MIPS指令转换器说明文档

张溢弛 3180103772

1.开发环境

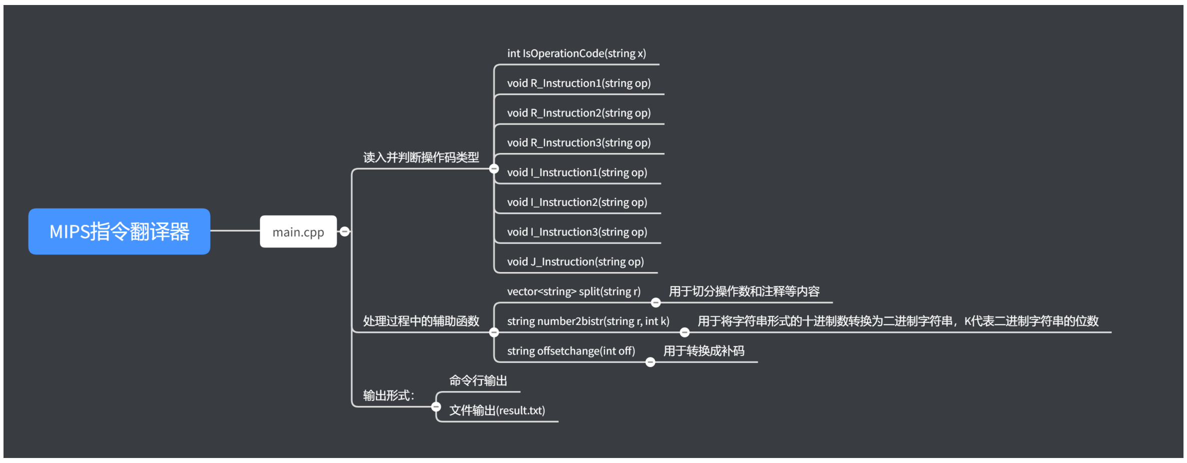
- OS: win 10
- IDE: Visual Studio 2017
- language：C++

2.程序结构

- 主要分为头文件(main.h)和cpp文件(main.cpp)
- main.h中主要给出了若干个函数的定义

```
void InitCode();//初始化机器码查询map
void code_output(string r);//用于在命令行与文件中输出最终结果
int IsOperationCode(string x);//判断输入的是否为操作码

//以下若干函数是划分的几种不同指令类型的转换操作处理函数
void R_Instruction1(string op);
void R_Instruction2(string op);
void R_Instruction3(string op);

void I_Instruction1(string op);
void I_Instruction2(string op);
void I_Instruction3(string op);

void J_Instruction(string op,int count);

//这是几个程序中非常重要的操作函数
vector<string> split(string r);//字符串划分函数，将一行命令按照“,","#","("等关键符号划分为不同的操作数和注释等内容
string number2bistr(string r, int k);//将一个以字符串存储的数转换为若干位二进制码，k表示二进制位数
string offsetchange(int off);//处理反码的转换
```

- main.cpp中主要实现了这些函数的功能，源代码将放在文档的末尾,**程序结构用框图**表示如下


MIPS指令翻译器
main.cpp
读入并判断操作码类型
int IsOperationCode(string x)
void R_Instruction1(string op)
void R_Instruction2(string op)
void R_Instruction3(string op)
void I_Instruction1(string op)
void I_Instruction2(string op)
void I_Instruction3(string op)
void J_Instruction(string op)
处理过程中的辅助函数
vector<string> split(string r)
用于切分操作数和注释等内容
string number2bistr(string r, int k)
用于将字符串形式的十进制数转换为二进制字符串，K代表二进制字符串的位数
string offsetchange(int off)
用于转换成补码
输出形式：
命令行输出
文件输出(result.txt)

3.功能详解

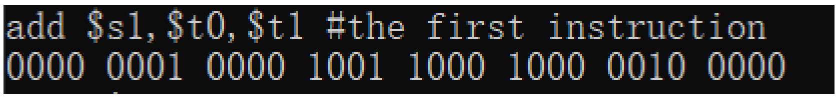
- 本程序实现的功能
  - 实现了将若干条MIPS基本指令翻译为机器码的功能
  - 实现了**大小写指令**均可以被识别和翻译的功能
  - 实现了J类指令的**相对寻址和绝对寻址**的功能
  - 实现了对注释和行首的标记**过滤**的功能
  - 实现了将转换的结果在**命令行**和**文件**(result.txt)中输出的功能
  - 实现了**输入end即可退出程序运行**的功能
- 输入的**格式要求**
  - 标号和操作码之间必须要用**冒号和空格**
  - 操作码，操作数，注释之间必须要有**空格**
  - 例如

  ```
  exjt: add $s1,$t1,$t2
  ```

