中国科学院大学计算机组成原理实验课

实 验 报 告

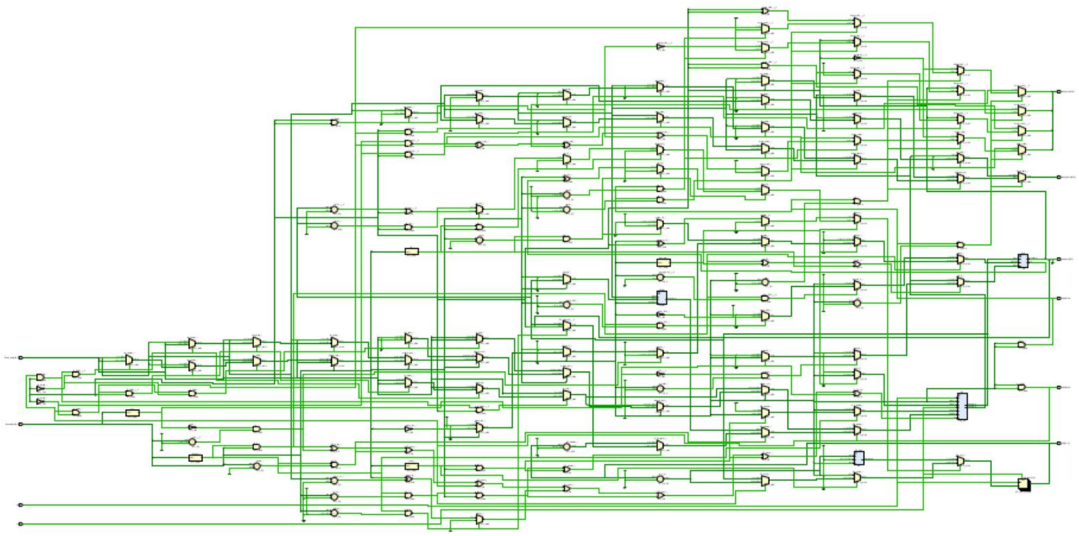
学号：____2018K8009929030____ 姓名：_热伊莱·图尔贡_ 专业：__计算机科学与技术__

实验序号：_02_ 实验名称：__单周期处理器设计__

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在本地仓库的主目录下。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：2019K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：学号-prj5-projectname.pdf，例如：2019K8009929000-prj5-dma.pdf。具体要求详见实验项目 5 讲义。
注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。
注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等）

原理图：



1. 通过寻找共同点定义指令 opcode。

```
assign RegDst = (opcode[5:0] == 6'b000000)?1:0;
assign R_C = (RegDst && (func[5] ==1'b1))?1:0;
assign Shift = (RegDst && (func[5:3] == 3'b000))?1:0;
assign R_J = (RegDst && ( {func[5:3],func[1]} == 4'b0010))?1:0;
assign Move = (RegDst && ( {func[5:3],func[1]} == 4'b0011))?1:0;
assign Regimm = (opcode [5:0] == 6'b000001)?1:0;
assign Jump = (opcode[5:1] == 5'b00001)?1:0;
assign I_B = (opcode [5:2] == 4'b0001)?1:0;
assign Lui = (opcode [5:0] == 6'b001111)?1:0;
assign I_C = (!Lui && (opcode [5:3] == 3'b001))?1:0;
assign Load = opcode[5]&(~opcode[3]);
assign Store = opcode[5]&opcode[3];
assign Jalr = (R_J && func[0])?1:0;
assign Jal = (Jump && opcode[0])?1:0;
assign MemRead=Load;
assign MemWrite=Store;
assign Address[31:0] = {alu_result [31:2],2'b0};
```

2. 指令手册中出现的各种扩展信号，先分别扩展，再根据具体指令来进行选择。

```
//extend
wire [31:0] extend;
wire [31:0] lui_extend;
wire [31:0] zero_extend;
wire [31:0] sign_extend;
assign lui_extend = {immediate[15:0],16'b0};
assign zero_extend = {16'b0,immediate[15:0]};
assign sign_extend = {{16{immediate[15]}},immediate[15:0]};
assign extend = (!Lui && (opcode[5:2] == 4'b0011))?zero_extend:(Lui?lui_extend:sign_extend);
```

