中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2018K8009929021　姓名：袁欣怡　专业：计算机科学与技术

实验序号：Prj2　实验名称：单周期处理器设计

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）

1. Mips_core 部分

```
assign Addiu=(op==6'b001001);
assign Addu=(op==6'b000000)&&(funct==6'b100001);
assign Subu=(op==6'b000000)&&(funct==6'b100011);
assign And=(op==6'b000000)&&(funct==6'b100100);
assign Andi=(op==6'b001100);
assign Nor=(op==6'b000000)&&(funct==6'b100111);
assign Or=(op==6'b000000)&&(funct==6'b100101);
assign Ori=(op==6'b001101);
assign Xor=(op==6'b000000)&&(funct==6'b100110);
assign Xori=(op==6'b001110);
assign Slt=(op==6'b000000)&&(funct==6'b101010);
assign Slti=(op==6'b001010);
assign Sltu=(op==6'b000000)&&(funct==6'b101011);
assign Sltiu=(op==6'b001011);
assign Sll=(op==6'b000000)&&(funct==6'b000000);
assign Sllv=(op==6'b000000)&&(funct==6'b000100);
assign Sra=(op==6'b000000)&&(funct==6'b000011);
assign Srav=(op==6'b000000)&&(funct==6'b000111);
assign Srl=(op==6'b000000)&&(funct==6'b000010);
assign Srlv=(op==6'b000000)&&(funct==6'b000110);
```

图 1

```
assign op=Instruction[31:26];
assign rs=Instruction[25:21];
assign rt=Instruction[20:16];
assign rd=Instruction[15:11];
assign shamt=Instruction[10:6];
assign funct=Instruction[5:0];
```

图 2

图 1 展示程序中设置的 44 个 reg 信号中的一部分，它们用来记录当前运行的指令。其中的 op 信号和 funct 信号均从 Instruction 中读出。同时读出的信号还有 rs、rt、rd 和 shamt（见图 2）

```
        assign extend1=extend<<2;
        alu alu1(next1,extend1,3'b010,Overflow1,Carryout1,Zero1,PC1);
        assign PCnext=(J||Jal||Jalr||Jr)?next:(Branch&&!addr_sign&&!Zero)?PC1:
(Branch&&addr_sign&&Zero)?PC1:next1;

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

        assign next1=PC+32'd4;
        assign next=((Jr||Jalr)==1)?A:{next1[31:28],Instruction[25:0],2'b00};
```

图 3

```
reg_file reg_file1(clk,rst,RF_waddr,rs,rt,RF_wen,RF_wdata,A,B);
```

图 4

图 3 展示了 PC 的计算方法。在 always 时序逻辑中，给不需要重置的 PC 赋值 PCnext。PCnext 有三种可能，next 表示跳转指令时的 PCnext，PC1 表示 branch 指令时的 PCnext，next 表示其余时刻的 PCnext。

图 3 和以后图中涉及到的 A 和 B 的定义在图 4 中显示，读出数据的地址为 rs 和 rt，当与某指令的具体要求不同时再做修改调整。

```
assign W_strb_sb=(Result[1:0]==2'b00)?4'b0001:
                 (Result[1:0]==2'b01)?4'b0010:
                 (Result[1:0]==2'b10)?4'b0100:
                 (Result[1:0]==2'b11)?4'b1000:0;
assign W_strb_sh=(Result[1]==1'b0)?4'b0011:
                 (Result[1]==1'b1)?4'b1100:0;
assign W_strb_swl=(Result[1:0]==2'b00)?4'b0001:
                  (Result[1:0]==2'b01)?4'b0011:
                  (Result[1:0]==2'b10)?4'b0111:
                  (Result[1:0]==2'b11)?4'b1111:0;
assign W_strb_swr=(Result[1:0]==2'b00)?4'b1111:
                  (Result[1:0]==2'b01)?4'b1110:
                  (Result[1:0]==2'b10)?4'b1100:
                  (Result[1:0]==2'b11)?4'b1000:0;
assign Write_strb=(Sb==1)?W_strb_sb:
                  (Sh==1)?W_strb_sh:
                  (Swl==1)?W_strb_swl:
                  (Swr==1)?W_strb_swr:
                  (Xori==1)?0:4'b1111;
```

