《计算机结构与组成》
课程设计报告

班级序号：33

学　　号：20171002608

姓　　名：徐鸿飞

指导教师：杨林权/罗忠文/刘袁缘

成　　绩：

中国地质大学信息工程学院

2019 年 1 月

实习题目一

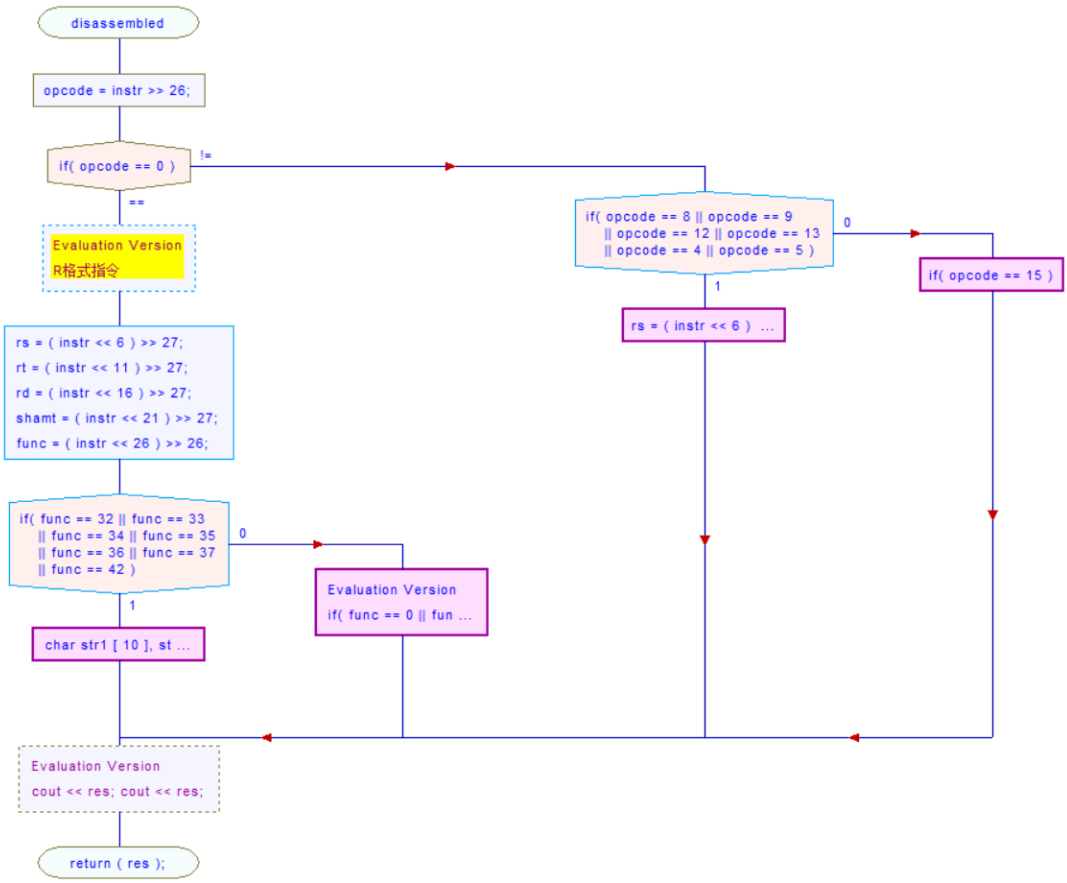
<概述题目要求>

本题你需要构建一个 MIPS 汇编语言指令集的子集的指令解释器。它将实现取指、反汇编、解码,并执行 MIPS 机器指令。你将构建一个简化版的 MARS。当然，本题和 MARS 有一个非常重要的区别。MARS 输入为汇编语言，而本题的输入是.bin 文件（即 MIPS 机器语言程序）， 输出是汇编代码，并显示执行结果（内存和寄存器值的变化）。

<设计思路>

1、 阅读已有的代码框架，知现有的框架已经实现了读取内存、内存和寄存器的初始化、输出等大部分功能，需要我们来实现的只有两个函数：disassembled 和 simulateInstruction。

2、 Disassembled 函数：将程序读取的机器码语言转换为汇编语言。实现思路：分割传入的参数 instr，先读取前六位为 opcode（26：31），若 opcode 为 0，则将传入的 instr 分为 rs（21：25），rt（16：20），rd（11：15），shamt（6：10），func（0：5）几个部分，再根据 func 的值来确定指令内容，即可翻译为 MIPS 汇编指令；若 opcode 为其他值，则根据格式（由 opcode 确定）来分割 instr，具体为：opcode 等于 8、9、12、13、4、5、35、43、15 时分割为 rs（21：25），rt（16：20），immediate（0：15）；opcode 等于 2、3 时分割为 opcode（26：31）、address（0：25）再翻译为 MIPS 汇编指令，若 opcode 为其他值，则提示并退出。



3、 simulateInstruction 函数：执行传入的指令，与 Disassembled 函数类似，先读取前六位 opcode，再根据格式来分割指令，然后根据指令的内

容来更改寄存器与内存。

<实现和测试>

核心功能函数代码：

立即数补码转真值：

```
int ImmediatelyConvertToDecimal(int immediate)
{
    int x, y;
    y = (immediate << 16) >> 16;
    x = y >> 15;
    if (x == 1)
        y = -((y - 1) ^ 0xffff);
    return y;
};
```

十进制转 16 进制：

```
string tentohex(int im){
    char s[50];
    sprintf_s(s, "%x", im);
    return string(s);
};
```

