浙江大学

本科实验报告

课程名称:	计算机组成
姓 名:	臧可
学 院:	计算机科学与技术学院
系:	计算机科学与技术系
专 业:	计算机科学与技术
学 号:	3180102095
指导教师:	姜晓红

2020年　4月　29日

一、 实验目的和要求

• Supporting the following 18 MIPS instructions: Add, Sub, And, Or, Addi, Ori, Sll，Srl，Lw, Sw, Lui, Slt, Slti, Beq, Bne, J, Jal, Jr.

• input a MIPS assembly program, output its MIPS machine code in hexadecimal code.

• Input a MIPS machine code in hexadecimal code, output its MIPS assembly program with PC value.

二、 实验内容和原理

1. class

mips: show assembler operation

```
class mips
{
public:
    mips(string file1, string file2);
private:
    struct Jump {//Record jump instructions
        int locate;//Number of lines where the instruction is located
        string type;//Instruction name
        string Label;//Label name
    }jump[100];
    int line;//File lines
    int structsize;//Number of jump instructions
    ofstream outfile;
```

```
14
15      void opToHex(string op);//mips command to hexadecimal
16      string toLower(string s);//Uppercase to lowercase
17      int* getReg(char* p);//Uppercase to lowercase
18      int* iToBin(int reg, int width);//int to binary
19      int* iToBinOff(int reg, int width);//Calculate the offset
20      int* numToBin(char* p);//address behind J_type to binary
21      char* binToHex(int* b);//32-bit binary to hexadecimal
22  };
```

Invert: show disassembler operation

```
1   class Invert
2   {
3   public:
4       Invert(string file);//Read instruction file
5       void Convert(string file1, string file2);
6   private:
7       struct ins {//Store instruction
8           string type;//Instruction type
9           string op;//Instruction opcode
10          string name;//Instruction name
11      };
12      vector<ins> R_type, I_type, J_type;//Store 3 kinds of instructions
    separately
13      string findName(const vector<ins>& v, string s);//Find the
    corresponding command name
14      int binToInt(string s);//Binary to int
15      static bool cmp(ins a, ins b);//Used for instruction queue sorting
16      string hexToBin(string h);//4-bit hexadecimal to 16-bit binary
17      string iToSymbol(int t);//int to register symbol
18      char* iToHex(int reg);//int to 8-bit hexadecimal
19      char* binToHex(int* b);//32-bit binary to 8-bit hexadcimal
20  };
```

lineProcess: Realize the removal of spaces and comments on files

```
1   class lineProcess
2   {
3   private:
4       string str;//the input line
5       string comment_str;//Comment string
6   public:
7       lineProcess(string& s, string c);
8       string Cut();//Cut unnecessary parts
9   };
```

2. Main function

```
1   #include "lineProcess.h"
```

```
#include "mips.h"
#include "Invert.h"

void Assembler();
void Dissembler();

int main() {
    int AsOrDiss;
    cout << "Choose assembler or dissembler, 1 is assembler and 0 is dissembler" << endl;
    cin >> AsOrDiss;
    if (AsOrDiss) Assembler();
    else Dissembler();

    return 0;
}

void Assembler() {
    string filename;
    cout << "Please input the name of the file you want to open." << endl;
    cin >> filename;
    mips a(filename, "mips.coe");
    cout << "You can find the answer in the file \"mips.coe\"." << endl;
}

void Dissembler() {
    string filename;
    cout << "Please input the name of the file you want to open." << endl;
    cin >> filename;
    Invert d("instruction.txt");
    d.Convert(filename, "invert.txt");
    cout << "You can find the answer in the file \"invert.txt\"." << endl;
}
```

3. .cpp file

mips.cpp

```
#include "mips.h"
#include "lineProcess.h"

mips::mips(string file1, string file2) {
    ifstream infile(file1);
    outfile.open(file2);
    int i, j;
    string str;

    if (!infile) {
        cout << "Open error!" << endl;
        exit(1);
    }
    outfile << "memory_initialization_radix=16;" << endl;
    outfile << "memory_initialization_vector = " << endl;
    cnt = 0;

