中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2017K8009929044　姓名：李昊宸　专业：计算机科学与技术

实验序号：2　实验名称：单周期处理器设计

注 1：请在实验项目个人本地仓库中创建顶层目录 doc。撰写此 Word 格式实验报告后以 PDF 格式保存在 doc 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 5 的阿拉伯数字。例如：2015K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。

注 2：使用 git add 命令将 doc 目录下的实验报告 PDF 文件添加到本地仓库，然后 git push 推送提交。

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）

关键代码段：

1） 控制单元

```
assign op_ADDIU =   (op == 6'b001001);
assign op_BNE   =   (op == 6'b000101);
assign op_LW    =   (op == 6'b100011);
assign op_SW    =   (op == 6'b101011);
assign op_SLL   =   (op == 6'b000000)&&( func == 6'b000000);
assign op_ADDU  =   (op == 6'b000000)&&( func == 6'b100001);
assign op_BEQ   =   (op == 6'b000100);
assign op_J     =   (op == 6'b000010);
assign op_JAL   =   (op == 6'b000011);
assign op_JR    =   (op == 6'b000000)&&( func == 6'b001000);
assign op_LUI   =   (op == 6'b001111);
assign op_OR    =   (op == 6'b000000)&&( func == 6'b100101);
assign op_SLT   =   (op == 6'b000000)&&( func == 6'b101010);
assign op_SLTI  =   (op == 6'b001010);
assign op_SLTIU =   (op == 6'b001011);

assign op_AND   =   (op == 6'b000000)&&( func == 6'b100100);
assign op_ANDI =   (op == 6'b001100);
assign op_BGEZ =   (op == 6'b000001)&&( rt == 5'b00001);
assign op_BLEZ =   (op == 6'b000110);
assign op_BLTZ =   (op == 6'b000001);
assign op_JALR =   (op == 6'b000000)&&( func == 6'b001001);
assign op_LB   =   (op == 6'b100000);
assign op_LBU  =   (op == 6'b100100);
assign op_LH   =   (op == 6'b100001);
assign op_LHU  =   (op == 6'b100101);
assign op_LWL  =   (op == 6'b100010);
assign op_LWR  =   (op == 6'b100110);
assign op_MOVN =   (op == 6'b000000)&&( func == 6'b001011);
assign op_MOVZ =   (op == 6'b000000)&&( func == 6'b001010);
assign op_NOR  =   (op == 6'b000000)&&( func == 6'b100111);
assign op_ORI  =   (op == 6'b001101);
assign op_SB   =   (op == 6'b101000);
assign op_SH   =   (op == 6'b101001);
assign op_SWL  =   (op == 6'b101010);
assign op_SWR  =   (op == 6'b101110);
assign op_SLLV =   (op == 6'b000000)&&( func == 6'b000100);
assign op_SLTU =   (op == 6'b000000)&&( func == 6'b101011);
assign op_SRA  =   (op == 6'b000000)&&( func == 6'b000011);
assign op_SRAV =   (op == 6'b000000)&&( func == 6'b000111);
assign op_SRL  =   (op == 6'b000000)&&( func == 6'b000010);
assign op_SRLV =   (op == 6'b000000)&&( func == 6'b000110);
assign op_SUBU =   (op == 6'b000000)&&( func == 6'b100011);
assign op_XOR  =   (op == 6'b000000)&&( func == 6'b100110);
assign op_XORI =   (op == 6'b001110);
```

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

将 Instruction 信号切片，赋值给 op、rs、rt、rd、sa、func 五个信号段。之后使用条件表达式，对不同的 op 和 func 的组合对 44 个 wire 型变量进行赋值，当前正在执行的指令的 op_*的值为 1，其余的为 0。

```
assign RegDst   =   (op_SLL||op_ADDU||op_BEQ||op_OR||
op_SLT||op_AND||op_BGEZ||op_JALR||op_MOVN||op_MOVZ||op_NOR||
op_SLLV||op_SLTU||op_SRA||op_SRAV||op_SRL||op_SRLV||op_SUBU||
op_XOR)?1:0;
```

RegDst 信号用于控制通用寄存器堆 Reg_file 的写头地址

```
assign Branch   =   (op_BNE||op_BEQ||op_BGEZ||op_BLEZ||
op_BLTZ)?1:0;
```

Branch 用于控制对 PC 的 Branch 型指令，1 为按延迟槽跳转，0 为加 4

```
assign MemRead  =   (op_LW||op_LB||op_LBU||op_LH||
op_LHU||op_LWL||op_LWR)?1:0;
```

MemRead 信号用来控制从内存读取数据，1 读取，0 不读取

```
assign MemToReg =   (op_LW||op_LB||op_LBU||op_LH||
op_LHU||op_LWL||op_LWR)?1:0;
```