图 5

```
assign W_data_sb=(Write_strb==4'b0001)?{24'd0,B[7:0]}:
                 (Write_strb==4'b0010)?{16'd0,B[7:0],8'd0}:
                 (Write_strb==4'b0100)?{8'd0,B[7:0],16'd0}:
                 (Write_strb==4'b1000)?{B[7:0],24'd0}:0;
assign W_data_sh=(Write_strb==4'b0011)?{16'd0,B[15:0]}:
                 (Write_strb==4'b1100)?{B[15:0],16'd0}:0;
assign W_data_swl=(Write_strb==4'b0001)?{24'd0,B[31:24]}:
                  (Write_strb == 4'b0011)?{16'd0,B[31:16]}:
                  (Write_strb == 4'b0111)?{8'd0,B[31:8]}:
                  (Write_strb == 4'b1111)?{B[31:0]}:0;
assign W_data_swr=(Write_strb == 4'b1000)?{B[7:0],24'b0}:
                  (Write_strb == 4'b1100)?{B[15:0],8'b0}:
                  (Write_strb == 4'b1110)?{B[23:0],8'd0}:
                  (Write_strb == 4'b1111)?{B[31:0]}:0;
assign Write_data=(Sb==1)? W_data_sb:
                  (Sh==1)?W_data_sh:
                  (Swl==1)?W_data_swl:
                  (Swr==1)?W_data_swr:B;
```

图 6

图 5&6 为写入内存的数据 Write_data 和位数 Write_strb 的赋值语句。因为 sb 等指令的情况比较复杂，因此多用了几个寄存器来写，方便检查。

```
        assign RF_wen=(Addiu||Addu||Subu||Sll||Sllv||Slt||Slti||Sltu||Sltiu||And||
Andi||Or||Ori||Nor||Xor||Xori||Jal||Jalr||(Lw&&Address[1:0]==2'b0)||Lui||Lb||Lbu||
Lh||Lhu||Lwl||Lwr||(Movn&&!Zero)||(Movz&&Zero)||Sra||Srav||Srl||Srlv);

        assign addr_sign=(Sll||Sllv||Sltu||Slt||And||Or||Xor||Nor||Addu||Beq||Bgez||
Sra||Srav||Subu||Srl||Srlv||Movz||Movn||Jalr);
        assign RF_waddr=(Jal)?5'b11111:(addr_sign)?rd:rt;
```

图 7

```
assign data1=(Lb==1)?data1_lb:
                     (Lbu==1)?data1_lbu:
                     (Lh==1)?data1_lh:
                     (Lhu==1)?data1_lhu:
                     (Lwl==1)?data1_lwl:
                     (Lwr==1)?data1_lwr:
                     (Lw==1)?data1_lw:
                     0;
assign RF_wdata=(Nor==1)?~Result:
                               (Srl==1)?(B>>shamt):
                               (Srlv==1)?(B>>A[4:0]):
                               (Sll==1)?(B<<shamt):
                               (Sllv==1)?(B<<A[4:0]):
                               ((Jal||Jalr)==1)?next1+32'd4:
                               ((Movn||Movz)==1)?A:
                               (Lui==1)?{Instruction[15:0],16'd0}:
                               (MemRead==1)?data1:
                               Result;
```

图 8

图 7&8 是在图 4 例化 reg_file 时涉及的 RF_wen, RF_waddr 和 RF_wdata 的赋值语句。图 8 中 data1 的定义中，为了检查方便采用了和图 5、图 6 一

样的赋值方式。

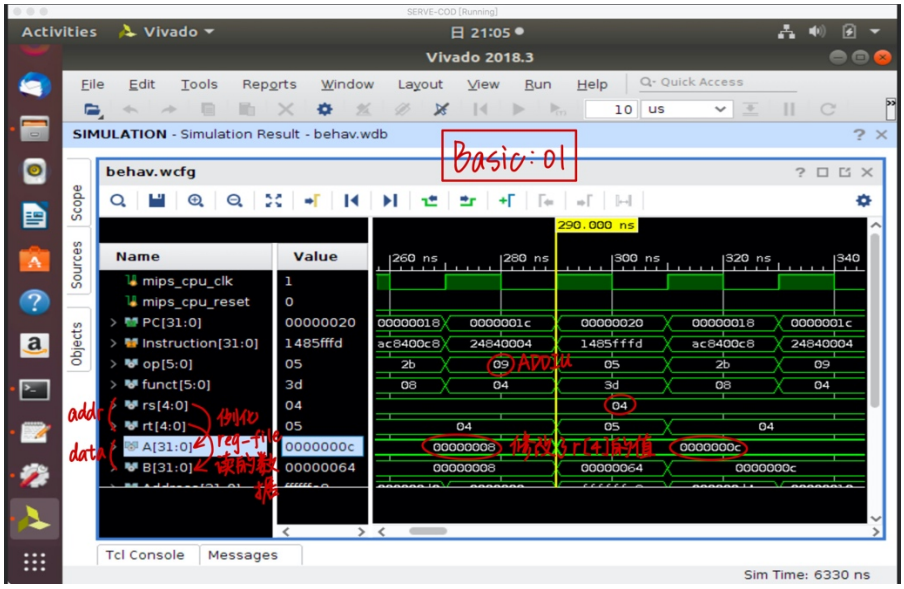
2. Alu 部分

```
assign ResultAND=A & B;

assign ResultOR=A | B;

assign ResultXOR=A ^ B;

assign SIGNED=~{B[31],B}+33'd1;
assign SIGNED1=(ALUop==3'b010)?{B[31],B}:SIGNED;
assign UNSIGNED=~{1'b0,B}+33'd1;
assign UNSIGNED1=(ALUop==3'b010)?{1'b0,B}:UNSIGNED;

assign temp={32'hffffffff,A}>>B;
assign ResultRight=(A[31]==1)?temp[31:0]:A>>B;

assign calculate={A[31],A}+SIGNED1;
assign calculate1={1'b0,A}+UNSIGNED1;

assign Overflow=calculate[32]^calculate[31];
assign CarryOut=calculate1[32];

assign Result=(ALUop==3'b000)?ResultAND:(ALUop==3'b001)?ResultOR:(ALUop==3'b010)?
calculate[31:0]:(ALUop==3'b011)?ResultXOR:(ALUop==3'b100)?ResultRight:(ALUop==3'b101)?
CarryOut:(ALUop==3'b110)?calculate[31:0]:(ALUop==3'b111)?{31'b0,calculate[31]^Overflow}:
0;

assign Zero=(Result==0)?1:0;
```

