上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

计算机系统结构实验报告

实验名称：简单的类 MIPS 单周期处理器部件实现-整体调试

姓　　名：洪瑄锐

学　　号：517030910227

班　　级：F1703302

手　　机：15248246044

邮　　箱：1204378645@qq.com

目录

1. 实验目的

1) 完成单周期的类 MIPS 处理器

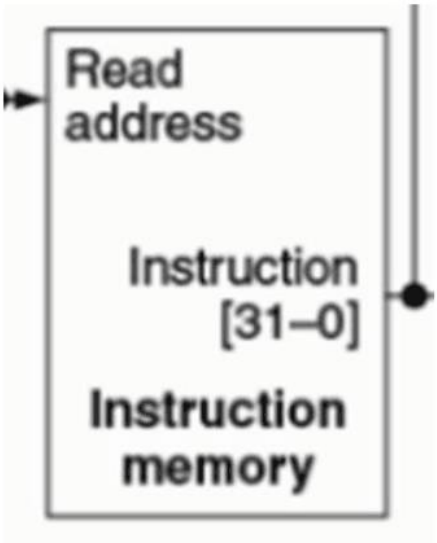
2) 设计支持条 MIPS 指令

2. 实验原理与代码实现

2.1 Instruction memory 的实现

2.1.1 主要思想

InstMemory 模块输入为所读指令的地址，输出为位指令。



2.1.2 InstMemory 模块代码

```
module InstMemory(
    input [31:0] ReadAddr,
    output [31:0] Inst
    );

    reg [31:0] memFile[63:0];
    reg [31:0] inst;

    initial
    begin
        $readmemb("C:/archlabs/project5/project5.srcs/mem_inst.txt",memFile,8'h0);
    end

    assign Inst = inst;

    always @(ReadAddr)
        begin
            inst <= memFile[ReadAddr >> 2];
        end
endmodule
```

2.2 单周期 CPU 的实现与调试

2.2.1 主要思想

单周期 CPU 的各个部件如 ALU,ALUCtr、Ctr、Register、Memory 等已经在之前的实验中写好了 lab05 所做的最主要的事情就是将各个部件通过顶层模块 Top 连接在一起，使之能够作为一个整体工作。

2.2.2 指令扩充

与 lab03 所写的 ALU、ALUCtr、Ctr 相比，本次实验加入了 addi、andi、ori、sll、srl 等指令，这就需要对原来的模块进行修改。

例如对于 andi 和 ori，在 Ctr 模块中根据 opCode 将两者的 aluOp 都赋值为 11。

```
6'b001100: //andi
begin
    regDst = 0;
    aluSrc = 1;
    memToReg = 0;
    regWrite = 1;
    memRead = 0;
    memWrite = 0;
    branch = 0;
    aluOp = 2'b11;
    jump = 0;
    zext = 0;
end
```

```
6'b001101: //ori
begin
    regDst = 0;
    aluSrc = 1;
    memToReg = 0;
    regWrite = 1;
    memRead = 0;
    memWrite = 0;
    branch = 0;
    aluOp = 2'b11;
    jump = 0;
    zext = 1;
end
```

在 ALUCtr 模块中，增加一个输入 opCode，根据 opCode 区分 addi 和 ori。

```
8'b11xxxxxx:
begin
    case(opCode)
    6'b001100:
        aluCtrOut = 4'b0000;//andi
    6'b001101:
        aluCtrOut = 4'b0001;//ori
    endcase
end
```

在 ALU 模块中，根据 aluCtrOut 的值指定 ALU 的运算。

```
4'b0000: //and
    begin
        aluRes = input1 & input2;
        if(aluRes == 0)
            zero = 1;
        else
            zero = 0;
    end
4'b0001: //or
    begin
        aluRes = input1 | input2;
        if(aluRes == 0)
            zero = 1;
        else
            zero = 0;
    end
```

其他添加指令代码：

在 Ctr 模块中 R 型指令对应 aluOp 为 10，左移 funct 为 000000，右移 funct 为 000010。

```
8'b10000000:
    aluCtrOut = 4'b0011;//左移

8'b10000010:
    aluCtrOut = 4'b0100;//右移
```

在 ALU 模块中：

```
4'b0011: //sll
    begin
        aluRes = input2 << shamt;
        if(aluRes == 0)
            zero = 1;
        else
            zero = 0;
    end
4'b0100: //srl
begin
    aluRes = input2 >> shamt;
    if(aluRes == 0)
        zero = 1;
    else
        zero = 0;
end
```

2.2.3 Top 模块

在 Top 模块中，我们需要为 CPU 所有部件的输出输出端口设置接线，并且将各模块实例化，通过设置的接线将各模块的输入输出连接起来，比如 Ctr 模块的 aluOp 输出是 ALUCtr 模块的输入，那么就要设置一根线 aluOp，在实例化中放在 Ctr 模块和 ALUCtr 模块的对应位置。

实例化代码：

```
Ctr mainCtr(INST[31:26], REG_DST, ALU_SRC, MEM_TO_REG, REG_WRITE, MEM_READ, MEM_WRITE, BRANCH, ALU_OP, JUMP, ZEXT);
dataMemory data(clk, DataAddr, WriteDataMem, MEM_WRITE, MEM_READ, ReadData);
InstMemory instMem(READ_ADDR, INST);
Registers regs(clk, READ_REG1, READ_REG2, WRITE_REG, WRITE_DATA, REG_WRITE, READ_DATA1, READ_DATA2, reset);
signext sig(INST[15:0], signext, ZEXT);
ALU alu(INPUT_1, INPUT_2, ALU_CTR, INST[10:6], ALU_ZERO, ALU_RES);
ALUCtr ctr(ALU_OP, INST[5:0],INST[31:26], ALU_CTR);
```

