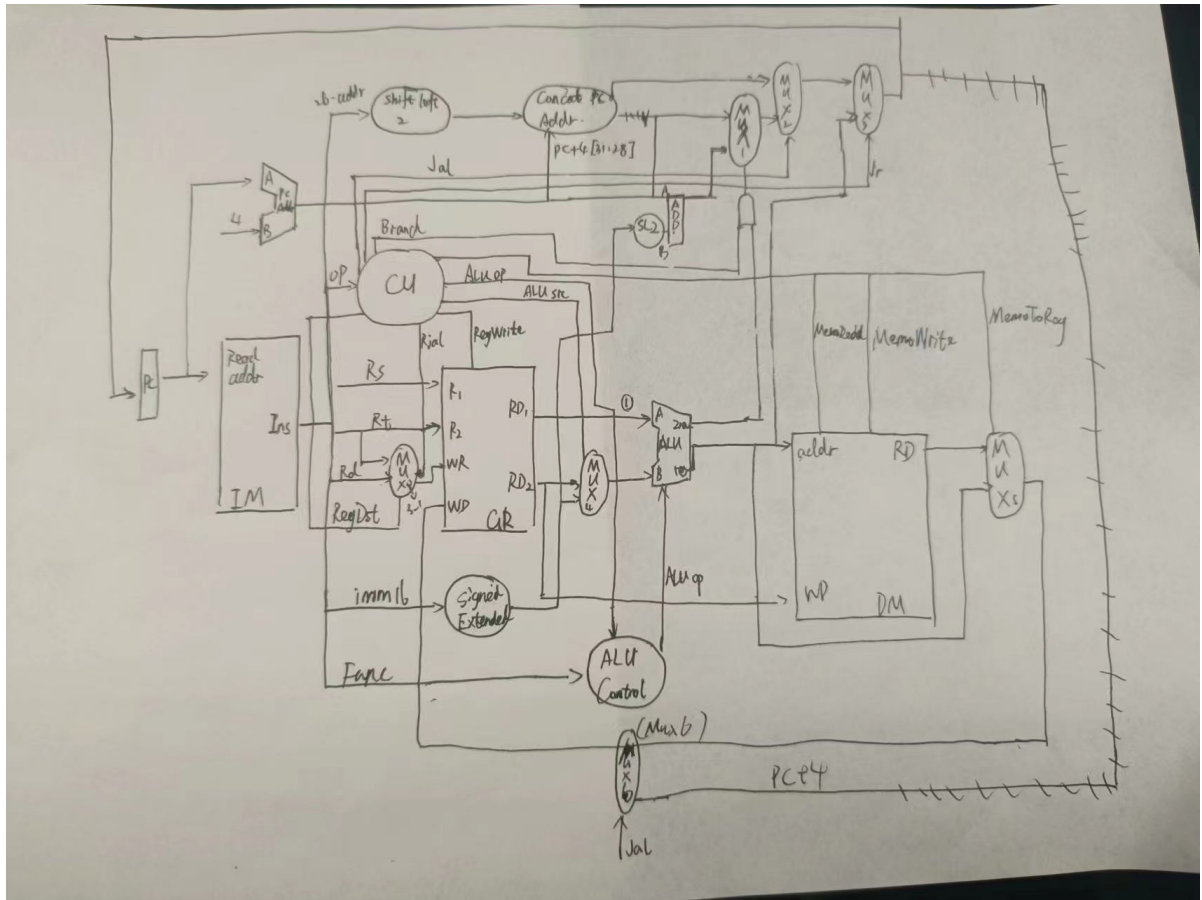
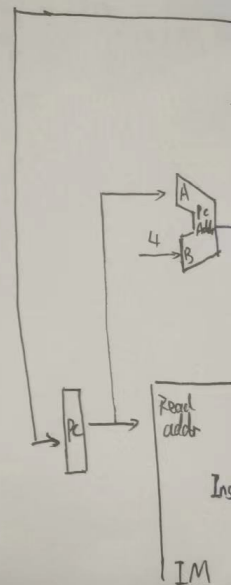


# 单周期CPU报告

## 1.整体架构



①处加一个 Mux-2, 专门为 lui 指令设计 (Mux-2-7)