```

```
    i = -1;
    line = -1;

    while (getline(infile, str))//Traverse the file for the first time, record the jump instruction  and label information
    {
        lineProcess linep(str, "//");//Remove multiple spaces and tabs at the beginning and end of lines

        str = linep.Cut();
        if (str.empty()) continue;//continue if line is empty
        line++;
        istringstream iss(str);
        string op1, op2, op3, op4;
        string::size_type position;
        iss >> op1;
        position = op1.find(":");
        if (op1 == "j" || op1 == "jal") {
            iss >> op2;
            i++;
            jump[i].locate = line;
            jump[i].type = op1;
            jump[i].Label = op2;
        }
        else if (op1 == "beq" || op1 == "bne") {
            iss >> op2 >> op3 >> op4;
            i++;
            jump[i].locate = line;
            jump[i].type = op1;
            jump[i].Label = op4;
        }
        else if (position != op1.npos) {
            op1.erase(position);
            i++;
            jump[i].locate = line;
            jump[i].type = "tab";
            jump[i].Label = op1;
            line--;
        }

    }
    structsize = i;

    infile.clear();
    infile.seekg(0, ios::beg);
    i = 0;
    line = -1;
    while (getline(infile, str)) {
        lineProcess linep(str, "//");//Remove multiple spaces and tabs at the beginning and end of lines
        str = linep.Cut();
        if (str.empty()) continue;//continue if line is empty
        istringstream iss(str);
        string op;
        iss >> op;
        if (op.back() == ':') continue;//Skip when reading label
        line++;
        opToHex(str);
```

```
    }

    infile.close();
}

void mips::opToHex(string op) {//mips command to hexadecimal
    int i, j;
    string p0;
    string p1;
    char* p2 = new char[10];
    int* nowbin = new int[32];//Record the 32-bit binary machine code corresponding to the mips instruction
    char* nowhex = new char[8];//Record the 8-digit hexadecimal machine code corresponding to the mips instruction
    istringstream iss(op);
    op = toLower(op);
    iss >> p0;
    for (i = 0; i < 32; ++i)
        nowbin[i] = 0;
    if (p0 == "add" || p0 == "sub" || p0 == "and" || p0 == "or" || p0 == "slt" || p0 == "nor") {
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rd = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rs = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        for (i = 6; i < 11; ++i) {
            nowbin[i] = rs[i - 6];
            nowbin[i + 5] = rt[i - 6];
            nowbin[i + 10] = rd[i - 6];
        }
        nowbin[26] = 1;
        if (p0 == "sub")
            nowbin[30] = 1;
        else if (p0 == "and")
            nowbin[29] = 1;
        else if (p0 == "or")
            nowbin[29] = nowbin[31] = 1;
        else if (p0 == "slt")
            nowbin[28] = nowbin[30] = 1;
        else if (p0 == "nor")
            nowbin[29] = nowbin[30] = nowbin[31] = 1;

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "sll" || p0 == "srl") {
        iss >> p1;
        strcpy(p2, p1.c_str());
```

```
        int* rd = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int t = atoi(p2);
        int* shamt = iToBin(t, 5);
        for (i = 11; i < 16; ++i) {
            nowbin[i] = rt[i - 11];
            nowbin[i + 5] = rd[i - 11];
            nowbin[i + 10] = shamt[i - 11];
        }
        if (p0 == "srl")
            nowbin[30] = 1;

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "addi" || p0 == "ori" || p0 == "slti") {
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rs = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* imm = numToBin(p2);
        if (p0 == "addi")
            nowbin[2] = 1;
        else if (p0 == "ori")
            nowbin[0] = nowbin[2] = nowbin[4] = nowbin[5] = 1;
        else if (p0 == "slti")
        {
            nowbin[2] = nowbin[4] = 1;
        }
