MIPS模拟机

Name: 王子腾

StuID: 3180102173

Date: 2020/04/28

1.实验描述

以程序模拟MIPS运行，功能包括：

1. 汇编器：输入汇编指令，转换成机器码模拟执行。
2. 读入汇编好的机器码(二进制)，显示对应的汇编指令(反汇编)，并模拟执行。

---

1. 模拟器运行界面设计：可以命令行或窗口界面。列表显示32个寄存器。
2. 可执行多条指令。可观察寄存器、内存的变化。（命令行版可参考DEBUG）

DEBUG命令：

> -->R-看寄存器，
>
> -->D-数据方式看内存，
>
> -->U-指令方式看内存，
>
> -->A-写汇编指令到内存，
>
> -->T-单步执行内存中的指令

2. 程序架构

2.1 MIPS类（模拟机）

```
class MIPS
{
private:
    int PC;     // Program Counter
    int num;    // num of instructions
    Memory m;
    Register reg;
    Instruction ins;
public:
    MIPS(int type);
    ~MIPS();
    void Regshow();
    void Addins();
    void execute();
    void Memshowdata();
    void Memshowins();
};
```

2.2 Memory类（内存）

```
class Memory
{
private:
    const int capacity = 1024;      // 1kb
    int ptr = 0;                    // ins-ptr
    string memory[1024];            // 32bit
    int record[1024];               // 1: Instruction 2: Data
public:
    void writedata(int pos, int type,string val);
    string read(int pos);
    int showadd(int pos);
    void showdata();
};
```

2.3 Register类（32个寄存器）

```
class Register
{
private:
    const string reg[32] = {"zero", "at", "v0", "v1", "a0", "a1", "a2",
"a3", "t0", "t1", "t2", "t3", "t4","t5", "t6", "t7", "s0", "s1", "s2",
"s3", "s4", "s5", "s6", "s7", "t8", "t9", "k0", "k1", "gp", "sp",
"fp","ra"};
    int val[32];
public:
    Register();
    ~Register();
    int regindex(string);
    string regname(int);
    void setval(int index, int v);
    int getval(int);
    void show();
};
```

2.4 Instruction类（指令系统）

```
class Instruction
{
private:
    vector<string> sins;
    // vector<int> iins;
    vector<string> bins;
    vector<string> label;

public:
    vector<string> option;
    Instruction(int type);      // 1: Assemble 2: Binary
    ~Instruction();
    string getins(int type, int pos);
    int getsize();
    void showins(int);
    void addins(string);
    void compile(int,string);   // Compile Unit
    void reverse(int);          // Reverse-Compile
```

```
    int findlabel(string);      // Label-Check
};
```

可执行指令

```
add     rd, rs, rt
addi    rt, rs, dat
sub     rd, rs, rt
beq     rs, rt, label
bne     rs, rt, label
j       label
lw      rt, dat(rs)
sw      rt, dat(rs)
```

3. 核心代码

3.1 汇编指令编译

3.1.1 字符串预处理（标签 + 操作符 分割）

```
// Label Apart
regex e("^([a-z 0-9]+):");              // Label Process
smatch m;
bool found = regex_search(instruction,m,e);
if(found)
{
    lab = m.str();
    lab.erase(lab.end()-1);
    int i = 0;
    while(instruction[i]!=':')
        i++;
    instruction.erase(0,i+1);
}
else lab = "";

label.push_back(lab);                   // label

// clean space
while(instruction[0]==' ')
    instruction.erase(instruction.begin());

string op;
int i;
// op
for(i = 0; instruction[i]!=' '; i++)
{
    op+=instruction[i];
}
option.push_back(op);
// clean space
for(; i < instruction.length(); i++)
{
    if(instruction[i]==' ')
    {
```

```
        instruction.erase(instruction.begin()+i);
        i--;
    }
}
sins.push_back(instruction);
```

3.1.2 编译转二进制

例：add运算符

```
string R1, R2, R3, str, code;
int r1, r2, r3, ofs;
Register r;
if(op=="add"){
    code="000000";
    int i = 0, j = 0;
    // rd
    i = sins[index].find('$');
    j = sins[index].find(',', i+1);
    R1 = sins[index].substr(i+1, j-i-1);
    r1 = r.regindex(R1);
    // rs
    i = j+1;//$
    j = sins[index].find(',', i+1);
    R2 = sins[index].substr(i+1, j-i-1);
    r2 = r.regindex(R2);
    // rt
    i = j+1;
    R3 = sins[index].substr(i+1);
    r3 = r.regindex(R3);

    code += numtobin(r2, 5);
    code += numtobin(r3, 5);
    code += numtobin(r1, 5);
    code += "00000100000";
}
```