Disassembled 函数：

```
char* disassembled (unsigned int instr, unsigned int pc) {
    /* You replace this code by the right stuff. */
    //if (/*instruction isn't supported */) exit (0); // Your program must exit when
an unsupported instruction is detected
    int opcode, rs, rt, rd, shamt, func, immediate;
    opcode = instr >> 26;
    if (opcode == 0) {//R格式指令
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        rd = (instr << 16) >> 27;
        shamt = (instr << 21) >> 27;
        func = (instr << 26) >> 26;
        if (func == 32 || func == 33 || func == 34 || func == 35 || func == 36 ||
func == 37 || func == 42 ) {
            char str1[10], str2[10] = "\t$", str3[10] = ", $", str4[10] = ", $";
            switch (func)
            {
            case 32:
                strcpy_s(str1, "add"); strcpy_s(res, str1); break;
            case 33:
                strcpy_s(str1, "addu"); strcpy_s(res, str1); break;
            case 34:
                strcpy_s(str1, "sub"); strcpy_s(res, str1); break;
            case 35:
                strcpy_s(str1, "subu"); strcpy_s(res, str1); break;
```

```
            case 36:
                strcpy_s(str1, "and"); strcpy_s(res, str1); break;
            case 37:
                strcpy_s(str1, "or"); strcpy_s(res, str1); break;
            case 42:
                strcpy_s(str1, "slt"); strcpy_s(res, str1);break;
            default:break;
            }
            strcat_s(str2, to_string(rd).c_str()); strcat_s(res, str2);
            strcat_s(str3, to_string(rs).c_str()); strcat_s(res, str3);
            strcat_s(str4, to_string(rt).c_str()); strcat_s(res, str4);
        }
        else if (func == 0 || func == 2) {

            char str1[10], str2[10] = "\t$", str3[10] = ", $", str4[10]=", ";
            switch (func)
            {
            case 0:
                strcpy_s(str1, "sll"); strcpy_s(res, str1); break;
            case 2:
                strcpy_s(str1, "srl"); strcpy_s(res, str1); break;
            default:break;
            }
            strcat_s(str2, to_string(rd).c_str()); strcat_s(res, str2);
            strcat_s(str3, to_string(rt).c_str()); strcat_s(res, str3);
            strcat_s(str4, to_string(tento(shamt)).c_str()); strcat_s(res,
str4);
        }
        else if (func==8)
        {
            char str1[10] = "jr\t$"; strcat_s(str1, to_string(rs).c_str());
            strcpy_s(res, str1);
        }
        else
        {
            exit(0);
        }
    }
    else if (opcode==8 || opcode == 9 || opcode == 12 || opcode == 13 || opcode ==
4 || opcode == 5)
    {
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
```

```
        char str1[10], str2[10] = "\t$", str3[10] = ", $", str4[30] = ", ";
        switch (opcode)
        {
        case 8:
            strcat_s(str2, to_string(rt).c_str()); strcat_s(str3,
to_string(rs).c_str());
           strcpy_s(str1, "addi"); strcat_s(str4, to_string(immediate).c_str());
break;
        case 9:
            strcat_s(str2, to_string(rt).c_str()); strcat_s(str3,
to_string(rs).c_str());
            strcpy_s(str1, "addiu"); strcat_s(str4,
to_string(immediate).c_str()); break;
        case 12:
            strcat_s(str2, to_string(rt).c_str()); strcat_s(str3,
to_string(rs).c_str());
            strcpy_s(str1, "andi"); strcat_s(str4, "0x"); strcat_s(str4,
tentohex(immediate).c_str()); break;
        case 13:
            strcat_s(str2, to_string(rt).c_str()); strcat_s(str3,
to_string(rs).c_str());
            strcpy_s(str1, "ori");  strcat_s(str4, "0x"); strcat_s(str4,
tentohex(immediate).c_str()); break;
        case 4:
            strcat_s(str2, to_string(rs).c_str()); strcat_s(str3,
to_string(rt).c_str());
            strcpy_s(str1, "beq"); strcat_s(str4, "0x"); strcat_s(str4,
tentohex(pc + immediate*4+4 ).c_str()); break;
        case 5:
            strcat_s(str2, to_string(rs).c_str()); strcat_s(str3,
to_string(rt).c_str());
            strcpy_s(str1, "bne"); strcat_s(str4, "0x"); strcat_s(str4,
tentohex(pc + immediate*4+4 ).c_str()); break;
        default:break;
        }strcpy_s(res, str1);
        strcat_s(res, str2);
        strcat_s(res, str3);
        strcat_s(res, str4);
    }
    else if (opcode==15)
    {
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
        char str1[] = "lui", str2[10] = "\t$",str3[7]=", 0x"; strcpy_s(res, str1);
```

```
        strcat_s(str2, to_string(rt).c_str());
        strcat_s(res, str2);
        strcat_s(str3, tentohex(immediate).c_str());
        strcat_s(res, str3);
    }
    else if (opcode == 35 || opcode == 43) {
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
        char str1[10], str2[10] = "\t$", str3[10] = " ,", str4[10] = "($";
        if (opcode == 35) { strcpy_s(str1, "lw"); }
        else { strcpy_s(str1, "sw"); }
        strcpy_s(res, str1); strcat_s(str2, to_string(rt).c_str()); strcat_s(res,
str2);
        strcat_s(str3, to_string(immediate).c_str()); strcat_s(res, str3);
        strcat_s(str4, to_string(rs).c_str()); strcat_s(str4, ")"); strcat_s(res,
str4);\
    }
    else if (opcode == 2 || opcode == 3) {
        int address = (instr << 6) >> 6;
        char str1[10], str2[30] = "\t0x";
        if (opcode == 2) { strcpy_s(str1, "j"); }
        else { strcpy_s(str1, "jal"); }
        strcat_s(str2, tentohex(address).c_str());
        strcpy_s(res, str1); strcat_s(res, str2);
    }
    else exit(0);
    return (res);
}
```