        for (i = 6; i < 11; ++i) {
            nowbin[i] = rs[i - 6];
            nowbin[i + 5] = rt[i - 6];
        }
        for (i = 16; i < 32; ++i)
            nowbin[i] = imm[i - 16];


        nowhex = binToHex(nowbin);
        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "beq" || p0 == "bne") {
```

```
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rs = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        iss >> p1;
        int offset;
        for (j = 0; j <= structsize; ++j) {
            if (jump[j].locate == line && jump[j].type == p0)
                break;
        }
        for (i = 0; i <= structsize; ++i) {
            if (jump[i].Label == jump[j].Label && jump[i].type == "tab") {
                offset = jump[i].locate - line - 1;
                offset *= 4;
                break;
            }
        }
        int* imm = iToBinOff(offset, 16);//Calculate the offset

        nowbin[3] = 1;
        if (p0 == "bne")
            nowbin[5] = 1;

        for (i = 6; i < 11; ++i) {
            nowbin[i] = rs[i - 6];
            nowbin[i + 5] = rt[i - 6];
        }
        for (i = 16; i < 32; ++i)
            nowbin[i] = imm[i - 16];

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "lui") {
        nowbin[2] = nowbin[3] = nowbin[4] = nowbin[5] = 1;
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* imm = numToBin(p2);
        for (i = 11; i < 16; ++i)
            nowbin[i] = rt[i - 11];
        for (i = 16; i < 31; ++i)
            nowbin[i] = imm[i - 16];

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
```

```
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "sw" || p0 == "lw") {
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rt = getReg(p2);
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* imm;
        if (*p2 == '(')
            imm = iToBin(0, 16);
        else {
            i = 0;
            int t, reg = 0;
            while (p2[i] != '(') {
                t = p2[i] - '0';
                reg = reg * 10 + t;
                i++;
            }
            imm = iToBin(reg, 16);
        }
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rs = getReg(p2);
        nowbin[0] = nowbin[4] = nowbin[5] = 1;
        if (p0 == "sw") nowbin[2] = 1;
        for (i = 6; i < 11; ++i) {
            nowbin[i] = rs[i - 6];
            nowbin[i + 5] = rt[i - 6];
        }
        for (i = 16; i < 32; ++i)
            nowbin[i] = imm[i - 16];

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else if (p0 == "jr") {
        iss >> p1;
        strcpy(p2, p1.c_str());
        int* rs = getReg(p2);
        for (i = 6; i < 11; ++i)
            nowbin[i] = rs[i - 6];
        nowbin[28] = 1;

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
```

```
    else if (p0 == "j" || p0 == "jal") {
        int offset;
        for (j = 0; j <= structsize; ++j) {
            if (jump[j].locate == line && jump[j].type == p0)
                break;
        }
        for (i = 0; i <= structsize; ++i) {
            if (jump[i].Label == jump[j].Label && jump[i].type == "tab") {
                offset = jump[i].locate;
                offset *= 4;
                break;
            }
        }
        int* addroff = iToBin(offset, 32);//Calculate pseudo absolute address
        nowbin[4] = 1;
        if (p0 == "jal")
            nowbin[5] = 1;
        for (i = 6; i < 32; ++i)
            nowbin[i] = addroff[i - 2];

        nowhex = binToHex(nowbin);

        for (i = 0; i < 8; ++i) {
            outfile << nowhex[i];
        }
        outfile << ", ";
        cnt++;
    }
    else {//Errors when entering illegal instructions
        cout << "Error input!" << endl;
        exit(1);
    }
    if (cnt % 8 == 0)
        outfile << endl;
}

string mips::toLower(string s) {
    int len = s.size();
    for (int i = 0; i < len; i++) {
        if (s[i] >= 'A' && s[i] <= 'Z') {
            s[i] += ('a' - 'A');