例：J 运算符

```
else if(op=="j"){
    code="000010";
    str = sins[index].substr(1);
    ofs = findlabel(str);
    code += numtobin(ofs, 26);
}
```

例：lw 运算符

```
else if(op=="lw"){
    code="100011";

    int i, j;
    // rs
    i = sins[index].find('$');
    j = sins[index].find(',', i+1);
    R1 = sins[index].substr(i+1, j-i-1);
```

```
    r1 = r.regindex(R1);
    // rt
    i = sins[index].find('$', j);
    j = sins[index].find(')', i);
    R2 = sins[index].substr(i+1, j-i-1);
    r2 = r.regindex(R2);

    code += numtobin(r1, 5);
    code += numtobin(r2, 5);

    i = sins[index].find(',');
    j = sins[index].find('(');
    str = sins[index].substr(i+1, j-i-1);
    istringstream s(str);
    int a;
    s>>a;
    code+=numtobin(a, 16);
}
```

3.2 反汇编

例：add & sub 运算符

```
string op, ins;
string R1, R2, R3, str;
int temp;
Register reg;
op = bins[index].substr(0, 6);
if(op == "000000")//add & sub
{
    if(bins[index][30]=='1')//sub
    i   ns+="sub ";
    else ins+="add ";

    //rs
    R1 = bins[index].substr(6,5);
    R1 = "$"+reg.regname(bintonum(R1));
    //rt
    R2 = bins[index].substr(11,5);
    R2 = "$"+reg.regname(bintonum(R2));
    //rd
    R3 = bins[index].substr(16,5);
    R3 = "$"+reg.regname(bintonum(R3));

    ins = ins + R3 + "," + R1 + "," + R2;
}
```

例：beq 运算符

```
else if(op == "000100")//beq
{
    ins+="beq ";
    //rs
    R1 = bins[index].substr(6,5);
    R1 = "$"+reg.regname(bintonum(R1));
```

```
    //rt
    R2 = bins[index].substr(11,5);
    R2 = "$"+reg.regname(bintonum(R2));

    str = bins[index].substr(16);
    temp = bintonum(str);

    stringstream ss;
    ss << temp;
    ss >> str;

    ins += R1+","+R2+","+str;
}
```

例：sw 运算符

```
else if(op == "101011")//sw
{
    ins+="sw ";

    //rs
    R1 = bins[index].substr(6,5);
    R1 = "$"+reg.regname(bintonum(R1));
    //rt
    R2 = bins[index].substr(11,5);
    R2 = "$"+reg.regname(bintonum(R2));

    str = bins[index].substr(16);
    temp = bintonum(str);

    stringstream ss;
    ss << temp;
    ss >> str;

    ins += R2+","+str+"("+R1+")";
}
```

3.3 指令执行

例：add & sub 指令

```
size_t i = ins.getsize();
if (PC >= i)
{
    cout << "You Have Run All Codes" << endl;
    return;
}
string str = ins.getins(0,PC), R1, R2, R3, addr;
cout << "PC: "<< dec <<PC<<endl;
PC++;
string op = str.substr(0,6);
int r1, r2, r3, ofs;
if(op=="000000")//add & sub
{
    int flag = 1;
```

```
    if(str[30]=='1')//sub
    flag=-1;

    R1 = str.substr(6,5);
    r1 = reg.getval(bintonum(R1));
    R2 = str.substr(11,5);
    r2 = reg.getval(bintonum(R2));
    R3 = str.substr(16,5);

    reg.setval(bintonum(R3), r1+(flag*r2));
}
```

例：bne 指令

```
else if(op=="000101")//bne
{
    R1 = str.substr(6,5);
    r1 = reg.getval(bintonum(R1));
    R2 = str.substr(11,5);
    r2 = reg.getval(bintonum(R2));

    addr = str.substr(16);
    ofs = bintonum(addr);
    if(r1!=r2)
    PC+=ofs-1;
}
```

例： J 指令

```
else if(op=="000010")//j
{
    addr = str.substr(6);
    ofs = bintonum(addr);
    PC = ofs;
}
```

3.4 内存操作

3.4.1 读取

```
string Memory::read(int pos)
{
    return memory[pos];
}
```

3.4.2 写入

```
void Memory::writedata(int pos, int type, string val)//ins:1 data:2
{
    if(type == 1)
    {
        memory[ptr] = val;
        record[ptr] = type;
        ptr++;
```

```
    }
    else
    {
        memory[pos] = val;
        record[pos] = type;
    }
}
```