simulateInstruction 函数：

```
void simulateInstruction (Computer* mips, unsigned int instr, int *changedReg, int
*changedMem) {
    /* You replace this code by the right stuff. */
    mips->pc = mips->pc + 4;
    *changedReg = -1;
    *changedMem = -1;
    int opcode, rs, rt, rd, shamt, func, immediate; char res[100];
    opcode = instr >> 26;
    if (opcode == 0) {//R格式指令
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        rd = (instr << 16) >> 27;
        shamt = (instr << 21) >> 27;
        func = (instr << 26) >> 26;
```

```
        if (func == 32 || func == 33 || func == 34 || func == 35 || func == 36 ||
func == 37 || func == 42) {
            //char str1[5], str2[5] = "\t$", str3[5] = ", $", str4[5] = ", $";
            switch (func)
            {
            case 32://add
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] + mips->registers[rt];
break;
            case 33://addu
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] + mips->registers[rt];
break;
            case 34://sub
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] - mips->registers[rt];
break;
            case 35://subu
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] - mips->registers[rt];
break;
            case 36://and
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] & mips->registers[rt];
break;
            case 37://or
                *changedReg = rd;
                mips->registers[rd] = mips->registers[rs] | mips->registers[rt];
break;
            case 42://slt
                *changedReg = rd;
               mips->registers[rd] = (mips->registers[rs] < mips->registers[rt]);
break;
            default:break;
            }
        }
        else if (func == 0 || func == 2) {
            //char str1[5], str2[5] = "\t$", str3[5] = ", $", str4[5] = ", ";
            switch (func)
            {
            case 0://sll
                *changedReg = rd;
                mips->registers[rd] = (mips->registers[rt] << tento(shamt));
break;
            case 2://srl
```

```
                *changedReg = rd;
                mips->registers[rd] = (mips->registers[rt] >> tento(shamt));
break;
            default:break;
            }
        }
        else if (func == 8)
        {
            mips->pc = mips->registers[rs];
        }
        else
        {
            exit(0);
        }
    }
    else if (opcode == 8 || opcode == 9 || opcode == 12 || opcode == 13 || opcode
== 4 || opcode == 5)
    {
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
        char str1[10], str2[10] = "\t$", str3[10] = ", $", str4[10] = ", ";
        strcat_s(str2, to_string(rs).c_str()); strcat_s(str3,
to_string(rt).c_str());
        switch (opcode)
        {
        case 8://addi
            *changedReg = rt;
            mips->registers[rt] = mips->registers[rs] + tento(immediate); break;
        case 9://addiu
            *changedReg = rt;
            mips->registers[rt] = mips->registers[rs] + tento(immediate); break;
        case 12://andi
            *changedReg = rt;
            mips->registers[rt] = mips->registers[rs] & tento(immediate); break;
        case 13://ori
            *changedReg = rt;
            mips->registers[rt] = mips->registers[rs] | tento(immediate); break;
        case 4://beq
            if (mips->registers[rs] == mips->registers[rt]) { mips->pc = mips->pc
+ immediate * 4; }break;
        case 5://bne
            if (mips->registers[rs] != mips->registers[rt]) { mips->pc = mips->pc
+ immediate * 4; }break;
```

```
        default:break;
        }
    }
    else if (opcode == 15)//lui
    {
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
        *changedReg = rt;
        mips->registers[rt] = immediate * 65536;
    }
    else if (opcode == 35 || opcode == 43) {
        rs = (instr << 6) >> 27;
        rt = (instr << 11) >> 27;
        immediate = (instr << 16) >> 16;
        immediate = tento(immediate);
        if (opcode == 35) {
            *changedReg = rt;
            mips->registers[rt] = contents(mips, mips->registers[rs] +
immediate);
        }
        else {
            *changedMem = mips->registers[rs] + immediate;
            int index = (*changedMem - 0x00400000) / 4;
            if (((*changedMem & 0x3) != 0) || (index < 0) || (index >= (MAXNUMINSTRS
+ MAXNUMDATA))) {
                printf("Invalid Address: %8.8x", *changedMem);
                exit(0);
            }
            mips->memory[index] = mips->registers[rt];
        }
    }
    else if (opcode == 2 || opcode == 3) {
        int address = (instr << 6) >> 6;
        char str1[5], str2[27] = "\t";
        if (opcode == 2) { mips->pc = address * 4; }
        else {
            *changedReg = 31;
            mips->registers[31] = mips->pc + 4;
            mips->pc = address * 4;
        }
    }
}
```

测试用例：

H:\proj1\project1\Debug\COD18-mips.exe

```
>
Executing instruction at 00400000: 00420021
        addu    $0, $2, $2
New pc = 00400004
Updated r0 to 00000000
No memory location was updated.

>
Executing instruction at 00400004: 24830005
        addiu   $3, $4, 5
New pc = 00400008
Updated r3 to 00000005
No memory location was updated.

>
Executing instruction at 00400008: 24e6fff8
        addiu   $6, $7, -8
New pc = 0040000c
Updated r6 to fffffff8
No memory location was updated.

>
Executing instruction at 0040000c: 012a4023
        subu    $8, $9, $10
New pc = 00400010
Updated r8 to 00000000
No memory location was updated.
```