        }
    }
    return s;
}

int* mips::getReg(char* p) {
    int reg = 0;
    char* pt = p;

    if (*pt == 'a') {
        pt++;
        if (*pt == 't')
            reg = 1;
        else {
            reg = atoi(pt) + 4;
        }
```

```
359     }
360     else if (*pt == 'v') {
361         pt++;
362         reg = atoi(pt) + 2;
363     }
364     else if (*pt == 't') {
365         pt++;
366         reg = atoi(pt);
367         if (reg <= 7)
368             reg += 8;
369         else reg += 16;
370     }
371     else if (*pt == 's') {
372         pt++;
373         if (*pt == 'p')
374             reg = 29;
375         else {
376             reg = atoi(pt) + 16;
377         }
378     }
379     else if (*pt == 'k') {
380         pt++;
381         reg = atoi(pt) + 26;
382     }
383     else if (*pt == 'z')
384         reg = 0;
385     else if (*pt == 'g')
386         reg = 28;
387     else if (*pt == 'f')
388         reg = 30;
389     else if (*pt == 'r')
390         reg = 31;
391     else {
392         cout << "Error input!" << endl;
393         exit(1);
394     }
395     int* q = iToBin(reg, 5);
396 
397     return q;
398 }
399 
400 int* mips::iToBin(int reg, int width) {
401     int* q = new int[width];
402     for (int i = width - 1; i >= 0; --i) {
403         q[i] = reg % 2;
404         reg = reg / 2;
405     }
406     return q;
407 }
408 
409 int* mips::iToBinOff(int reg, int width) {
410     int sign = 0;
411     int i;
412     if (reg < 0) {
413         reg *= (-1);
414         sign = 1;
415     }
416     int* q = new int[width + 2];
```

```
    for (i = width + 1; i >= 0; --i) {
        q[i] = reg % 2;
        reg = reg / 2;
    }
    int* qt = new int[width];
    for (i = 0; i < width; ++i)
        qt[i] = q[i];
    if (sign) {
        for (i = 0; i < width; ++i) {
            if (qt[i]) qt[i] = 0;
            else qt[i] = 1;
        }
        qt[width - 1]++;
        for (i = width - 1; i >= 0; --i) {
            if (qt[i] == 1)
                break;
            qt[i] = 0;
            if (i >= 1) {
                qt[i - 1]++;
            }
        }
    }
    return qt;
}

int* mips::numToBin(char* p) {
    int i = 0;
    int j = -1;
    int k = 0;
    int t;
    int reg;
    int* q = new int[16];
    char* pt = p;
    if (pt[1] == 'x' || pt[1] == 'X') {//If expressed in hexadecimal
        //pt[2]~pt[5]
        for (i = 2; i < 6; ++i) {
            if (isupper(pt[i]))
                pt[i] += ('a' - 'A');
            if (islower(pt[i]))
                t = pt[i] - 'a' + 10;
            else if (isdigit(pt[i]))
                t = pt[i] - '0';
            for (j = 4 * (i - 1) - 1; j >= 4 * (i - 2); --j) {
                q[j] = t % 2;
                t = t / 2;
            }
        }
    }
    else {
        i = 0;
        if (pt[0] == '-')
            i++;
        t = reg = 0;
        k = 1;
        while (pt[i] != '\0') {
            t = pt[i] - '0';
            reg = reg * 10 + t;
            i++;
```

```
            if (pt[i] == '/')
                break;
        }
        q = iToBin(reg, 16);
        if (*p != '-')
            return q;
        for (i = 0; i < 16; ++i) {//Negate
            if (q[i]) q[i] = 0;
            else q[i] = 1;
        }
        q[15]++;
        for (i = 15; i >= 0; --i) {//+1
            if (q[i] == 1) break;
            if (q[i] == 2) {
                q[i] = 0;
                if (i != 0)
                    q[i - 1]++;
            }
        }
    }

    return q;
}

char* mips::binToHex(int* b) {
    int i, j;
    int t, k, reg;
    char* q = new char[8];
    for (i = 0; i < 32; i += 4) {
        t = reg = 0;
        k = 1;
        for (j = 3; j >= 0; --j)
        {
            t = k * b[i + j];
            reg += t;
            k *= 2;
        }
        if (reg <= 9)
            q[i / 4] = reg + '0';
        else
            q[i / 4] = reg - 10 + 'A';
    }
    return q;
}
```