3.5 辅助函数

3.5.1 二进制字符串转整数

```
int bintonum(string bin)
{
    int number = 0, bit = bin.size();
    for (int i = 0; i < bit; i++)
    {
        number = 2 * number + bin[i] - '0';
    }
    return number;
}
```

3.5.2 整数转二进制字符串（指定位数）

```
string numtobin(int num, int bit)
{
    char str[100];
    itoa(num, str, 2);
    string s = str;
    if (s.size() > bit)
        s = s.substr(s.size() - bit);
    for (int i = s.size(); i < bit; i++)
        s = '0' + s;
    return s;
}
```

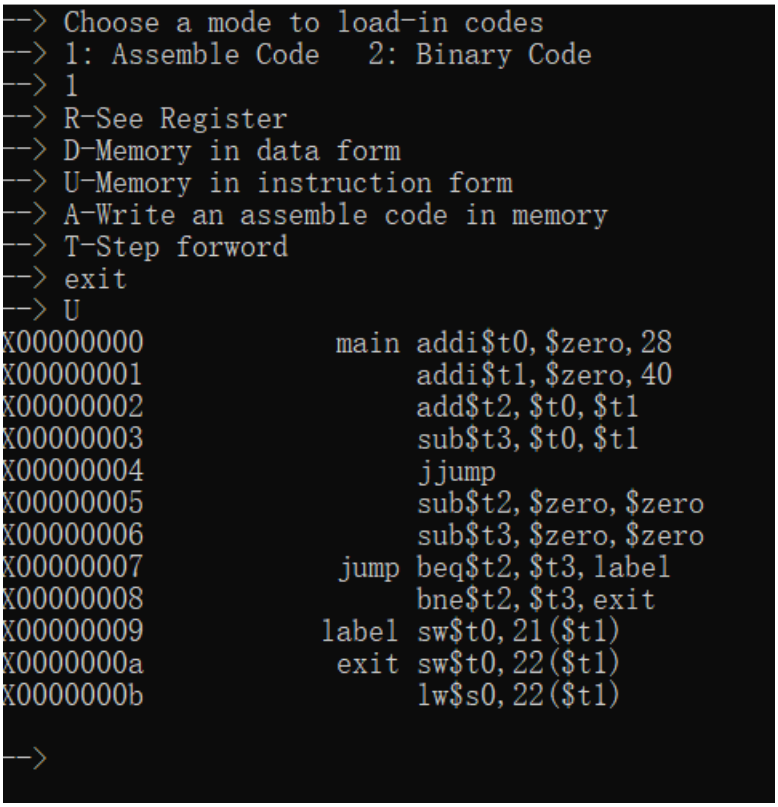
4 实验验收

4.1 输入 MIPS 汇编码

```
12
main:    addi $t0,$zero,28
         addi $t1,$zero,40
         add $t2,$t0,$t1
         sub $t3,$t0,$t1
         j jump
         sub $t2,$zero,$zero
         sub $t3,$zero,$zero
jump:    beq $t2,$t3,label
         bne $t2,$t3,exit
label:   sw $t0,21($t1)
exit:    sw $t0,22($t1)
         lw $s0,22($t1)
```

测试反馈

1. U-指令方式看内存



2. D-数据方式看内存

```
--> D
X00000000       00100000000010000000000000011100
X00000001       00100000000010010000000000101000
X00000002       00000001000010010101000000100000
X00000003       00000001000010010101100000100010
X00000004       00001000000000000000000000000111
X00000005       00000000000000000101000000100010
X00000006       00000000000000000101100000100010
X00000007       00010001010010110000000000000010
X00000008       00010101010010110000000000000010
X00000009       10101101000010010000000000010101
X0000000a       10101101000010010000000000010110
X0000000b       10001110000010010000000000010110

-->
```

3. T-单步执行内存中的指令

```
--> T
PC: 0

--> T
PC: 1

--> T
PC: 2

--> T
PC: 3

--> T
PC: 4

--> T
PC: 7

--> T
PC: 8

-->
```

4. R-看寄存器

```
--> R
| Register |  Value | Register |  Value | Register |  Value | Register |  Value |
|    $zero |      0 |      $at |      0 |      $v0 |      0 |      $v1 |      0 |
|      $a0 |      0 |      $a1 |      0 |      $a2 |      0 |      $a3 |      0 |
|      $t0 |     28 |      $t1 |     40 |      $t2 |     68 |      $t3 |    -12 |
|      $t4 |      0 |      $t5 |      0 |      $t6 |      0 |      $t7 |      0 |
|      $s0 |      0 |      $s1 |      0 |      $s2 |      0 |      $s3 |      0 |
|      $s4 |      0 |      $s5 |      0 |      $s6 |      0 |      $s7 |      0 |
|      $t8 |      0 |      $t9 |      0 |      $k0 |      0 |      $k1 |      0 |
|      $gp |      0 |      $sp |      0 |      $fp |      0 |      $ra |      0 |
-->
```