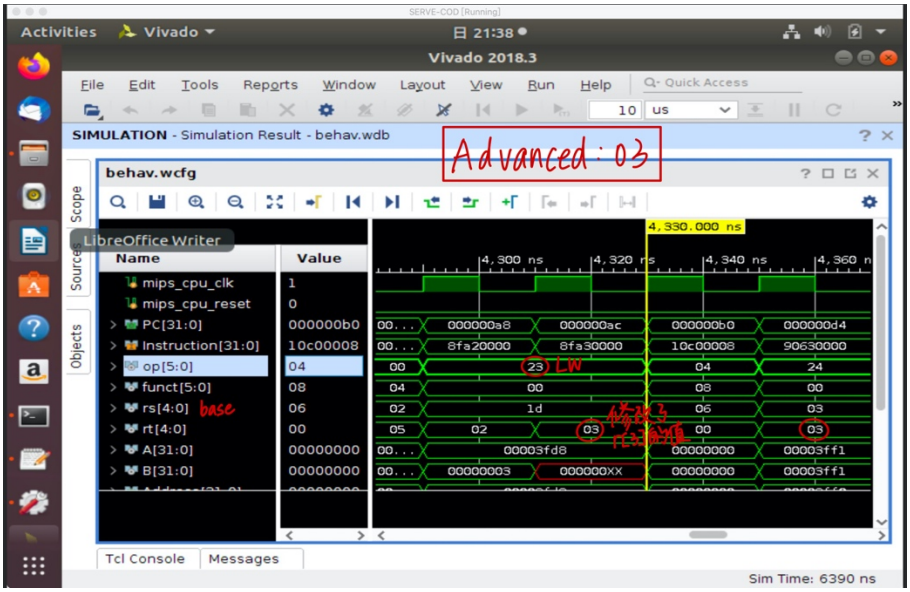
和 prj1 中的 alu 相比，这里添加了三个功能：异或、右移和进位。同时修改了之前的写法，用了 unsigned 和 signed，更方便阅读。

3. 举两个指令为例，展示其功能。

举 basic:01 中的 addiu 为例，修改了寄存器 r[4]的值。



举 advanced:03 中的 lw 为例，修改了寄存器 r[3]的值。



4. 代码原理图附在文档最后。

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、本地上板及云平台调试过程中的难点等）

1. 在处理纷繁复杂的指令时容易出现小错误。这是因为对 mips 指令的功能还不完全熟悉。通过整理每条指令对应的操作可以解决此问题。
2. 仿真中需要查看的信号比较多，且 reference 给出的信号有限，在调试时找到出错的信号比较困难。这时应该参考 mips 手册，查询指令的功能，再和自己的信号对照着看。
3. 有时出错不一定只与当前周期有关，也可能是之前的操作没有完成到位。比如我有一次出错是因为前几个周期写数据时没有写成功，但是一直盯着最后一个周期看，浪费了很多时间。
4. 在验收的时候，我因为写的注释较少，而且为了节约寄存器，将几个逻辑上无关的操作放在一起处理，导致最后的代码难理解，我也因为记忆模糊没法讲得很清楚。助教老师指出了这个问题。我以后要注意写注释，而且要注重代码的逻辑性。

三、 对讲义中思考题（如有）的理解和回答

无。

四、 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）

感觉比之前的难了好多（哈哈哈）。一开始感觉挺没有头绪的……可能因为当时章老师班还没讲到单周期处理器。写程序的时候比较仓促，没有来得及完成多周期处理器，这是比较可惜的……

感谢助教老师给我提供了一部分正确的波形图作参考，帮我解决了一个很久都没解决掉的 bug。