Invert.cpp

```
#include "Invert.h"
#include "lineProcess.h"

Invert::Invert(string path) {
    ifstream infile(path);
    ins tmp;
    while (!infile.eof()) {
        infile >> tmp.type >> tmp.name >> tmp.op;
```

```
        if (tmp.type == "R")
            R_type.push_back(tmp);
        else if (tmp.type == "I")
            I_type.push_back(tmp);
        else
            J_type.push_back(tmp);
    }
    infile.close();
    sort(R_type.begin(), R_type.end(), cmp);
    sort(I_type.begin(), I_type.end(), cmp);
    sort(J_type.begin(), J_type.end(), cmp);
}

void Invert::Convert(string file1, string file2) {
    ifstream infile(file1);
    ofstream outfile(file2);
    string str;
    if (!infile) {
        cout << "Open error!" << endl;
        exit(1);
    }
    while (getline(infile, str)) {
        lineProcess linep(str, ";");//Remove multiple spaces and tabs at the beginning and end of lines
        str = linep.Cut();
        if (str.empty()) continue;//continue if line is empty
        if (!(str[0] >= '0' && str[0] <= '9') && !(str[0] >= 'a' && str[0] <= 'f')
            && !(str[0] >= 'A' && str[0] <= 'F'))
            continue;
        istringstream iss(str);
        string s;
        while (iss >> s) {//Read each machine code in the line
            string newline = hexToBin(s);//Hexadecimal machine code to binary
            string op = newline.substr(0, 6);
            if (op == "000000") {//Determine the type according to the op value
                string instructor_name = findName(R_type, newline.substr(26, 6));
                string rs, rt, rd, shamt;
                string rss, rts, rds, shamts;
                int rsi, rti, rdi, shamti;
                rs = newline.substr(6, 5);
                rt = newline.substr(11, 5);
                rd = newline.substr(16, 5);
                shamt = newline.substr(21, 5);
                rsi = binToInt(rs);
                rti = binToInt(rt);
                rdi = binToInt(rd);
                outfile << instructor_name << " $";
                if (instructor_name == "sll" || instructor_name == "srl")
                    outfile << iToSymbol(rdi) << ",$" << iToSymbol(rti) << "," << binToInt(shamt) << endl;
                else if (instructor_name == "jr")
                    outfile << iToSymbol(rsi) << endl;
                else
```

```
                    outfile << iToSymbol(rdi) << ",$" << iToSymbol(rsi) <<
",$" << iToSymbol(rti) << endl;
            }
            else if (op == "000010" || op == "000011") {
                string instructor_name = findName(J_type, op);
                string address = newline.substr(6, 26);
                int addri = binToInt(address);
                char* addrh;
                addrh = iToHex(addri);
                outfile << instructor_name << " 0x";
                for (int i = 0; i < 8; ++i)
                    outfile << addrh[i];
                outfile << endl;
            }
            else {
                string instructor_name = findName(I_type, op);
                string rs, rt, immediate;
                int rsi, rti;
                rs = newline.substr(6, 5);
                rt = newline.substr(11, 5);
                rsi = binToInt(rs);
                rti = binToInt(rt);
                immediate = newline.substr(16, 16);
                outfile << instructor_name << " $";
                if (instructor_name == "lui")
                    outfile << iToSymbol(rti) << "," <<
binToInt(immediate) << endl;
                else if (instructor_name == "lw" || instructor_name ==
"sw")
                    outfile << iToSymbol(rti) << "," <<
binToInt(immediate) << "($" << binToInt(rs) << ")" << endl;
                else if (instructor_name == "beq" || instructor_name ==