5. A-写汇编指令到内存

```
--> A
--> add $t4,$t2,$zero
--> U
X00000000              main addi$t0,$zero,28
X00000001                   addi$t1,$zero,40
X00000002                   add$t2,$t0,$t1
X00000003                   sub$t3,$t0,$t1
X00000004                   jjump
X00000005                   sub$t2,$zero,$zero
X00000006                   sub$t3,$zero,$zero
X00000007               jump beq$t2,$t3,label
X00000008                   bne$t2,$t3,exit
X00000009             label sw$t0,21($t1)
X0000000a              exit sw$t0,22($t1)
X0000000b                   lw$s0,22($t1)
X0000000c                   add$t4,$t2,$zero
```

执行后$t4增加68

```
--> T
PC: 12

--> R
| Register|   Value| Register|   Value| Register|   Value| Register|   Value|
|    $zero|       0|      $at|       0|      $v0|       0|      $v1|       0|
|      $a0|       0|      $a1|       0|      $a2|       0|      $a3|       0|
|      $t0|      28|      $t1|      40|      $t2|      68|      $t3|     -12|
|      $t4|      68|      $t5|       0|      $t6|       0|      $t7|       0|
|      $s0|      28|      $s1|       0|      $s2|       0|      $s3|       0|
|      $s4|       0|      $s5|       0|      $s6|       0|      $s7|       0|
|      $t8|       0|      $t9|       0|      $k0|       0|      $k1|       0|
|      $gp|       0|      $sp|       0|      $fp|       0|      $ra|       0|
-->
```

4.2 输入二进制机器码

```
12
00100000000010000000000000011100
00100000000010010000000000101000
00000001000010010101000000100000
00000001000010010101100000100010
00001000000000000000000000000111
00000000000000000101000000100010
00000000000000000101100000100010
00010001010010110000000000000010
00010101010010110000000000000010
10101101000010010000000000010101
10101101000010010000000000010110
10001110000010010000000000010110
```

测试反馈

1. U-指令方式看内存

```
--> Choose a mode to load-in codes
--> 1: Assemble Code   2: Binary Code
--> 2
--> R-See Register
--> D-Memory in data form
--> U-Memory in instruction form
--> A-Write an assemble code in memory
--> T-Step forword
--> exit
--> U
X00000000                addi $t0,$zero,28
X00000001                addi $t1,$zero,40
X00000002                add $t2,$t0,$t1
X00000003                sub $t3,$t0,$t1
X00000004                j 7
X00000005                sub $t2,$zero,$zero
X00000006                sub $t3,$zero,$zero
X00000007                beq $t2,$t3,2
X00000008                bne $t2,$t3,2
X00000009                sw $t1,21($t0)
X0000000a                sw $t1,22($t0)
X0000000b                lw $t1,22($s0)

-->
```

2. D-数据方式看内存

```
--> D
X00000000      00100000000010000000000000011100
X00000001      00100000000010010000000000101000
X00000002      00000001000010010101000000100000
X00000003      00000001000010010101100000100010
X00000004      00001000000000000000000000000111
X00000005      00000000000000000101000000100010
X00000006      00000000000000000101100000100010
X00000007      00010001010010110000000000000010
X00000008      00010101010010110000000000000010
X00000009      10101101000010010000000000010101
X0000000a      10101101000010010000000000010110
X0000000b      10001110000010010000000000010110

-->
```

3. T-单步执行内存中的指令

```
--> T
PC: 4

--> T
PC: 7

--> T
PC: 8

--> T
PC: 10

--> T
PC: 11

--> T
You Have Run All Codes
```

4. R-看寄存器

```
--> R
| Register|   Value| Register|   Value| Register|   Value| Register|   Value|
|    $zero|       0|      $at|       0|      $v0|       0|      $v1|       0|
|      $a0|       0|      $a1|       0|      $a2|       0|      $a3|       0|
|      $t0|      28|      $t1|      40|      $t2|      68|      $t3|     -12|
|      $t4|       0|      $t5|       0|      $t6|       0|      $t7|       0|
|      $s0|      28|      $s1|       0|      $s2|       0|      $s3|       0|
|      $s4|       0|      $s5|       0|      $s6|       0|      $s7|       0|
|      $t8|       0|      $t9|       0|      $k0|       0|      $k1|       0|
|      $gp|       0|      $sp|       0|      $fp|       0|      $ra|       0|
-->
```