- 已经实现的指令(约50条)
  - 碍于时间限制和查的MIPS指令集内容不统一，没有实现所有指令，但是指令之间的语法非常相近，以下指令中已经实现了所有格式的指令的翻译
  - eret
  - systemcall
  - add
  - addu
  - sub
  - subu
  - mul
  - mult
  - multu
  - div
  - divu
  - and
  - or
  - nor
  - slt
  - sltu
  - sllv
  - srlv

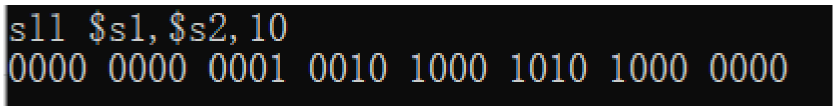
    - srav
    - sll
    - srl
    - sra
    - jr
    - jral
    - addi
    - andui
    - ori
    - xori
    - beq
    - bne
    - slti
    - sltiu
    - lui
    - lw
    - lwx
    - lh
    - lhx
    - lhu
    - lhux
    - sw
    - swx
    - sh
    - shx
    - j
    - jal
- 以下是**程序运行测试的截图**
    - 考虑到篇幅问题，每种格式类型的指令只举一例

R型指令1



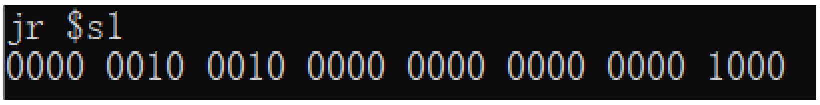
```
add $s1,$t0,$t1 #the first instruction
0000 0001 0000 1001 1000 1000 0010 0000
```

R型指令2



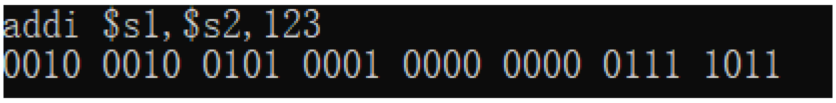
```
sll $s1,$s2,10
0000 0000 0001 0010 1000 1010 1000 0000
```

R形指令3



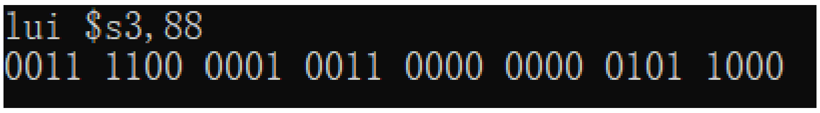
```
jr $s1
0000 0010 0010 0000 0000 0000 0000 1000
```

I型指令1



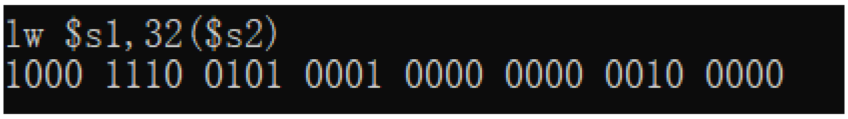
```
addi $s1,$s2,123
0010 0010 0101 0001 0000 0000 0111 1011
```

I型指令2



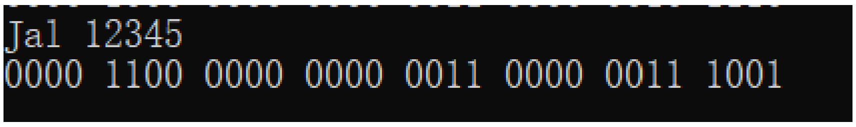
```
lui $s3,88
0011 1100 0001 0011 0000 0000 0101 1000
```

I型指令3



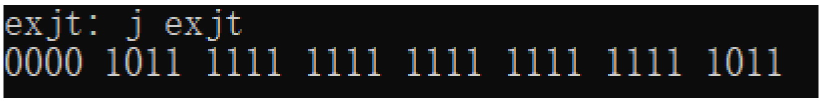
```
lw $s1,32($s2)
1000 1110 0101 0001 0000 0000 0010 0000
```

