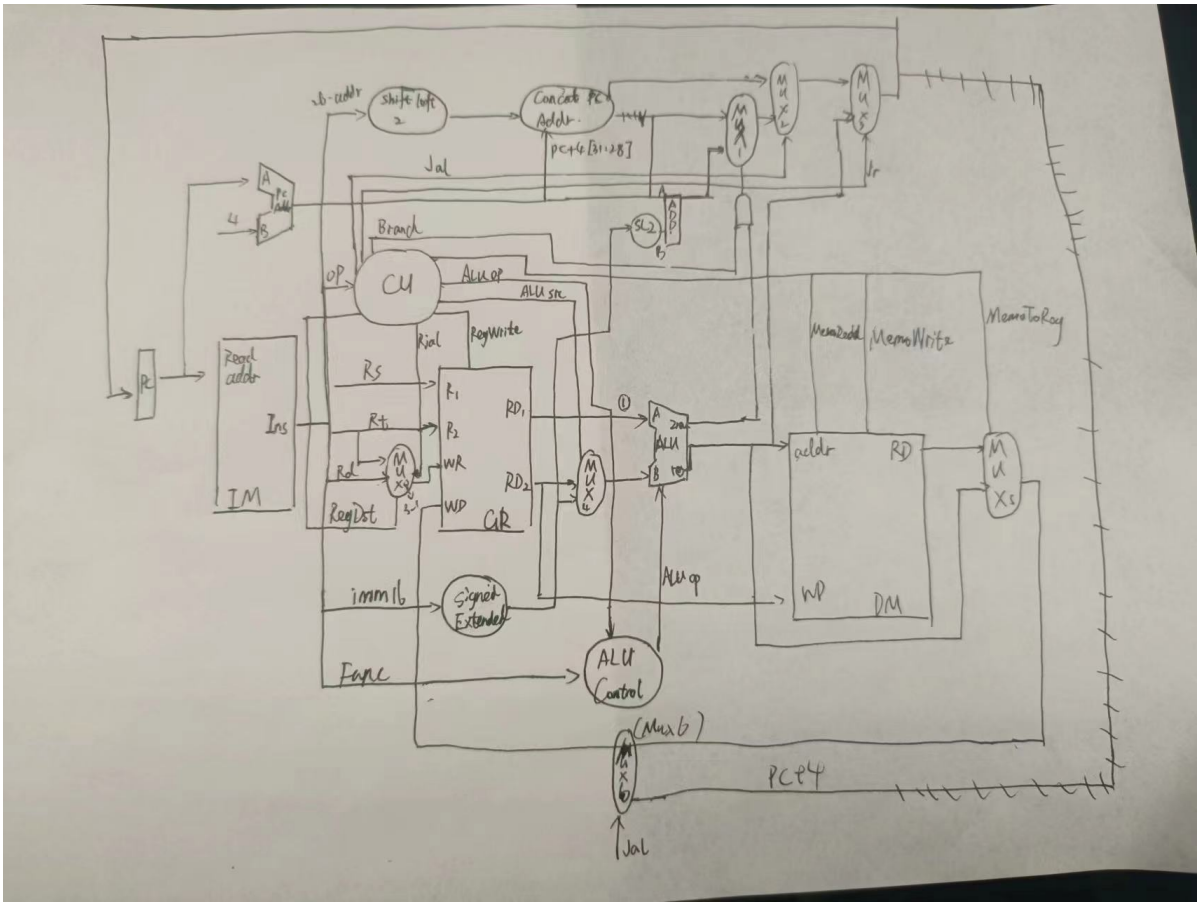