对应线连接代码：

```
assign INPUT_1 = READ_DATA1;
assign INPUT_2 = (ALU_SRC == 1'b0) ? READ_DATA2 : signext;

assign WRITE_REG = (REG_DST == 1'b1) ? INST[15:11] : INST[20:16];
assign WRITE_DATA = (MEM_TO_REG == 1'b1) ? ReadData : ALU_RES;

assign READ_REG1 = INST[25:21];
assign READ_REG2 = INST[20:16];

assign READ_ADDR = PC;

assign DataAddr = ALU_RES;
assign WriteDataMem = READ_DATA2;
```

此外，对于程序计数器 PC,设置一个寄存器变量存储 pc 的值，初始化为 0，每执行完一条指令根据上一条指令的类型而变化，如果是普通指令，则执行 pc+4，如果是 beq 指令，当 zero 信号为 1 即两个操作数相等时，pc+带符号扩展 32 位立即数，如果是 jump 指令，pc 前 4 位与立即数左移两位的结果拼接在一起。

代码：

```
always @ (posedge clk)
begin
    if (reset == 1) PC <= 0;
    else if (JUMP) PC <= {PC[31:28], (INST[25:0] << 2)};
    else if (BRANCH & ALU_ZERO) PC <= (signext << 2) + PC;
    else PC <= PC+4;
end

always @ (negedge clk)
begin
   if (reset == 1) PC = 0;
end
```

最后，对于 MUX,它应用在于 ALU 操作数的选择等地方。

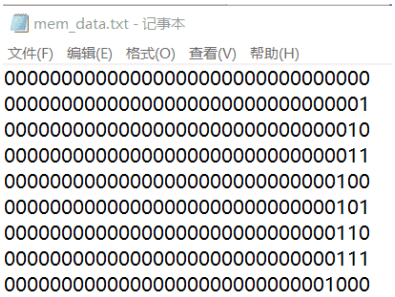
代码：

```
assign INPUT_2 = (ALU_SRC == 1'b0) ? READ_DATA2 : signext;
```

2.2.4 仿真测试

利用实验指导书中的$readmemh(“绝对路径”)给内存模块和指令内存模块赋值，通过查看寄存器中的值得知仿真正确。

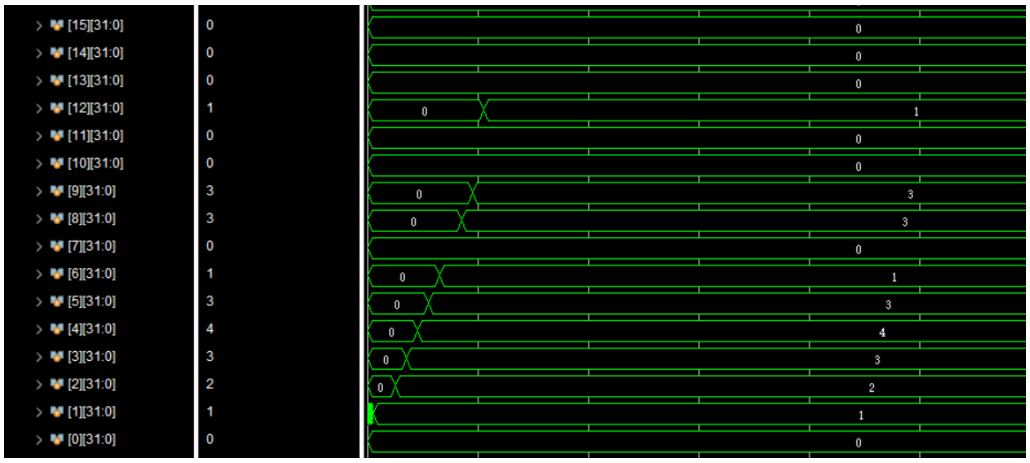
mem_data. txt：



mem_inst. txt：

```
10001100000000010000000000000001//lw $1,1($0)
10001100000000010000000000000010//lw $2,2($0)
10001100000000110000000000000011//lw $3,3($0)
10001100000001000000000000000100//lw $4,4($0)
00000000001000100010100000100000//add $5,$1,$2
00000000001000001001100000100010//sub $6,$2,$1
00000000001000100011100000100100//and $7,$1,$2
00000000001000100100000000100101//or $8,$1,$2
00100000001010010000000000000010//addi $9,$1,2
00000000001000100110000000101010//slt $12,$1,$2
00001000000000000000000000000100//jump
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000011001000110100000100000//add $13,$3,$4
```

仿真结果：



3. 感想

lab05 较上四次实验相比，难度有了很大的提高。在做实验的过程中，我思考了如何添加指令，如何读入 instrction 和 datamemory 的数据，如何设置 PC，最重要的是如何将各个端口没有遗漏的连接起来。经过这次实验，我了解了单周期 CPU 运行的方法，以及感受到了将各个模块的工作合在一起的力量。最后，感谢老师和同学在本次实验中给予我的经验和指导。

4. 参考文献

《cs145 实验指导书 lab05》