J型指令的绝对寻址



```
Jal 12345
0000 1100 0000 0000 0011 0000 0011 1001
```

J型指令的相对寻址



```
exjt: j exjt
0000 1011 1111 1111 1111 1111 1111 1011
```

4.源代码(SourceCode)

- main.h

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
using namespace std;

void InitCode();
void code_output(string r);
int IsOperationCode(string x);//判断输入的是否为操作码

void R_Instruction1(string op);
void R_Instruction2(string op);
void R_Instruction3(string op);

void I_Instruction1(string op);
void I_Instruction2(string op);
void I_Instruction3(string op);


void J_Instruction(string op,int count);

vector<string> split(string r);
string number2bistr(string r, int k);
string offsetchange(int off);
```

- main.cpp

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
#include<map>
#include<fstream>
#include<stdlib.h>

#include "main.h"
```

```cpp
using namespace std;

map<string, int> position;
map<string, string> reg_code;
map<string, string> op_code;
string shamt = "00000";

ofstream fp("result.txt");

int main()
{
    cout << "Input end to exit and put one MIPS instruction in ont line" << 
endl;
    cout << "Before you input the instruction you should read the document 
otherwise you will get an error" << endl;
    string op;
    int i,count = 1;
    InitCode();
    while (cin >> op) {
        for (i = 0; i < op.size(); i++) {
            if (op[i] >= 'A'&&op[i] <= 'Z') {
                op[i] = op[i] - 'A' + 'a';
            }
        }
        if (op == "end")
            return 0;
        int x = IsOperationCode(op);
        if (x == 1 ) {
            R_Instruction1(op);
        }
        else if (x == 2 ) {
            R_Instruction2(op);
        }
        else if (x == 3) {
            R_Instruction3(op);
        }
        else if (x == 4) {
            I_Instruction1(op);
        }
        else if (x == 5) {
            I_Instruction2(op);
        }
        else if (x == 6) {
            I_Instruction3(op);
        }
        else if (x == 7) {
            J_Instruction(op,count);
        }
        else if (x == 8) {
            if (op == "eret") {
                cout << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                if (fp.is_open()) {
                    fp << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                }
            }
            else {
                cout << "0000 0000 0000 0000 0000 0000 0000 1100" << endl;
                if (fp.is_open()) {
                    fp << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                }
            }
        }
        else if (x == 0) {
            position[op] = count;
            string op2;
            cin >> op2;
            int y = IsOperationCode(op2);
            if (op == "end")
                return 0;
            if (y == 1) {
                R_Instruction1(op2);
            }
            else if (y == 2) {
                R_Instruction2(op2);
            }
            else if (y == 3) {
                R_Instruction3(op2);
            }
            else if (y == 4) {
                I_Instruction1(op2);
            }
            else if (y == 5) {
                I_Instruction2(op2);
            }
            else if (y == 6) {
                I_Instruction3(op2);
            }
            else if (y == 7) {
                J_Instruction(op2,count);
            }
            else if (y == 8) {
                if (op == "eret") {
                    cout << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                    if (fp.is_open()) {
                        fp << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                    }
                }
                else {
                    cout << "0000 0000 0000 0000 0000 0000 0000 1100" << endl;
                    if (fp.is_open()) {
                        fp << "0100 0001 0000 0000 0000 0000 0001 1000" << endl;
                    }
                }
            }
            else {
                cout << "error!" << endl;
            }
        }

        count++;
    }
    system("pause");
    return 0;
}
```

```
//对输入的若干操作数进行切分
//碰到$时跳过，碰到逗号时换操作数，碰到#表示注释直接结束
vector<string> split(string r)
{
    vector<string> result(3);
    //cout << r;
    int i = 0, j = 0;
    /*