MemRead 信号控制写入 Reg_file 的数据是否来自于内存，1 为来自内存，0 为来自其他

```
assign ALUOp    =   (op_ADDIU || op_LW || op_SW||
op_ADDU||op_LUI||op_LB||op_LBU||op_LH||op_LHU||op_MOVN||
op_MOVZ||op_SB||op_SH||op_SWL||op_SWR||op_LWL||op_LWR)?3'b010:
                    (op_BNE||op_BEQ||op_SUBU)?3'b110:
                    (op_SRA||op_SRAV)?3'b100:
                    //(op_SLL||op_SLLV)?3'b100:
                    (op_OR||op_NOR||op_ORI)?3'b001:
                    (op_SLT||op_SLTI||op_BGEZ||
op_BLEZ||op_BLTZ)?3'b111:
                    (op_SLTIU||op_SLTU)?3'b101:
                    (op_XOR||op_XORI)?3'b011:
                    3'b000;
```

ALUOp 信号为 ALU 运算的控制信号，通过改变该信号控制 ALU 做相应的运算。010 为加法，110 为减法，100 为算术右移，001 为按位或，000 为按位与，011 为按位异或，101 为无符号数比较，111 为有符号数比较

```
assign MemWrite =   (op_SW||op_SB||op_SH||op_SWL||
op_SWR)?1:0;
```

MemWrite 为是否向内存写入数据信号，1 为写入，0 为不写入

```
assign ALUSrc    =   (op_ADDIU ||op_LW ||op_SW ||
op_SLL||op_LUI||op_SLTI||op_SLTIU||op_ANDI||op_LB||op_LBU||
op_LH||op_LHU||op_LWL||op_LWR||op_ORI||op_SB||op_SH||op_SWL||
op_SWR||op_SRA||op_XORI)?1:0;
```

ALUSrc 控制进入 ALU 的第二个数据位的信号是什么。1 为来自 Sign-extend 或立即数，0 为来自寄存器堆读取

```
assign RegWrite =   (op_ADDIU ||op_SLL||(op_LW &&
Address[1:0]==2'b00)||op_ADDU||op_JAL||op_LUI||op_OR||op_SLT||
op_SLTI||op_SLTIU||op_AND||op_ANDI||op_JALR||op_LB||op_LBU||
op_LH||op_LHU||op_LWL||op_LWR||(op_MOVN&&(!Zero))||
(op_MOVZ&&Zero)||op_NOR||op_ORI||op_SLLV||op_SLTU||op_SRA||
op_SRAV||op_SRL||op_SRLV||op_SUBU||op_XOR||op_XORI)?1:0;
```

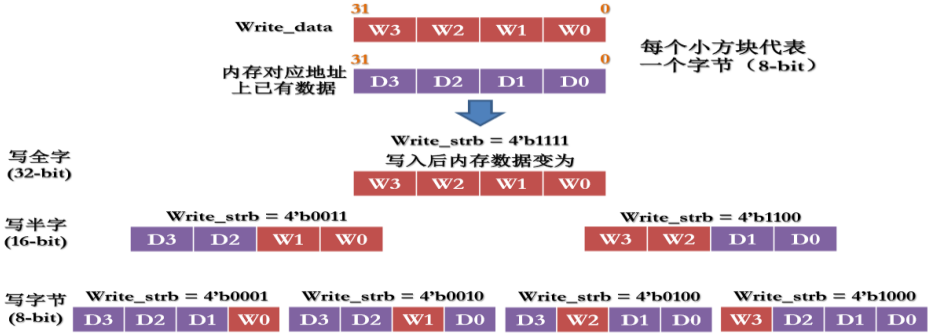
RegWrite 为控制是否将数据写入寄存器堆，1 为写入，0 为不写入

```
assign Write_strb = (op_ADDIU ||op_BNE||op_LW||op_SW||
op_SLL||op_ADDU||op_BEQ||op_J||op_JAL||op_JR||
        op_LUI||op_OR||op_SLT||op_SLTI||op_SLTIU||op_AND||
op_ANDI||op_BGEZ||op_BLEZ||op_BLTZ||op_JALR||op_LB||op_LBU||
op_LH||op_LHU||op_LWL||op_LWR||op_MOVN||op_MOVZ||op_NOR||
op_ORI||op_SLLV||op_SLTU||op_SRA||op_SRAV||op_SRL||op_SRLV||
op_SUBU||op_XOR)?4'b1111
        :(op_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
        :(op_SH)?
                        (Result[1]==1'b0)?4'b0011:
                        (Result[1]==1'b1)?4'b1100:0
        :(op_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

        :(op_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

        :0;
```