课程实习时的指令测试：

Microsoft Visual Studio 调试控制台

```
>
Executing instruction at 00400004: 20110006
        addi    $17, $0, 6
New pc = 00400008
Updated r17 to 00000006
No memory location was updated.

>
Executing instruction at 00400008: 02002020
        add     $4, $16, $0
New pc = 0040000c
Updated r4 to 00000005
No memory location was updated.

>
Executing instruction at 0040000c: 02202820
        add     $5, $17, $0
New pc = 00400010
Updated r5 to 00000006
No memory location was updated.

>
Executing instruction at 00400010: 0c10000b
        jal     0x10000b
New pc = 0040002c
Updated r31 to 00400018
No memory location was updated.

>
Executing instruction at 0040002c: 20020000
        addi    $2, $0, 0
New pc = 00400030
Updated r2 to 00000000
No memory location was updated.

>
Executing instruction at 00400030: 20140004
        addi    $20, $0, 4
New pc = 00400034
Updated r20 to 00000004
No memory location was updated.

>
Executing instruction at 00400034: 20150008
        addi    $21, $0, 8
New pc = 00400038
Updated r21 to 00000008
No memory location was updated.

>
```

遇到的问题及解决方法：

1、 Disassembled 函数返回的 char*字符串输出来为乱码，经提示使用了全局变量来存储字符串；

2、 翻译出来的 MIPS 与答案的寄存器顺序不一样，经检查是因为 MIPS 的 R 格式通常是第一个寄存器来存储返回的结果，但是在机器码中，通常用最后一个

寄存器来存储结果；

3、 内存错误，发现是定义的字符串长度过短。

实习题目二

<概述题目要求>

本题需要使用 Logisim 来创建一个 16-位单时钟周期 CPU.

需要实现一个简单的 16-位处理器(即每个指令字长为 16 位,寄存器也是 16 位)，该处理器有四个寄存 器($r0 到$r3). 具有独立的数据和指令内存(即有两个内存,一个指令内存,一个数据内存)。

下表给出了指令编码表。通过查询 opcode 字段(高四位,即 15-12 位)的值，可知半字编码所对应的 指令 . 注意,表中的 opcode 不到 16 个,而 funct 也不到 8 个. 原因是指令少一些，使同学们更容易实现。

15-12	11	10	9	8	7	6	5	4	3	2	1	0	
0	rs		rt		rd		party bits!			funct			参见 R-type Instructions
1	rs		rt		immediate-u								disp: DISP[imm] = $rs
2	rs		rt		immediate-u								lui: $rt = imm << 8
3	rs		rt		immediate-u								ori: $rt = $rs | imm
4	rs		rt		immediate-s								addi: $rt = $rs + imm
5	rs		rt		immediate-u								andi: $rt = $rs & imm
6	rs		rt		immediate-s								lw: $rt = MEM[$rs + imm]
7	rs		rt		immediate-s								sw: MEM[$rs+imm] = $rt
8	jump address												jump
9	rs		rt		offset								beq
10	rs		rt		offset								bne

R-Type Instructions

funct	meaning
0	or: $rd = $rs | $rt
1	and: $rd = $rs & $rt
2	add: $rd = $rs + $rt
3	sub: $rd = $rs - $rt
4	sllv: $rd = $rs << $rt
5	srlv: $rd = $rs >> $rt
6	srav: $rd = $rs >> $rt
7	slt: $rd = ($rs < $rt) ? 1 : 0

<设计思路>

（1） 首先我设计了 PC 的逻辑，pc 需要的传入值位 jump、branch、zero、imm16、imm12、clk、Reset，pc 有一个独立的寄存器，指向当前执行指令或下一条指令，输出的是当前 pc 寄存器中的值，pc 的更新值对应为：

① Jump=0，branch=0，或 branch=1，zero=0：pc=pc+1；

② Jump=0，branch=1 且 zero=1：pc=pc+1+offset,此处 offset 需要位拓展；

③ Jump=1，branch=1，zero=x：PC = (PC & 0xF000) | address，同样这里需要一些拓展；

（2） 然后是寄存器：传入值为 RegWrite（是否需要写寄存器），Ra，Rb，Rw，W（需要写入的值），clk，Reset；传出值为 0123 三个低 8 位结果显示，和读取 Ra，Rb 的两个结果；

（3） 之后做了一个拓展器：支持零拓展和位拓展；

（4） ALU 的实现：利用了 logisim 自己提供的内建库实现，共支持 11 种运算，包括：与、或、加、减、逻辑左移、逻辑右移、算术右移、判等、判小于、判不等、8 位转高 8 位，运算方式由 ALUctr 控制；