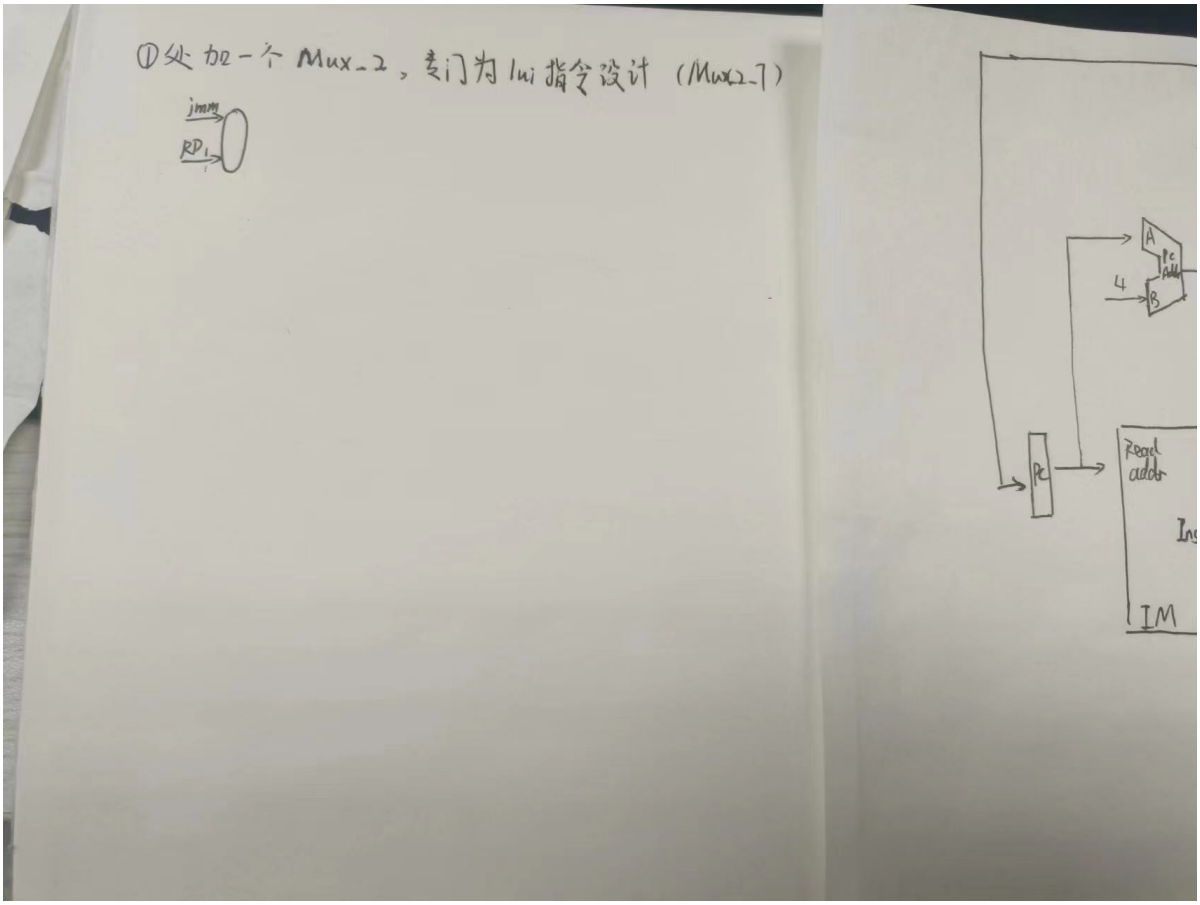
单周期CPU报告