Write_strb 为表示向内存写入数据的位数。如果 Write_strb[i]==1,那么把数据的[8*i+7，8*i]位写入到内存对应位置。如下图表示：



2) PC 段

```
always@(posedge clk)
begin
if(rst)
PC<=32'b0;
else PC<=PC2;
end
assign PC3 = PC + 32'd4;
assign PCJUMP = (op_JR||op_JALR)?data1:{PC3[31:28],Instruction[25:0],2'b00};

alu alu2(PC3,(Sign_extend <<2),3'b010,Overflow1,CarryOut1,Zero1,PC1);

assign PC2 = (op_J||op_JAL||op_JR||op_JALR)?PCJUMP:((Branch&&(!Zero)&&(!RegDst))||(Branch&&Zero&&RegDst))?PC1:PC3;
```

PC 的更新使用上升沿触发，将 PC2 的值更新至 PC。PC2 的值的来源有三个：PC1,PC3 和 PCJUMP。PC3 的值为上一拍 PC 的值加 4，PC1 的值为满足 branch 指令的跳转值，PCjump 为满足 jump 类型指令的跳转值。

3) reg_file 段

```
reg_file reg_file1(clk,rst,RF_waddr,rs,rt,RF_wen,RF_wdata,data1,data2);

assign RF_waddr = (op_JAL)?5'd31:(RegDst)?rd:rt;

assign RF_wen = RegWrite;

assign RF_wdata =
(op_SRL)?(data2>>sa)
:(op_SRLV)?(data2>>{data1[4:0]})
:(op_SLL)?(data2<<sa)
:(op_SLLV)?(data2<<{data1[4:0]})
:(op_NOR)?~Result
:(op_MOVN||op_MOVZ)?data1
:(op_LB||op_LBU||op_LH||op_LHU||op_LWL||op_LWR)?Rf_wdata
:(op_JAL||op_JALR)?PC+32'd8
:(op_LUI)?{Instruction[15:0],16'd0}
:(MemToReg)?Read_data
:Result;

assign Rf_wdata = (op_LB)?
                  (Result[1:0]==2'b00)?(Read_data[7]==1)?{24'hffffff,Read_data[7:0]}:{24'd0,Read_data[7:0]}:
                  (Result[1:0]==2'b01)?(Read_data[15]==1)?{24'hffffff,Read_data[15:8]}:{24'd0,Read_data[15:8]}:
                  (Result[1:0]==2'b10)?(Read_data[23]==1)?{24'hffffff,Read_data[23:16]}:{24'd0,Read_data[23:16]}:
                  (Result[1:0]==2'b11)?(Read_data[31]==1)?{24'hffffff,Read_data[31:24]}:{24'd0,Read_data[31:24]}:0
                  :(op_LBU)?
                  (Result[1:0]==2'b00)?$unsigned(Read_data[7:0]):
                  (Result[1:0]==2'b01)?$unsigned(Read_data[15:8]):
                  (Result[1:0]==2'b10)?$unsigned(Read_data[23:16]):
                  (Result[1:0]==2'b11)?$unsigned(Read_data[31:24]):0
                  :(op_LH)?
                  (Result[1]==1'b0)?(Read_data[15]==1)?{16'hffff,Read_data[15:0]}:{16'd0,Read_data[15:0]}:
                  (Result[1]==1'b1)?(Read_data[31]==1)?{16'hffff,Read_data[31:16]}:{16'd0,Read_data[31:16]}:0   //C
                  :(op_LHU)?
                  (Result[1]==1'b0)?$unsigned(Read_data[15:0]):
                  (Result[1]==1'b1)?$unsigned(Read_data[31:16]):0
                  :(op_LWL)?
                  (Result[1:0]==2'b00)?{Read_data[7:0],data2[23:0]}:            //CONCERNNED!
                  (Result[1:0]==2'b01)?{Read_data[15:0],data2[15:0]}:
                  (Result[1:0]==2'b10)?{Read_data[23:0],data2[7:0]}:
                  (Result[1:0]==2'b11)?{Read_data[31:0]}:0
                  :(op_LWR)?
                  (Result[1:0]==2'b00)?{Read_data[31:0]}:
                  (Result[1:0]==2'b01)?{data2[31:24],Read_data[31:8]}:
                  (Result[1:0]==2'b10)?{data2[31:16],Read_data[31:16]}:
                  (Result[1:0]==2'b11)?{data2[31:8],Read_data[31:24]}:0

                  :0;
```