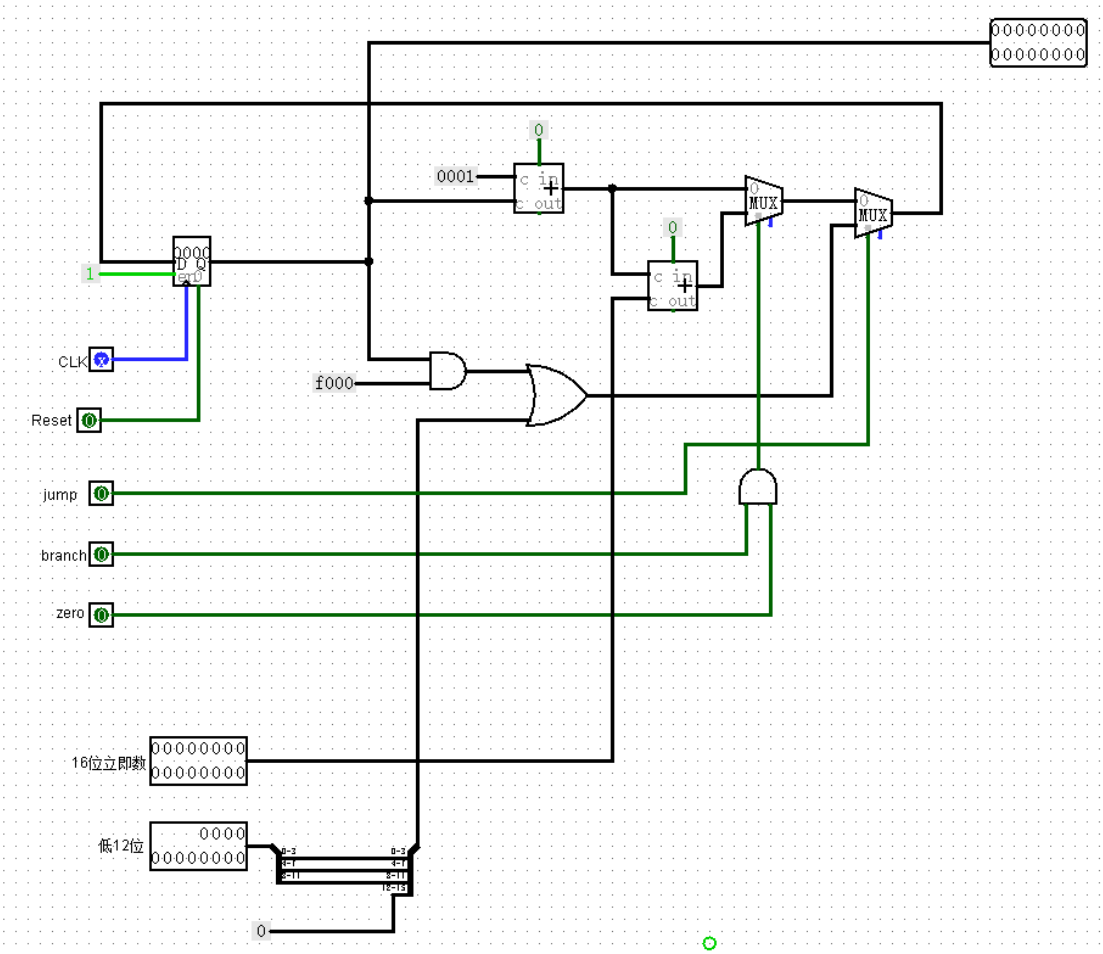
（5） 数据内存：利用 logisim 的 RAM 模块，实现数据的读写操作，此处由 MemWrite 控制；

（6） 控制器的实现：通过分析给出的指令集，做出相应的根据 opcode 和 funct 字段得出 ALUsrc、MemtoReg、RegWrite、MemWrite、branch、jump、extop、Regdst 和 ALUctr 的值，做出相应的真值表（在 logisim 中输入了真值表，忘记截图了😰），利用 logisim 的 combinational analysis 工具输入真值表绘制出相应的组合逻辑电路，为了简化操作，此处需要两个真值表，一个 opcode，一个 funct，然后需要做简单优化；

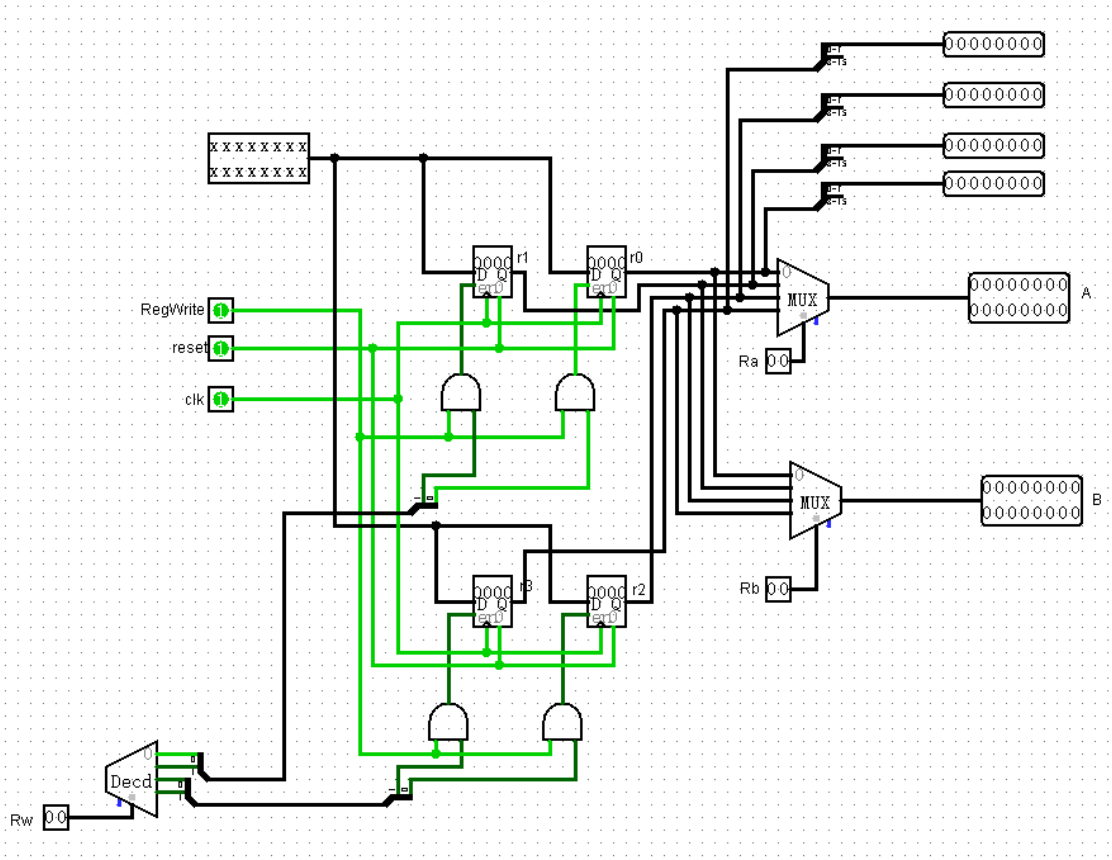
（7） 然后就是数据通路和控制通路的设计，此处参考书上的 32 位 cpu 设计。