1.整体架构





整体的架构参考了老师上课的PPT，在设计jal，jr和lui等指令时加入了自己的设计

2.控制信号实现

因为在自己调试的时候错误比较多，所以在每个指令执行时我将所有的控制信号都设置成了最保险的值，其实应该不用再每个指令执行时对所有控制信号进行操作，只是为了尽快将程序跑通，采用了比较保险的做法，之后可以优化。

addu, subu, jr

```
        6'b000000: begin     // addu, subu, jr
                case (func)
                    6'b100001: begin
                        alu_op_code <= 4'b0001; // addu
//                          $display("alu_control addu");
                        regWrite <= 1'b1;       // 需要写寄存器
                        regDst <= 3'b001;       // Mux选择信号，选择rd，rt和$31号寄存器中的一个
                        lui_mark <= 1'b0;       // 选寄存器
                        alu_src <= 1'b0;        // Mux选择信号，选择imm16和rt中的一个
                        jr <= 1'b0;             // 将jr的信号置为0
                        MemoToReg <= 1'b0;      // 没有发生DataMemory到寄存器的数据转移
                        jal <= 1'b0;            // 将jal信号置为0
                        MemoRead <= 0;          // 没有发生DataMemory读操作
                        MemoWrite <= 0;         // 没有发生DataMemory写操作
                        Branch <= 0;            // 没有发生分支跳转操作
                    end
                    6'b100011: begin
                        alu_op_code <= 4'b0010; // subu
                        regWrite <= 1'b1;
                        regDst <= 3'b001;
                        lui_mark <= 1'b0;       // Mux选择信号，在imm16和rs中选一个
                        alu_src <= 1'b0;
                        jr <= 1'b0;
                        regDst <= 3'b001;
                        MemoToReg <= 1'b0;
                        jal <= 1'b0;
                    end
                    6'b001000: begin
                        alu_op_code <= 4'b0101; // jr
                        regWrite <= 1'b0;
                        regDst <= 3'b001;
                        lui_mark <= 1'b0;      // 选寄存器
                        alu_src <= 1'b0;
                        jr <= 1'b1;              // Mux选择信号，选择属于jr指令的那个分支所传过来的数据
                        regDst <= 3'b001;
                        MemoToReg <= 1'b0;
                        MemoWrite <= 1'b0;
                        jal <= 1'b0;             // 这个信号其实无所谓，因为jr=1时选择器会直接选择另一条路，这条路的值并不会产生影响
                    end
                    endcase
//              $display("addu, subu, jr");
```

```
        end
```

ori

主要是设置寄存器写的信号

```
        6'b001101: begin    // ori
            alu_op_code <= 4'b0100;
            regWrite <= 1'b1;
            alu_src <= 1'b1;
            regDst <= 3'b010;
            MemoToReg <= 1'b0;
            MemoRead <= 1'b0;
            MemoWrite <= 1'b0;
            lui_mark <= 1'b0;
            Branch <= 1'b0;
            jal <= 1'b0;
            jr <= 1'b0;
//            $display("ori regDst");
        end
```

lw

将$DataMemory$到GPR的写信号设置为1，其余与之前相同

```
        6'b100011: begin    // lw
            alu_op_code <= 4'b0001;        //  0
            MemoWrite <= 1'b0;
            MemoRead <= 1'b1;
            MemoToReg <= 1'b1;
            Branch <= 0;
            jr <= 0;
            jal <= 0;
            regWrite <= 1'b1;
            regDst <= 3'b010;
            lui_mark <= 1'b0;
            alu_src <= 1'b1;
        end
```

sw

将写$DataMemory$的信号设置为1

```
        6'b101011: begin    // sw
            alu_op_code <= 4'b0001;
            MemoWrite <= 1'b1;
            MemoToReg <= 1'b0;
            MemoRead <= 1'b0;
            lui_mark <= 1'b0;      // 上面选rs
            alu_src <= 1'b1;       // 下面选imm
            regWrite <= 1'b0;
            jr <= 1'b0;
            jal <= 1'b0;
            regWrite <= 1'b0;
//            $display("sw");
        end
```

beq

将分支跳转的信号设置为1，并且将相关的Mux的选择信号设置成相应值

```

        6'b000100: begin    // beq
//            $display("BEQ BRANCH");
            alu_op_code <= 4'b0010;        // 1
            Branch <= 1'b1;
            alu_src <= 1'b0;
            lui_mark <= 1'b0;
            jal <= 1'b0;
            jr <= 1'b0;
            regWrite <= 1'b0;
            MemoToReg <= 0;
            MemoWrite <= 0;
            MemoRead <= 0;

        end
```