3. LOAD 类型指令的数据选择，根据指令手册的图示以及小字部分实现。

```
//lb_data:11-[31:24];10-[23:16];01-[15:8];00-[7:0];
assign lb_data = (Load&alu_result[1]&&alu_result[0])?Read_data[31:24]:
                 (Load&alu_result[1]&&!alu_result[0])?Read_data[23:16]:
                 (Load&!alu_result[1]&&alu_result[0])?Read_data[15:8]:
                 Read_data[7:0];
//lh_data:00-[0:15];
assign lh_data = (Load&&!alu_result[1]&&!alu_result[0])?{{16{Read_data[15]}},Read_data[15:0]}:
{{16{Read_data[31]}},Read_data[31:16]};
assign lhu_data = (Load&&!alu_result[1]&&!alu_result[0])?{16'b0,Read_data[15:0]}:{16'b0,Read_data[31:16]};
wire [31:0] lwl_data;
wire [31:0] lwr_data;
//lwl_data:00-{[7:0],rt[23:0]};01:{[15:0],rt[15:0]};10:{[23:0],rt[7:0]};11:[31:0];
assign lwl_data = (Load&&!alu_result[1]&&!alu_result[0])?{Read_data[7:0],rdata2[23:0]}:
                  (Load&&!alu_result[1]&&alu_result[0])?{Read_data[15:0],rdata2[15:0]}:
                  (Load&&alu_result[1]&&!alu_result[0])?{Read_data[23:0],rdata2[7:0]}:
                  Read_data[31:0];
//lwr_data:00:[31:0];01{rt[31:24],[31:8]}..
assign lwr_data = (Load&&!alu_result[1]&&!alu_result[0])?Read_data[31:0]:
                  (Load&&!alu_result[1]&&alu_result[0])?{rdata2[31:24],Read_data[31:8]}:
                  (Load&&alu_result[1]&&!alu_result[0])?{rdata2[31:16],Read_data[31:16]}:
                  {rdata2[31:8],Read_data[31:24]};
```

```
wire [31:0] wdata_res;
//lb:lb(000),lbu(100);lh:lh(001),lhu(101);lwl(010);lwr(110)
assign wdata_res = (opcode == 6'b100000 )?{{24{lb_data[7]}},lb_data[7:0]}:
                   ( opcode == 6'b100100) ? {24'b0,lb_data[7:0]}:
                   (opcode == 6'b100001 ) ?lh_data:
                   (opcode == 6'b100101)?lhu_data:
                   (opcode[2:0] == 3'b010)?lwl_data:
                   (opcode[2:0] == 3'b110)?lwr_data:
                    Read_data[31:0];
```

4. strb 信号的译码以及 STORE 类型指令的数据选择，根据指令手册实现。

```
//swl_data
assign swl_data = (!alu_result[1]&&!alu_result[0])?{24'b0,rdata2[31:24]}:
                  (!alu_result[1]&& alu_result[0])?{16'b0,rdata2[31:16]}:
                  ( alu_result[1]&&!alu_result[0])?{ 8'b0,rdata2[31: 8]}:
                  ( alu_result[1]&& alu_result[0])?rdata2[31: 0]:0;
//swr_data
assign swr_data = (!alu_result[1]&&!alu_result[0])?rdata2[31:0]:
                  (!alu_result[1]&& alu_result[0])?{rdata2[23:0], 8'b0}:
                  ( alu_result[1]&&!alu_result[0])?{rdata2[15:0],16'b0}:
                  ( alu_result[1]&& alu_result[0])?{rdata2[ 7:0],24'b0}:0;
//sb_data
assign sb_data =  (!alu_result[1]&&!alu_result[0])?{24'b0,rdata2[7:0]}:
                  (!alu_result[1]&& alu_result[0])?{16'b0,rdata2[7:0], 8'b0}:
                  ( alu_result[1]&&!alu_result[0])?{ 8'b0,rdata2[7:0],16'b0}:
                  ( alu_result[1]&& alu_result[0])?{rdata2[7:0],24'b0}:0;
//sh_data
assign sh_data = alu_result[1]?{rdata2[15:0],16'b0}:
                !alu_result[1]?{16'b0,rdata2[15:0]}:0;
//Write_data
assign Write_data = sb?sb_data:sh?sh_data:swl?swl_data:swr?swr_data:rdata2;
```

