        // to op_length's bits 2's complement
        std::string comple_str;
        for (int i = 0; i < opcode_length - 1; i++)
        {
            int tmp = num & (1<<i);
            comple_str += tmp ? '1' : '0';
        }
        comple_str += (num >= 0 ? '0' : '1');
        std::reverse(comple_str.begin(), comple_str.end());
        return comple_str;
    }
}
```

MAKEFILE的使用

在所需make的文件夹中添加所写代码及Makefile文件，然后执行 `$make` 指令

运行情况如下

```
zzx2002@LAPTOP-MDKNVTK3:/mnt/c/Users/Lenovo/ics2021/labA/assembler$ make
g++ -c -o assembler.o assembler.cpp -I. -g
g++ -o assembler assembler.o main.o -I. -g
```

目前目录如下

```
zzx2002@LAPTOP-MDKNVTK3:/mnt/c/Users/Lenovo/ics2021/labA/assembler$ tree
.
├── Makefile
├── assembler
├── assembler.cpp
├── assembler.h
├── assembler.o
├── fib.asm
├── main.cpp
└── main.o
```

fib.asm为此前实验所写的汇编代码，现执行 `$./assembler -f fib.asm`

```
zzx2002@LAPTOP-MDKNVTK3:/mnt/c/Users/Lenovo/ics2021/labA/assembler$ ./assembler -f fib.asm
zzx2002@LAPTOP-MDKNVTK3:/mnt/c/Users/Lenovo/ics2021/labA/assembler$ tree
.
├── Makefile
├── assembler
├── assembler.cpp
├── assembler.h
├── assembler.o
├── fib.asm
├── fib.bin
├── main.cpp
└── main.o
```

执行后生成fib.bin文件

原fib.asm和fib.bin文件代码如下，符合处理结果

```
.ORIG x3000
ADD R1,R1,#1
ADD R2,R2,#1
ADD R3,R3,#2
ADD R0,R0,#-2
BRz OUTZ
BRn OUTN
LOOP ADD R1,R1,R1
```

```
ADD R4,R1,R3
ADD R1,R2,#0
ADD R2,R3,#0
ADD R3,R4,#0
ADD R0,R0,#-1
BRp LOOP
OUTZ LD R5,MOD
AND R7,R5,R3
TRAP x25
OUTN ADD R7,R7,#1
TRAP x25
MOD .FILL x03FF
F20 .FILL #930
F11 .FILL #246
F16 .FILL #386
F69 .FILL #454
.END
```

```
0001001001100001
0001010010100001
0001011011100010
0001000000111110
0000010000001000
0000100000001010
0001001001000001
0001100001000011
0001001010100000
0001010011100000
0001011100100000
0001000000111111
0000001111111001
0010101000000100
0101111101000011
1111000000100101
0001111111100001
1111000000100101
0000001111111111
0000001110100010
0000000011110110
0000000110000010
0000000111000110
```