<实现和测试>

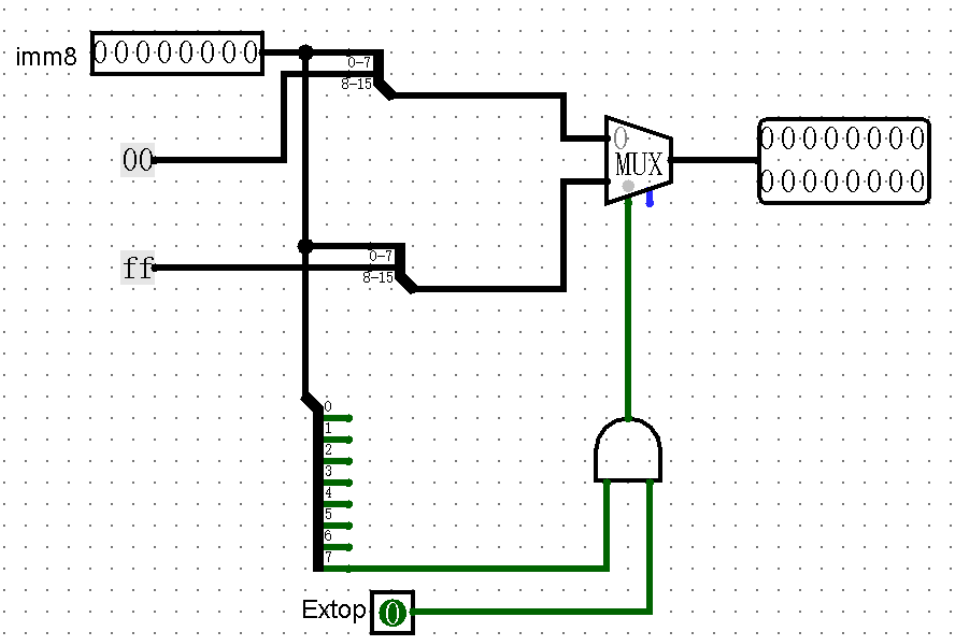
（1） NPC：


0001
MUX
MUX
1
CLK
Reset
f000
jump
branch
zero
16位立即数
低12位
0

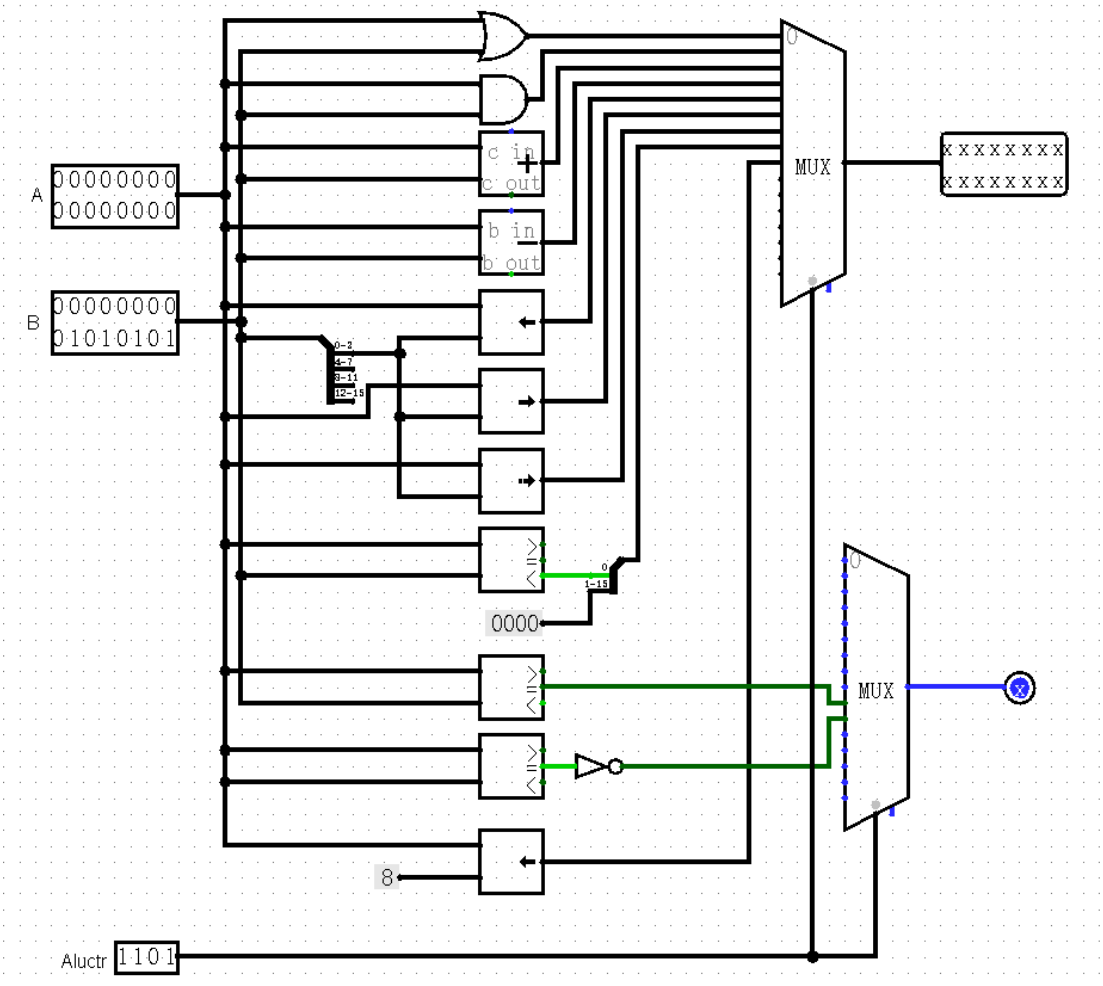
（2） Regs 寄存器文件：


r1
r0