mips_cpu.v × alu.v × shif

```
assign Write_strb[0]=sb?(!alu_result[1]&&!alu_result[0]):
                     sh?(!alu_result[1]):
                     swr?(!alu_result[1]&&!alu_result[0]):
                     swl?((!alu_result[1]&&!alu_result[0])|
                          (!alu_result[1]&& alu_result[0])|
                          ( alu_result[1]&&!alu_result[0])|
                          ( alu_result[1]&& alu_result[0])):1;
//Write_strb[1]:sb?01;sh?0x;swl?01,10,11;swr?00,01
assign Write_strb[1]=sb?(!alu_result[1]&& alu_result[0]):
                     sh?(!alu_result[1]):
                     swl?(alu_result[1] || alu_result[0]):
                     swr?(!alu_result[0]):
                     1;
//Write_strb[2]:sb?10;sh?1x;swl:10,11;swr:00,01,10;
assign Write_strb[2]=sb?( alu_result[1]&&!alu_result[0]):
                     sh?( alu_result[1]):
                     swl?( alu_result[1]):
                     swr?(!( alu_result[1]&& alu_result[0])):
                     1;
//Write_strb[3]:sb?11;sh?1x;swl?11;swr?xx
assign Write_strb[3]=sb?( alu_result[1]& alu_result[0]):
                     sh?( alu_result[1]):
                     swl?( alu_result[1]& alu_result[0]):
                     1:
```

5. PC 的赋值采用了时序逻辑，在每个时钟周期的上升沿进行判断，如果复位信号有效就将 PC 复位为 0，否则根据读取的指令对 PC 进行操作，如果是跳转分支类指令，就将 PC 赋为相应的值，否则给输出 PC+4 的值。

mips_cpu.v × alu.v ×

```
//PC
wire [31:0] PC_4;
wire [31:0] PC_result;
wire [31:0] PC_Branch;
wire [31:0] PC_next;
assign PC_4 = PC + 4;
assign PC_result = {PC_4[31:28],Instruction[25:0],2'b00};
assign PC_next = Branch?PC_Branch:
                 Jump?PC_result:
                 R_J?rdata1:
                 PC_4;
wire [31:0] PC_B;
assign PC_B={{14{immediate[15]}},immediate[15:0],2'b0};
//PC Adder
alu PC_add(
.A(PC_4),
.B(PC_B),
.ALUop(ADD),
.Overflow(),
.CarryOut(),
.Zero(),
.Result(PC_Branch)
);
```

```
always @(posedge clk) begin
        if(rst)
            PC<=32'b0;
        else
            PC<=PC_next;
end
```

6. 实例化 reg_file，alu 和移位器。

```
assign RF_wen = R_C|I_C|Shift|Load|move_c|Jalr|Jal|Lui;
assign RF_waddr = RegDst? rd : (Jal?32'd31:rt);
//RF_wdata--jalr:rd<-PC+8;mov:rd<-rs;jal:PC+8;
assign RF_wdata = Load? wdata_res:
                  Shift? Shift_result:
                  (Jalr|Jal)?PC+8:
                  Move?rdata1:
                  alu_result;;
assign raddr1 = rs;
assign raddr2 = Regimm? 5'b0:rt;
reg_file RF(
.clk(clk),
.rst(rst),
.waddr(RF_waddr),
.raddr1(raddr1),
.raddr2(raddr2),
.wen(RF_wen),
.wdata(RF_wdata),
.rdata1(rdata1),
.rdata2(rdata2)
);
```

对 aluop 编码：

由于我在 alu.v 文件里直接将 aluop 编码用 ADD,SUB 等直接用 parameter 定义，因此在 CPU 进行仿真时出现了错误。

```
parameter AND  =3'b000;
parameter OR   =3'b001;
parameter ADD  =3'b010;
parameter SUB  =3'b110;
parameter SLT  =3'b111;
parameter SLTU =3'b011;
parameter XOR  =3'b100;
parameter NOR  =3'b101;
```