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

```
1      6'b000000: begin          // addu, subu, jr
2          case (func)
3              6'b100001: begin
4                  alu_op_code <= 4'b0001; // addu
5              //                $display("alu_control addu");
6                  regwrite <= 1'b1;      // 需要写寄存器
7                  regDst <= 3'b001;      // Mux选择信号，选择rd, rt和$31号
//                  寄存器中的一个
8                  lui_mark <= 1'b0;      // 选寄存器
9                  alu_src <= 1'b0;      // Mux选择信号，选择imm16和rt中的
//                  一个
10                 jr <= 1'b0;            // 将jr的信号置为0
11                 MemoToReg <= 1'b0;     // 没有发生DataMemory到寄存器的数
//                  据转移
12                 jal <= 1'b0;           // 将jal信号置为0
13                 MemoRead <= 0;         // 没有发生DataMemory读操作
14                 Memowrite <= 0;        // 没有发生DataMemory写操作
15                 Branch <= 0;          // 没有发生分支跳转操作
16             end
17             6'b100011: begin
18                 alu_op_code <= 4'b0010; // subu
19                 regwrite <= 1'b1;
20                 regDst <= 3'b001;
21                 lui_mark <= 1'b0;      // Mux选择信号，在imm16和rs中选一
//                  个
22                 alu_src <= 1'b0;
23                 jr <= 1'b0;
24                 regDst <= 3'b001;
25                 MemoToReg <= 1'b0;
26                 jal <= 1'b0;
27             end
28             6'b001000: begin
29                 alu_op_code <= 4'b0101; // jr
30                 regwrite <= 1'b0;
31                 regDst <= 3'b001;
32                 lui_mark <= 1'b0;      // 选寄存器
33                 alu_src <= 1'b0;
34                 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
35                 regDst <= 3'b001;
36                 MemoToReg <= 1'b0;
37                 Memowrite <= 1'b0;
38                 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
39             end
40         endcase
41     //                $display("addu, subu, jr");
```

## ori

主要是设置寄存器写的信号

```
1      6'b001101: begin    // ori
2          alu_op_code <= 4'b0100;
3          regwrite <= 1'b1;
4          alu_src <= 1'b1;
5          regDst <= 3'b010;
6          MemoToReg <= 1'b0;
7          MemoRead <= 1'b0;
8          Memowrite <= 1'b0;
9          lui_mark <= 1'b0;
10         Branch <= 1'b0;
11         jal <= 1'b0;
12         jr <= 1'b0;
13     //         $display("ori regDst");
14     end
```

## lw

将*DataMemory*到*GPR*的写信号设置为1, 其余与之前相同

```
1      6'b100011: begin    // lw
2          alu_op_code <= 4'b0001;    // 0
3          Memowrite <= 1'b0;
4          MemoRead <= 1'b1;
5          MemoToReg <= 1'b1;
6          Branch <= 0;
7          jr <= 0;
8          jal <= 0;
9          regwrite <= 1'b1;
10         regDst <= 3'b010;
11         lui_mark <= 1'b0;
12         alu_src <= 1'b1;
13     end
```

## sw

将写*DataMemory*的信号设置为1

```

1      6'b101011: begin      // sw
2          alu_op_code <= 4'b0001;
3          MemoWrite <= 1'b1;
4          MemoToReg <= 1'b0;
5          MemoRead <= 1'b0;
6          lui_mark <= 1'b0;      // 上面选rs
7          alu_src <= 1'b1;      // 下面选imm
8          regWrite <= 1'b0;
9          jr <= 1'b0;
10         jal <= 1'b0;
11         regwrite <= 1'b0;
12     //         $display("sw");
13     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

```

1
2      6'b000100: begin      // beq
3  //         $display("BEQ BRANCH");
4          alu_op_code <= 4'b0010;      // 1
5          Branch <= 1'b1;
6          alu_src <= 1'b0;
7          lui_mark <= 1'b0;
8          jal <= 1'b0;
9          jr <= 1'b0;
10         regWrite <= 1'b0;
11         MemoToReg <= 0;
12         MemoWrite <= 0;
13         MemoRead <= 0;
14
15     end

```

## lui

注意将*lui\_mark*信号置为1，选择*imm*16拓展为32位数之后的值作为*ALU*的传参数。

```

1      6'b001111: begin      // lui
2          alu_op_code <= 4'b0011;      //
3          regWrite <= 1'b1;
4          lui_mark <= 1'b1;
5          alu_src <= 1'b0;
6          regDst <= 3'b010;
7          MemoToReg <= 1'b0;
8          MemoWrite <= 0;
9          MemoRead <= 0;
10         jal <= 0;
11         jr <= 0;
12         Branch <= 0;
13     //         $display("lui");
14     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。