    for (i = 0; i < r.length(); i++) {
        if (r[i] == '(' || r[i] == ')') {
            r[i] = ',';
        }
    }
    */
    for (i = 0; i < r.length(); i++) {
        if (r[i] == ',' || r[i] == '(')
            j++;
        else if (r[i] == ')')
            break;
        else if (r[i] == '$')
            continue;
        else if (r[i] != '#')
            result[j].push_back(r[i]);
        else
            break;
    }
    return result;
}

//将十进制数的字符串转化成16进制，k是需要转化的位数
//k在R型指令中转换成shamt是5，在I型指令中转换成immediate是16
string number2bistr(string r, int k)
{
    int num = atoi(r.c_str());
    cout << num << endl;
    if (num >= 32 && k<=5)
        return "11111";
    string result;
    vector<int> bin;
    while (num != 0) {
        bin.push_back(num % 2);
        num = num / 2;
    }
    for (int i = 0; i < k - bin.size(); i++)
        result += '0';
    for (int i = bin.size() - 1; i >= 0; i--) {
        if (bin[i] == 1)
            result += '1';
        else
            result += '0';
    }
    //cout << result << endl;
    return result;
}

string offsetchange(int off)
{
    int i,is_negative = 0;
    if (off < 0) {
        is_negative = 1;
        off = -off;
    }
    vector<int> bin;
    string result;
    while (off != 0) {
        bin.push_back(off % 2);
        off /= 2;
    }
    for (i = 0; i < 26 - bin.size(); i++) {
        result += '0';
    }
    for (i = bin.size() - 1; i >= 0; i--) {
        if (bin[i] == 1)
            result += '1';
        else
            result += '0';
    }
    // cout << result << endl;
    if (is_negative == 1) {
        for (i = 0; i < result.length(); i++) {
            if (result[i] == '0')
                result[i] = '1';
            else
                result[i] = '0';
        }
    }

    return result;
}

//输出结果的机器码
void code_output(string r)
{
    for (int i = 0; i < r.size(); i++) {
        cout << r[i];
        if (i % 4 == 3) {
            cout << " ";
        }
    }
    char *copy = new char[r.length() + 1];
    for (int i = 0; i < r.length(); i++) {
        copy[i] = r[i];
    }
    copy[r.length()] = '\n';

//fstream fp("result.txt");
    if (fp.is_open()) {
        fp << r << endl;
    }
    cout << endl;
}

//R-Instruction translation into machine code
void R_Instruction1(string op)
{
    string r;
    cin >> r;
    vector<string> result = split(r);
    string machine_code = "000000";
    machine_code += reg_code[result[1]] + reg_code[result[2]] +
reg_code[result[0]] + shamt + op_code[op];
    code_output(machine_code);
}

void R_Instruction2(string op)
{
    string r;
    cin >> r;
    vector<string> result = split(r);
    string machine_code = "000000" + reg_code[result[1]] + "00000" +
reg_code[result[0]];

    string shamt2 = number2bistr(result[2], 5);

    machine_code += shamt2 + op_code[op];
    code_output(machine_code);
}

void R_Instruction3(string op)
{
    string r;
    cin >> r;
    vector<string> result = split(r);
    string machine_code = "000000" + reg_code[result[0]] +
"000000000000000001000";
    code_output(machine_code);
}

void I_Instruction1(string op)
{
    string r;
    cin >> r;
    vector<string> result = split(r);
    string machine_code = op_code[op] + reg_code[result[1]] +
reg_code[result[0]];
    string immediate = number2bistr(result[2], 16);
    machine_code += immediate;
    code_output(machine_code);

}

void I_Instruction2(string op)
{
    string r;
    cin >> r;
    vector<string> result = split(r);
    string machine_code = "00111100000" + reg_code[result[0]];
    string immediate = number2bistr(result[1], 16);
    machine_code += immediate;
    //cout << immediate << endl;
    //cout << result[0] << endl;
    //cout << result[1] << endl;
    code_output(machine_code);
}
```

```
void I_Instruction3(string op)
{
    //lw $s1,10($s2)
    string r;
    cin >> r;
    vector<string> result = split(r);
    //cout << result[0] << endl;
    //cout << result[1] << endl;
    //cout << result[2] << endl;
    string machine_code = op_code[op] + reg_code[result[2]] +
reg_code[result[0]];
    string immediate = number2bistr(result[1], 16);
    machine_code += immediate;
    //cout << immediate << endl;
    code_output(machine_code);
}


void J_Instruction(string op, int count)