因此在 cpu 文件中定义了 localparam

```
);
localparam AND  =3'b000;
localparam OR   =3'b001;
localparam ADD  =3'b010;
localparam SUB  =3'b110;
localparam SLT  =3'b111;
localparam SLTU =3'b011;
localparam XOR  =3'b100;
localparam NOR  =3'b101;
```

```
//aluop
wire [2:0]  aluop_RC;
wire [2:0]  aluop_Regimm;
wire [2:0]  aluop_IB;
wire [2:0]  aluop_IC;
assign aluop_RC = (R_C && (func[3:2] == 2'b00))?{func[1],2'b10}:
                  (R_C && (func[3:2] == 2'b01))?{func[1],1'b0,func[0]}:
                  {~func[0],2'b11};
assign aluop_Regimm = SLT;
assign aluop_IB = Ib1?SUB:SLT;
assign aluop_IC = (opcode[2:1] == 2'b00)?{opcode[1],2'b10}:
                  (opcode[2] == 1'b1)?{opcode[1],1'b0,opcode[0]}:
                  {~opcode[0],2'b11};

assign aluop = R_C?aluop_RC:
               Regimm?aluop_Regimm:
               I_B?aluop_IB:
               I_C?aluop_IC:
               ADD;
```

Alu 实例化：

```
//module instance-alu
alu ALU(
.A(alu_A),
.B(alu_B),
.ALUop(aluop),
.Overflow(overflow),
.CarryOut(carryout),
.Zero(zero),
.Result(alu_result)
);
```

移位器：

```
//Shifter
wire [31:0] shift_A;
wire [31:0] shift_B;
wire [1:0] shiftop;
wire [31:0] Shift_result;
assign shift_A = rdata2;
assign shift_B = (func[2]==0)?{27'b0,shamt[4:0]}:rdata1;
assign shiftop = func[1:0];
//module instance-shifter
shifter Shifter(
.A(shift_A),
.B(shift_B),
.Shiftop(shiftop),
.Result(Shift_result)
);
```

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真、云平台调试过程中的难点等）

1. 问题：没有分清 lb 指令和 lbu 指令之间的区别，用了同样的符号扩展。

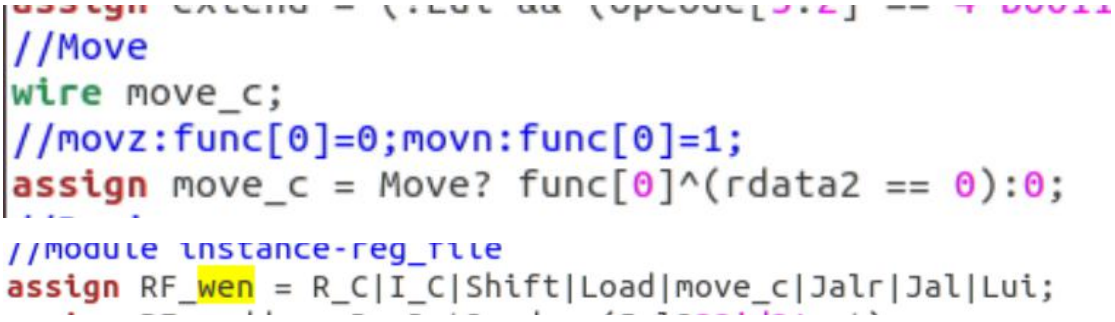
   解决方法：在仿真出错后，看到指令结果比对情况在王嵩岳助教的帮助下，重新看了 MIPS 指令集，将 lb 指令操作和 lbu 指令的 RF_wdata 结果分开描述。

2. 问题：没有分清 lh 指令和 lhu 指令之间的区别，用了同样的符号扩展。

   解决方法：在仿真出错后，看到指令结果比对情况在王嵩岳助教的帮助下，重新看了 MIPS 指令集，将 lh 指令操作和 lhu 指令的 RF_wdata 结果分开描述。

3. 问题：在 move 指令时，wen 写使能有效的选择出现问题。

   解决方法：将 movz 和 movn 指令操作用如下代码表示。



三、 对讲义中思考题（如有）的理解和回答

四、 在课后，你花费了大约____42_____小时完成此次实验。

五、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

在本次实验中，我深刻明白了 verilog 和 C 语言之间的区别。通过本次实验我对 Verilog 语言是电路模块之间的连接选择有了更深的认识。

实验本身并不复杂，但是由于我一开始没有区分好 c 语言和 Verilog 语言导致我的代码 debug 起来非常麻烦。同时本次实验最重要的是认真理解指令集小字部分操作，我因为 load 指令只看了其中一个的定义就字面理解式的将操作写了上去，因此到最后在王嵩岳助教的帮助下才发现 lb，lbu，lh，lhu 指令的问题。

最后非常感谢王嵩岳助教帮我 debug 以及让我学会了如何看波形 debug，非常感谢！