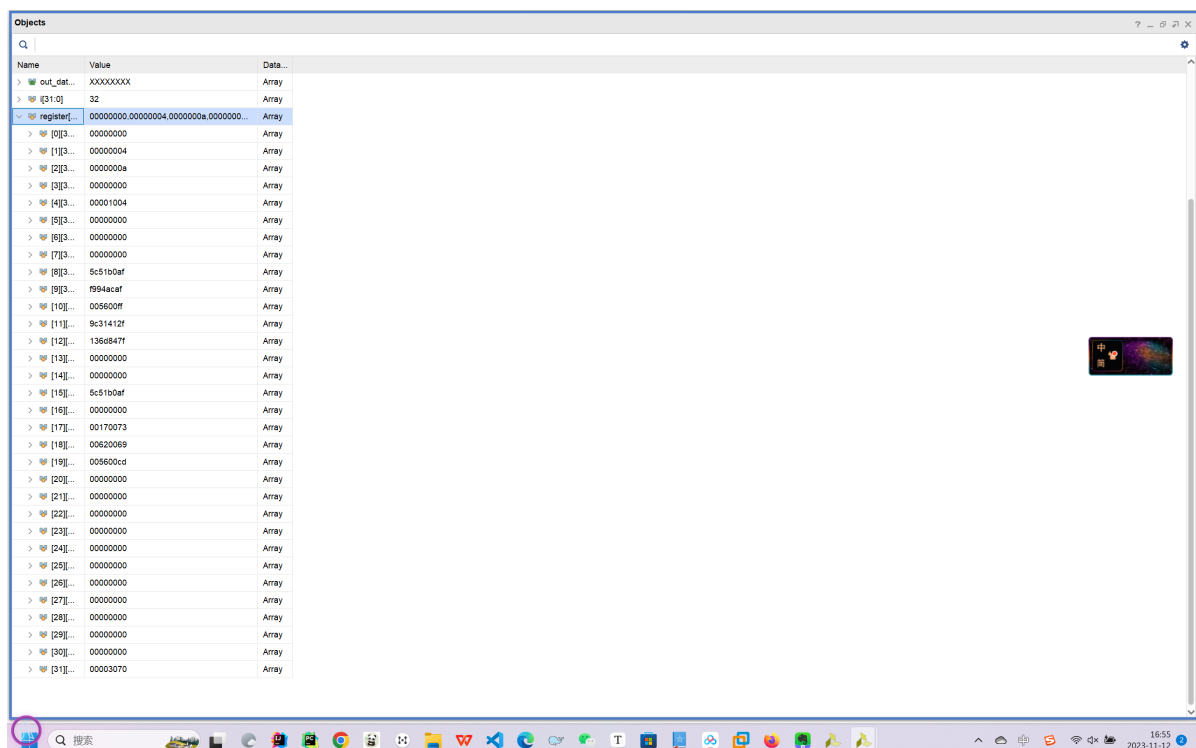
```
1      6'b000011: begin    // jal
2          jal_reg <= 5'b11111;
3          Branch <= 1'b0;
4          jr <= 1'b0;
5          jal <= 1'b1;
6          regDst <= 3'b100;
7          MemoRead <= 0;
8          MemoToReg <= 0;
9          Memowrite <= 0;
10         regWrite <= 1'b1;
11     //      $display("jal");
12     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。

```
1      6'b000010: begin    // j
2          jal <= 1'b1;
3          regWrite <= 1'b0;
4          Branch <= 1'b0;
5          jr <= 1'b0;
6          regDst <= 3'b100;
7          MemoRead <= 0;
8          MemoToReg <= 0;
9          Memowrite <= 0;
10     end
```

## 3.实验结果



project\_5 - [D:/CPU/project\_5/project\_5.apr] - Vivado 2018.3

File Edit Flow Tools Reports Window Layout View Run Help

Q Quick Access

10 us

Ready

Flow Navigator

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

Name	Dest...	Bloc...
mips...	Veril...	
CPU	Veril...	
gbl	Veril...	

Objects

Name	Value
reset	0
clock	0
K[31:0]	50000

CU.v x Untitled 1 x 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 < 30000; k = k + 1) begin // 3000 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 x Messages Log

```
8000030a0: $ 9 => 7b4113be
8000030a4: $ 9 => f994acff
8000030a8: $ 11 => 9b0b400e
8000030ac: $ 12 => a272350
8000030b0: $ 11 => 9c31410d
8000030b4: $ 12 => 136846ff
8000030b8: $ 15 => 5c5b0daf
8000030bc: $ 9 => f994acaf
8000030c0: $ 10 => 005600ff
8000030c4: $ 11 => 9c31412f
8000030c8: $ 12 => 136847f
8000030cc: *00001004 => f994acaf
8000030d0: *00001008 => 006000ff
8000030d4: *0000100c => 9c31412f
8000030d8: *00001010 => 136847f
8000030dc: $ 1 => 00000000
8000030e0: $ 1 => 00000004
8000030e4: $ 4 => 00001004
8000030e8: $ 2 => 0000000a
$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