{
    string r, s;
    cin >> r;
    s = r + ':';
    if (position[s] >= 1) {
        int offset = -4 * (count - position[s] + 1);
        //cout << offset << endl;
        string immediate = offsetchange(offset);
        string machine_code = op_code[op] + immediate;
        code_output(machine_code);
    }
    else {
        string immediate = number2bistr(r, 26);
        string machine_code = op_code[op] + immediate;
        code_output(machine_code);
    }
}

//我们使用stl中的map来存储MIPS指令和32个寄存器代表的机器码
//目前只实现了有严格标准定义的31条指令
void InitCode()
{
    op_code["eret"] = "011000";
    op_code["systemcall"] = "001100";

    //R-instruction set 1
    op_code["add"] = "100000";
    op_code["addu"] = "100001";
    op_code["sub"] = "100010";
    op_code["subu"] = "100011";
    op_code["mul"] = "000010";
    op_code["mult"] = "011000";
    op_code["multu"] = "011001";
    op_code["div"] = "011010";
    op_code["divu"] = "011011";

    op_code["and"] = "100100";
    op_code["or"] = "100101";
    op_code["nor"] = "100111";
    op_code["slt"] = "101010";
    op_code["sltu"] = "101011";
    op_code["sllv"] = "000100";
    op_code["srlv"] = "000110";
    op_code["srav"] = "000111";

    //R-instruction set 2
    op_code["sll"] = "000000";
    op_code["srl"] = "000010";
    op_code["sra"] = "000011";
    //R-instruction set 3
    op_code["jr"] = "0010000";
    op_code["jalr"] = "001001";

    //I-instruction set 1
    op_code["addi"] = "001000";
    op_code["andui"] = "001001";
    op_code["andi"] = "001100";
    op_code["ori"] = "101011";
    op_code["xori"] = "001110";
    op_code["beq"] = "000100";
    op_code["bne"] = "000101";

    op_code["slti"] = "001010";
    op_code["sltiu"] = "001011";

    //I-instruction set 2
    op_code["lui"] = "001111";

    //I-instruction set 3
    op_code["lw"] = "100011";
    op_code["lwx"] = "100011";
    op_code["lh"] = "100001";
    op_code["lhx"] = "100001";
    op_code["lhu"] = "100101";
    op_code["lhux"] = "100101";
    op_code["sw"] = "101011";
    op_code["swx"] = "101011";
    op_code["sh"] = "101001";
    op_code["shx"] = "101001";

    //J-instruction set
    op_code["j"] = "000010";
    op_code["jal"] = "000011";

    //regester code
    reg_code["zero"] = "00000";
    reg_code["at"] = "00001";
    reg_code["v0"] = "00010";
    reg_code["v1"] = "00011";

    reg_code["a0"] = "00100";
    reg_code["a1"] = "00101";
    reg_code["a2"] = "00110";
    reg_code["a3"] = "00111";
    reg_code["t0"] = "01000";
    reg_code["t1"] = "01001";
    reg_code["t2"] = "01010";
    reg_code["t3"] = "01011";
    reg_code["t4"] = "01100";
    reg_code["t5"] = "01101";
    reg_code["t6"] = "01110";
    reg_code["t7"] = "01111";

    reg_code["s0"] = "10000";
    reg_code["s1"] = "10001";
    reg_code["s2"] = "10010";
    reg_code["s3"] = "10011";
    reg_code["s4"] = "10100";
    reg_code["s5"] = "10101";
    reg_code["s6"] = "10110";
    reg_code["s7"] = "10111";

    reg_code["t8"] = "11000";
    reg_code["t9"] = "11001";

    reg_code["k0"] = "11010";
    reg_code["k1"] = "11011";

    reg_code["gp"] = "11100";
    reg_code["sp"] = "11101";
    reg_code["fp"] = "11110";
    reg_code["ra"] = "11111";
}


//当参数是操作码是会返回一个正数
//否则返回0
int IsOperationCode(string x)
{
    if (x == "add" || x == "addu" || x == "sub" || x == "subu" || x == "mul" ||
x == "mult" || x == "multu" || x == "div" || x == "divu" || x == "and" || x ==
"or" || x == "nor" || x == "slt" || x == "sltu" || x == "sllv" || x == "srlv" ||
x == "srav") {
        return 1;
    }
    else if (x == "sll"||x=="srl"||x=="sra") {
        return 2;
    }
    else if (x == "jr"||x=="jalr") {
        return 3;
    }
    else if (x == "addi" || x == "andui" || x == "andi" || x == "ori" || x ==
"xor" || x == "beq" || x == "bne" || x == "slti" || x == "sltiu") {
        return 4;
    }
    else if (x == "lui") {
        return 5;
    }
    else if (x == "lw" || x == "lwx" || x == "lh" || x == "lhx" || x == "lhu" ||
x == "lhux" || x == "sw" || x == "swx" || x == "sh" || x ==
"shx"||x=="slti"||x=="sltiu") {
        return 6;
    }
    else if (x == "j" || x == "jal") {
        return 7;
    }
    else if (x == "eret" || x == "systemcall") {
        return 8;
    }
    else
        return 0;
}
```