RegWrite
reset
clk
MUX
A
Ra
MUX
B
Rb
r3
r2
Decd
Rw

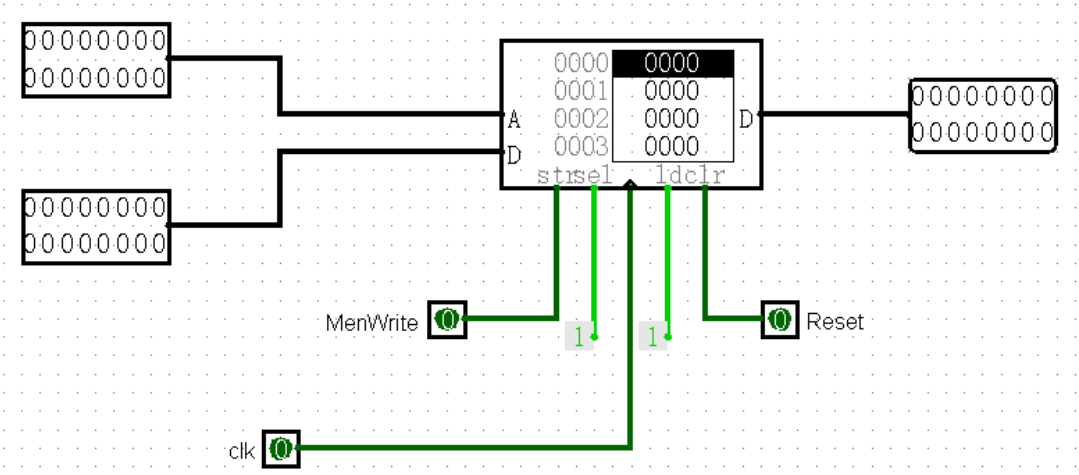
（3） 拓展器：


imm8
00000000
0-7
8-15
00
ff
0-7
8-15
0
MUX
00000000
00000000
0
1
2
3
4
5
6
7
Extop
0