lui

注意将lui_mark信号置为1，选择$imm16$拓展为32位数之后的值作为ALU的传参数。

```
        6'b001111: begin    //   lui
            alu_op_code <= 4'b0011;        //
            regWrite <= 1'b1;
            lui_mark <= 1'b1;
            alu_src <= 1'b0;
            regDst <= 3'b010;
            MemoToReg <= 1'b0;
            MemoWrite <= 0;
            MemoRead <= 0;
            jal <= 0;
            jr <= 0;
            Branch <= 0;
//            $display("lui");
        end
```

jal

这里需要注意的是要把$pc+4$传回31号寄存器，这里我选择在CU模块里传出寄存器选择信号。

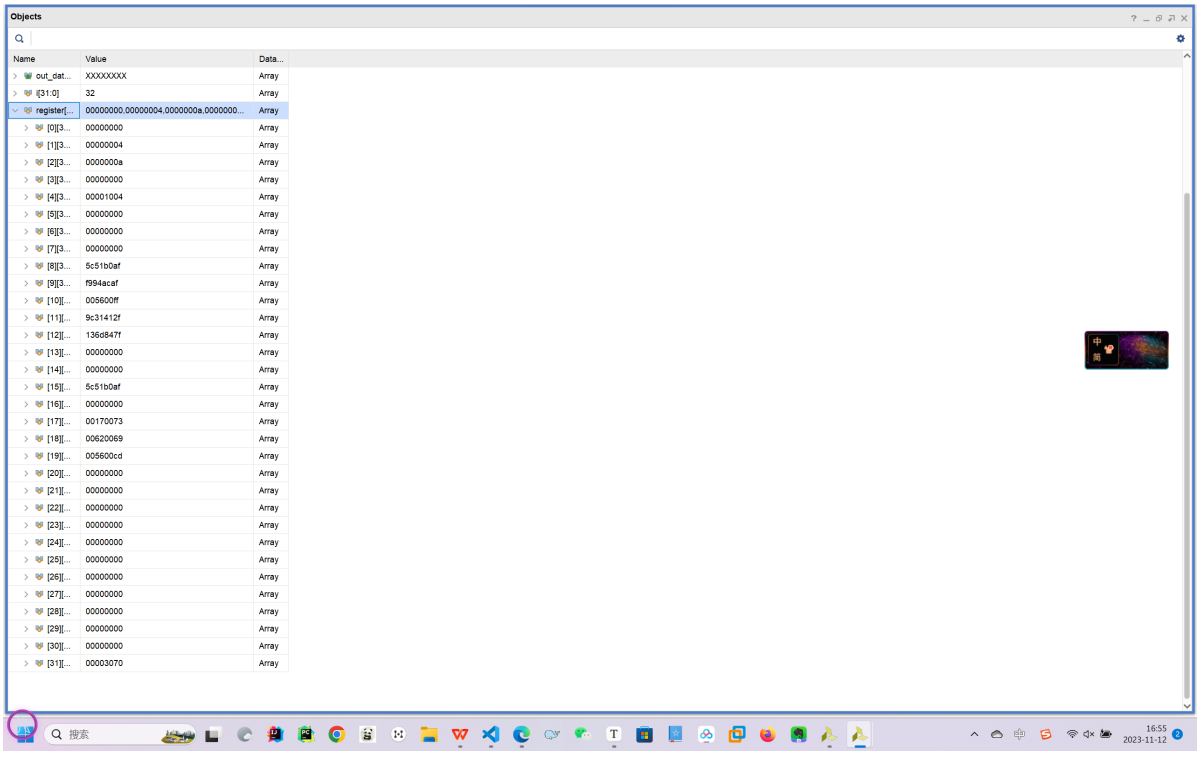
```
        6'b000011: begin   // jal
            jal_reg <= 5'b11111;
            Branch <= 1'b0;
            jr <= 1'b0;
            jal <= 1'b1;
            regDst <= 3'b100;
            MemoRead <= 0;
            MemoToReg <= 0;
            MemoWrite <= 0;
            regWrite <= 1'b1;
//            $display("jal");
        end
```