读头始终为 rs，rt 段。RF_waddr 为写头地址，RF_wdata 为写入数据，RF_wen 为写使能信号。

4） alu 段

```
assign Sign_extend = (op_ANDI||op_ORI||op_XORI)?{16'd0,Instruction[15:0]}:(op_SRA)?{26'd0,sa}:
                     (Instruction[15]==1)?{16'hffff,Instruction[15:0]}:{16'd0,Instruction[15:0]};

assign Mux_ALU = //(op_SRAV)?data1[4:0]:(op_BLEZ||op_BGEZ)?32'd1:(op_MOVN||op_MOVZ)?32'd0:(ALUSrc)?Sign_extend:data2;
                 (op_SRAV)?data1[4:0]:(op_BLEZ)?32'd1:(op_BGEZ||op_MOVN||op_MOVZ)?32'd0:(ALUSrc)?Sign_extend:data2;

assign data = (op_SLL||op_MOVN||op_MOVZ||op_SRAV||op_SRA)?data2:data1;

alu alu1(data,Mux_ALU,ALUOp,Overflow,CarryOut,Zero,Result);
```

data 为 ALU 的第一个输入数据，Mux_ALU 信号为 ALU 的第二个输入数据。

5） 内存段

```
assign Address = (op_LB||op_LBU||op_LH||op_LHU||op_LWL||op_LWR||op_SB||op_SH||op_SWL||op_SWR)?{Result[31:2],2'b00}:Result;

assign Write_data =
(op_SB)?
(Write_strb == 4'b0001)?{24'd0,data2[7:0]}:
(Write_strb == 4'b0010)?{16'd0,data2[7:0],8'd0}:
(Write_strb == 4'b0100)?{8'd0,data2[7:0],16'd0}:
(Write_strb == 4'b1000)?{data2[7:0],24'd0}:0
:(op_SH)?
(Write_strb == 4'b0011)?{16'd0,data2[15:0]}:
(Write_strb == 4'b1100)?{data2[15:0],16'd0}:0
:(op_SWL)?
(Write_strb == 4'b0001)?{24'd0,data2[31:24]}:
(Write_strb == 4'b0011)?{16'd0,data2[31:16]}:
(Write_strb == 4'b0111)?{8'd0,data2[31:8]}:
(Write_strb == 4'b1111)?{data2[31:0]}:0
:(op_SWR)?
(Write_strb == 4'b1111)?{data2[31:0]}:
(Write_strb == 4'b1110)?{data2[23:0],8'd0}:
(Write_strb == 4'b1100)?{data2[15:0],8'b0}:
(Write_strb == 4'b1000)?{data2[7:0],24'b0}:0
:data2;
```

Address 是内存写头地址，Write_data 是写入内存的数据。

6） ALU 运算模块核心代码

```
assign registerAND = A&B;

assign registerOR = A|B;

assign registerXOR = A^B;

assign BvertSIGNED = ~{B[31],B}+33'd1;
assign BvertUNSIGNED = ~{1'b0,B}+33'd1;
assign BnumberSIGNED = (ALUop==3'b010)?{B[31],B}:BvertSIGNED;
assign BnumberUNSIGNED = (ALUop==3'b010)?{1'b0,B}:BvertUNSIGNED;

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

assign calculate = {A[31],A}+BnumberSIGNED;
assign calculate1 = {1'b0,A}+BnumberUNSIGNED;
assign Overflow = calculate[32]^calculate[31];
assign CarryOut = calculate1[32];



assign Result = (ALUop==3'b011)?registerXOR:(ALUop==3'b000)? registerAND:(ALUop==3'b001)?
registerOR:(ALUop==3'b010)?calculate[31:0]:(ALUop==3'b110)?calculate[31:0]:(ALUop==3'b111)?
calculate[31]^Overflow:(ALUop == 3'b100)?registerRIGHT:(ALUop == 3'b101)?CarryOut:0;

assign Zero = !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、本地上板及云平台调试过程中的难点等）

最主要的问题是对文档 pdf 的理解问题。参考文档给出的信息太过简略，比如说在程序设计中的三个 RF 信号也没有明确的说明，为设计 cpu 带来了太多的不必要的时间浪费。

在花费大量时间对实验整体进行了一个全面的理解之后，basic 组进行的较为顺利，没有花费太久就通过了测试。medium 也较为快速的通过。但是进行到 advanced 段的时候，发现大量与之前设计构型不兼容的指令形式，这给代码整体的设计带来了麻烦。这也是第一次实现这种大型的程序设计，暴露出了一些书写代码的风格带来不便的问题。所幸有 testbench 协助查找问题，还是比较理想的解决了这些问题。

三、 对讲义中思考题（如有）的理解和回答

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）

建议将该实验在时间线上略微后调，基本处于主线课程讲完某一部分后的一周到两周再开始对该部分的程序设计，留出缓冲和理解的时间。资料总体上较为丰富，想要查询的部分都有，就是总量过于巨大不太方便查找。另外，MIPS 指令集 Vol.2 中 Operation 叙述有问题，有时不符合逻辑。比如对于 LB 的叙述就容易使人产生迷惑。