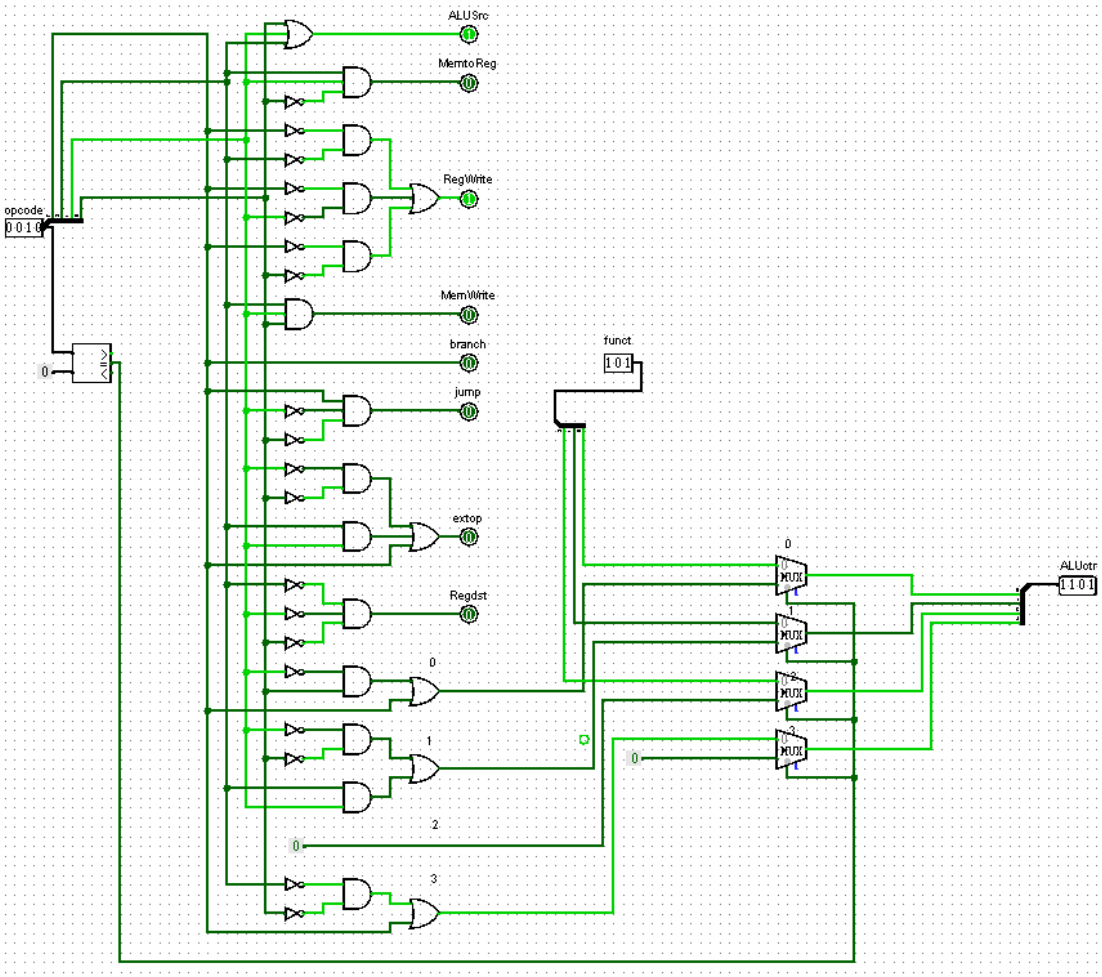
（4） ALU：


A
00000000
00000000
B
00000000
01010101
c_in
c_out
b_in
b_out
0-3
4-7
8-11
12-15
0
MUX
xxxxxxxx
xxxxxxxx
0000
0
1-15
0
MUX
X
8
Aluctr
1101

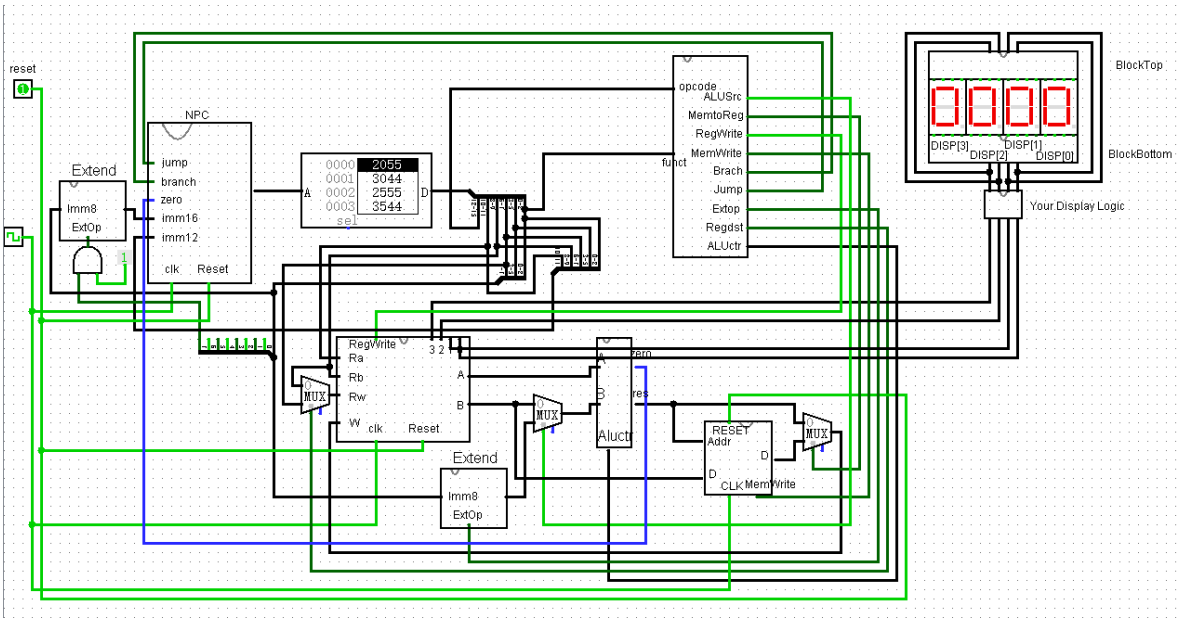
（5） 数据内存：



（6） 控制信号的实现：



（7） 连接之后：



至此，CPU 设计完成。

……

总　结

这次的课程设计对我说是很有难度的，特别是 cpu 的设计，因为之前没有接触过 logisim 这个软件，所以用起来的感觉就是无从下手，其中我还是有不理解的地方，比如 cpu 中的 disp 指令以及如何去显示寄存器的值但是通过这次课程设计，确实让我掌握了很多没有理解的知识，充分巩固了上课学的内容。

通过第一题，我更加深入的了解了汇编与反汇编的过程，以及 MIPS 指令工作时对应的相应寄存器/内存的变化，另外也加深了我对位操作的熟练程度；通过第二题，我更深刻地知道了 CPU 内部中的结构，虽然平时上课看过图，但实际动手操作起来还是非常有难度的，需要我们对其有充分的了解，慢慢的把 CPU 分解成多个小部件，再一个个的去实现，最后组装起来就变成了一个 CPU，虽然它现在还只是一个简单的 16 位单周期处理器模拟，但是我相信通过这次的课设，将来可以设计出更复杂的处理器。

总的来说，还是要多动手，不能只看书，也不能脱离书本，我们要把自己所学与实际相结合，才能做的更棒。