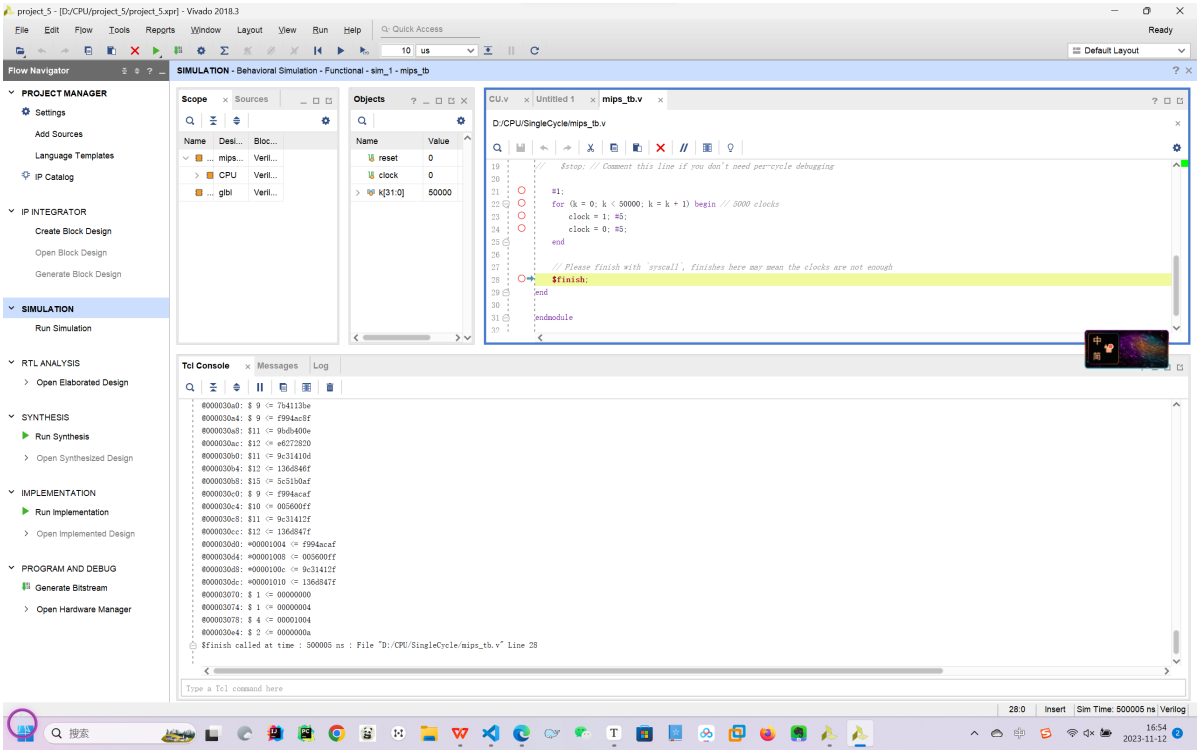
j

因为和jal指令的通路很相似，所以可以直接用$jal=1$在外加一些Mux选择信号直接实现。

```
        6'b000010: begin    // j
            jal <= 1'b1;
            regWrite <= 1'b0;
            Branch <= 1'b0;
            jr <= 1'b0;
            regDst <= 3'b100;
            MemoRead <= 0;
            MemoToReg <= 0;
            MemoWrite <= 0;
        end
```

3.实验结果




project_5 - [D:/CPU/project_5/project_5.xpr] - Vivado 2018.3
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - mips_tb
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Scope
Sources
Objects
Name
Value
reset
clock
k[31:0]
50000
CU.v
Untitled 1
mips_tb.v
D:/CPU/SingleCycle/mips_tb.v
// $stop; // Comment this line if you don't need per-cycle debugging
#1;
for (k = 0; k < 50000; k = k + 1) begin // 5000 clocks
clock = 1; #5;
clock = 0; #5;
end
// Please finish with `syscall`, finishes here may mean the clocks are not enough
$finish;
end
endmodule
Tcl Console
Messages
Log
$finish called at time : 500005 ns : File "D:/CPU/SingleCycle/mips_tb.v" Line 28
Type a Tcl command here
28:0
Insert
Sim Time: 500005 ns
Verilog
16:54
2023-11-12