"bne")
                    outfile << iToSymbol(rsi) << ",$" << iToSymbol(rti) <<
"," << binToInt(immediate) << endl;
                else
                    outfile << iToSymbol(rti) << ",$" << iToSymbol(rsi) <<
"," << binToInt(immediate) << endl;
            }
        }
    }
}

bool Invert::cmp(ins a, ins b) {
    return a.op < b.op;
}

string Invert::findName(const vector<ins>& v, string s) {
    int start = 0, end = v.size() - 1;
    while (start <= end) {
        int mid = (start + end) / 2;
        if (v[mid].op == s)
            return v[mid].name;
        else if (v[mid].op < s)
            start = mid + 1;
        else
            end = mid - 1;
    }
```

```
    return "";
}

int Invert::binToInt(string s) {
    int res = 0;
    for (int i = 0; i < s.length(); ++i) {
        res = res * 2 + (s[i] - '0');
    }
    return res;
}

string Invert::hexToBin(string h) {
    int i, j;
    int t;
    int* b = new int[32];
    char* bc = new char[32];
    string bs;
    for (i = 0; i < 8; ++i) {
        if (isupper(h[i]))
            h[i] += ('a' - 'A');
        if (islower(h[i]))
            t = h[i] - 'a' + 10;
        else if (isdigit(h[i]))
            t = h[i] - '0';
        for (j = 4 * (i + 1) - 1; j >= 4 * i; --j) {
            b[j] = t % 2;
            t = t / 2;
            bc[j] = b[j] + '0';
        }
    }
    bs = bc;
    return bs;
}

string Invert::iToSymbol(int t)
{
    string name;
    switch (t)
    {
    case(0):
        name = "zero";
        break;
    case(1):
        name = "at";
        break;
    case(2):
        name = "v0";
        break;
    case(3):
        name = "v1";
        break;
    case(4):
        name = "a0";
        break;
    case(5):
        name = "a1";
        break;
    case(6):
```

```
        name = "a2";
        break;
    case(7):
        name = "a3";
        break;
    case(8):
        name = "t0";
        break;
    case(9):
        name = "t1";
        break;
    case(10):
        name = "t2";
        break;
    case(11):
        name = "t3";
        break;
    case(12):
        name = "t4";
        break;
    case(13):
        name = "t5";
        break;
    case(14):
        name = "t6";
        break;
    case(15):
        name = "t7";
        break;
    case(16):
        name = "s0";
        break;
    case(17):
        name = "s1";
        break;
    case(18):
        name = "s2";
        break;
    case(19):
        name = "s3";
        break;
    case(20):
        name = "s4";
        break;
    case(21):
        name = "s5";
        break;
    case(22):
        name = "s6";
        break;
    case(23):
        name = "s7";
        break;
    case(24):
        name = "t8";
        break;
    case(25):
        name = "t9";
```

```
        break;
    case(26):
        name = "k0";
        break;
    case(27):
        name = "k1";
        break;
    case(28):
        name = "gp";
        break;
    case(29):
        name = "sp";
        break;
    case(30):
        name = "fp";
        break;
    case(31):
        name = "ra";
        break;
    default:
        break;
    }
    return name;
}

char* Invert::iToHex(int reg) {
    int* q = new int[32];
    char* h = new char[8];
    for (int i = 31; i >= 0; --i) {
        q[i] = reg % 2;
        reg = reg / 2;
    }
    h = binToHex(q);
    return h;
}

char* Invert::binToHex(int* b) {
    int i, j;
    int t, k, reg;
    char* q = new char[8];
    for (i = 0; i < 32; i += 4) {
        t = reg = 0;
        k = 1;
        for (j = 3; j >= 0; --j)
        {
            t = k * b[i + j];
            reg += t;
            k *= 2;
        }
        if (reg <= 9)
            q[i / 4] = reg + '0';
        else
            q[i / 4] = reg - 10 + 'A';
    }
    return q;
}
```

lineprocess.cpp

```cpp
#include "lineProcess.h"

lineProcess::lineProcess(string& s, string c)
{
    str = s;
    comment_str = c;
};

string lineProcess::Cut()
{
    for (char& c : str)
    {
        //Unify into spaces
        if (c == '\t' || c == ',' || c == ';' || c == '\r' || c == '\n' || c == '$')
            c = ' ';
    }

    str.erase(0, str.find_first_not_of(" "));//Remove head-of-line spaces
    str.erase(str.find_last_not_of(" ") + 1);//Remove end-of-line spaces

   //Find the position of the comment character, if it does not exist, get string :: npos
    int n_comment_start = str.find_first_of(comment_str);
    if (n_comment_start != string::npos)
        str.erase(n_comment_start);         //Delete comment

    return str;
}
```

三、 实验过程和数据记录及结果分析

1. Assembler

When we input 1, run the assembler. Then we input the name of the file we want to open.

We can find the answer in the file "mips.coe".

```
Choose assembler or dissembler, 1 is assembler and 0 is dissembler
1
Please input the name of the file you want to open.
scpu_demo_2018.asm
You can find the answer in the file "mips.coe".
```

the output is as followers

```
memory_initialization_radix=16;
memory_initialization_vector =
08000008, 00000020, 00000020, 00000020, 00000020, 00000020, 00000020,
00000020,
3C03F000, 00032040, 3C088000, 2014003F, 00083102, 00C33020, 00000827,
0020102A,
202AFFFF, AC660004, 8C650000, 00A52820, 00A52820, AC650000, 21290001,
AC890000,
8C0D0014, 8C650000, 00A52820, 00A52820, AC650000, 8C650000, 00A85824,
21AD0001,
15A00001, 0C000038, 8C650000, 20120008, 0252B020, 02569020, 00B25824,
11600005,
11720009, 20120008, 1172000A, AC890000, 08000019, 15410002, 00005027,
014A5020,
AC8A0000, 08000019, 8E290060, AC890000, 08000019, 8E290020, AC890000,
08000019,
8C0D0014, 014A5020, AD4A0001, 22310004, 02348824, 01224820, 15210001,
21290005,
8C650000, 00A55820, 016B5820, AC6B0000, AC660004, 03E00008,
```

2. Dissembler

When we input 0, run the dissembler. Then we input the name of the file we want to open.

We can find the answer in the file "invert.txt".

```
Choose assembler or dissembler, 1 is assembler and 0 is dissembler
0
Please input the name of the file you want to open.
scpu_demo_2018.coe
You can find the answer in the file "invert.txt".
```

the output is as followers.

```
j 0x00000008
add $zero,$zero,$zero
add $zero,$zero,$zero
add $zero,$zero,$zero
add $zero,$zero,$zero
add $zero,$zero,$zero
add $zero,$zero,$zero
add $zero,$zero,$zero
lui $v1,61440
sll $a0,$v1,1
lui $t0,32768
addi $s4,$zero,63
srl $a2,$t0,4
add $a2,$a2,$v1
nor $at,$zero,$zero
slt $v0,$at,$zero
addi $t2,$at,65535
sw $a2,4($3)
lw $a1,0($3)
add $a1,$a1,$a1
add $a1,$a1,$a1
sw $a1,0($3)
```

```
addi $t1,$t1,1
sw $t1,0($4)
lw $t5,20($0)
lw $a1,0($3)
add $a1,$a1,$a1
add $a1,$a1,$a1
sw $a1,0($3)
lw $a1,0($3)
and $t3,$a1,$t0
addi $t5,$t5,1
bne $t5,$zero,1
jal 0x00000038
lw $a1,0($3)
addi $s2,$zero,8
add $s6,$s2,$s2
add $s2,$s2,$s6
and $t3,$a1,$s2
beq $t3,$zero,5
beq $t3,$s2,9
addi $s2,$zero,8
beq $t3,$s2,10
sw $t1,0($4)
j 0x00000019
bne $t2,$at,2
nor $t2,$zero,$zero
add $t2,$t2,$t2
sw $t2,0($4)
j 0x00000019
lw $t1,96($17)
sw $t1,0($4)
j 0x00000019
lw $t1,32($17)
sw $t1,0($4)
j 0x00000019
lw $t5,20($0)
add $t2,$t2,$t2
ori $t2,$t2,1
addi $s1,$s1,4
and $s1,$s1,$s4
add $t1,$t1,$v0
bne $t1,$at,1
addi $t1,$t1,5
lw $a1,0($3)
add $t3,$a1,$a1
add $t3,$t3,$t3
sw $t3,0($3)
sw $a2,4($3)
jr $ra

```

四、 讨论与心得

1. Through this assignment, I have a deeper understanding of the address calculation of the mips j-type instruction and the two jump instructions of beq and bne. I didn't know how to implement these two types of instructions at the beginning. With the help of the teaching

assistant, I chose to traverse the file twice. When traversing for the first time all the jump instructions and the addresses of the jump instructions that need to be read. When reading the file for the second time, calculate the required value based on the relative address.

2. For the disassembler, in order to facilitate the conversion of instructions into machine code, first write all instruction names and unique opcodes in the instruction.txt file, read the file to create three queues, and sort. When the mips command is output later, the corresponding command name can be obtained according to the command operation code.