P5 实验设计报告

18373085 张海渝

一. 数据通路

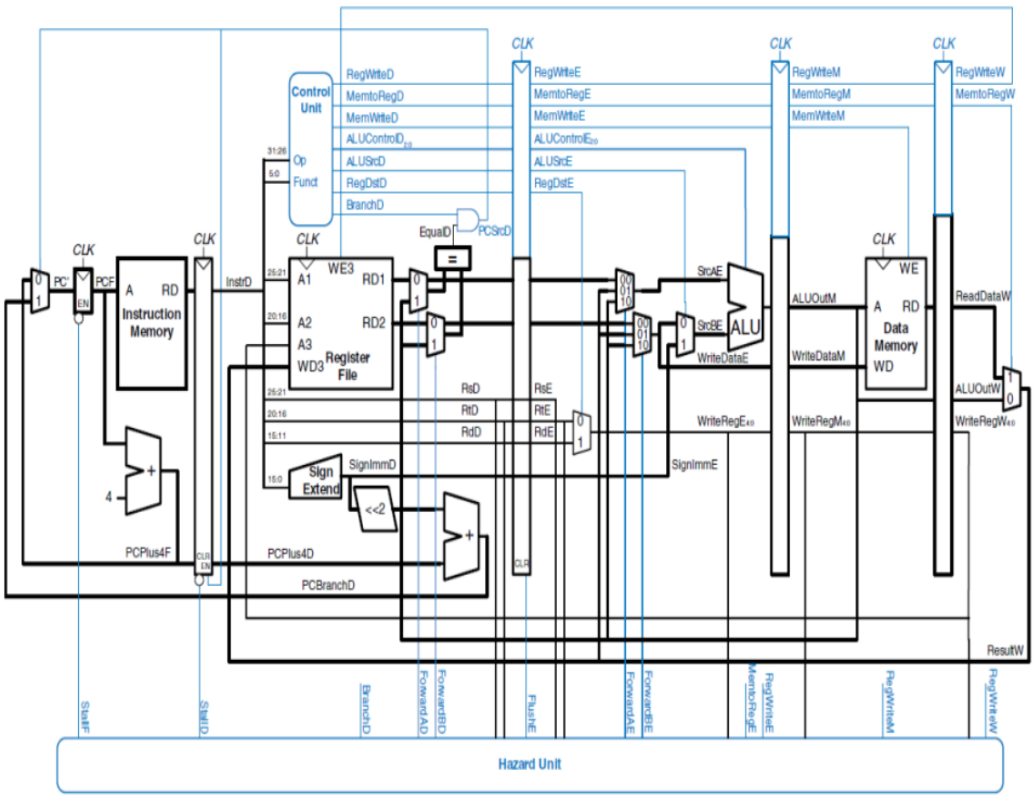

图 1：数据通路（以此图为基础，之后会说明修改的地方）

二. 模块规格

1. GetNpc：（程序计数器）

序号	功能名称	方向	功能描述
1	npc[31:0]	I	为下一 PC 值
2	reset	I	复位信号
3	clk	I	时钟信号
4	En	I	使能信号
5	PC[31:0]	O	当前 D 寄存器之前的 PC 值

模块解释：

1. 如果 Reset 信号有效时，用选择器将 PC 值复位。
2. 解释 En 信号：当需要暂停时，即 D 寄存器需要暂停的时候，需要对该模块不能向下传值，即保持 D 寄存器之前的 PC 值不被修改。

2. IM：（指令获取）

序号	功能名称	方向	功能描述
1	add[11:0]	I	指令地址
2	Instr	O	输出的机器码

模块解释：

1. add[11:0]是当前 PC 值的第 13-2 位(因为 ROM 地址为 12 位，并且 PC 每次加 4，相当于 ROM 加 1)。
2. 用$readmemh 载入机器码。
3. 用寄存器储存机器码。

```
module IM(
    input [11:0] add,
    output [31:0] Instr
     );
    reg [31:0] ROM [4095:0];
    initial  begin
       $readmemh("code.txt",ROM);
    end
    assign Instr=ROM[add];
endmodule
```

图 2：IM 模块代码图

3. GRF：（通用寄存器组，也称寄存器文件、寄存器堆）

序号	功能名称	方向	功能描述
1	pc[31]	I	当前 PC 值
2	A1[4:0]	I	机器码的第 25-21 位，寄存器编号
3	A2[4:0]	I	机器码的第 20-16 位，寄存器编号
4	A3[4:0]	I	需要改变的寄存器
5	WD3[31:0]	I	寄存器需要改变为的值
6	reset	I	同步复位信号
7	clk	I	时钟信号

8	WE	I	是否写入寄存器信号
9	RD1[31:0]	O	A1 寄存器中的值
10	RD2[31:0]	O	A2 寄存器中的值

模块解释：

1.0 号寄存器始终为 0，不改变。

2.一共有 32 个带有使能端的寄存器。

3.需要注意：首先这一模块是在 D/E 级，其次 PC 值并不是 D 级的 PC 值而是通过 W 级传输过来的值，WD3 以及 WE 都是通过 W 级传输过来的，还需要注意 RD1,RD2 并不是准确值，即并不是准确的寄存器里面的值，后面会通过转发暂停机制实现准确。

```
reg [31:0] a [31:0];
integer i;
initial begin
   for(i=0;i<=31;i=i+1) begin
      a[i]=0;
   end
end

assign RD1=a[A1];
assign RD2=a[A2];

always@(posedge clk) begin
   if(reset==1) begin
      for(i=0;i<=31;i=i+1) a[i]<=0;
   end
   else begin
      if(WE==1) begin
         if(A3!=0) begin
            $display("%d@%h: $%d <= %h", $time, pc, A3,WD3);
            a[A3]<=WD3;
         end
      end
   end
end
```

图 3：GRF 模块部分代码

4.ALU：（算数逻辑单元）

序号	功能名称	方向	功能描述
1	A[31:0]	I	进行操作的第一个数
2	B[31:0]	I	进行操作的第二个数
3	ALUctr[3:0]	I	进行操作的方式

4	ALUresult[31:0]	O	计算后的结果

模块解释：

1. ALUctr=0：输出 0 值

   ALUctr=1：加法

   ALUctr=2：减法

   ALUctr=3：或（or）操作

   ALUctr=4:输出 B 的值

   ALUctr=5：输出 A&B

   ALUctr=6：输出 A^B

   ALUctr=7：输出~（A|B）

   ALUctr=8：输出（$signed(A)<$signed(B)）

   ALUctr=9: 输出(A)<(B)

   ALUctr=其他：输出 0

2. 此模块存在于 E/M 级，需要注意其两个操作数的正确与否，需要通过转发机制实现，会在后面详细说明。

```
always@(*) begin
    if(ALUctr==0) begin
        ALUresult<=0;
    end
    else if(ALUctr==1) ALUresult<=A+B;
    else if(ALUctr==2) ALUresult<=A-B;
    else if(ALUctr==3) ALUresult<=A|B;
    else if(ALUctr==4) ALUresult<=B;
    else if(ALUctr==5) ALUresult<=A&B;
    else if(ALUctr==6) ALUresult<=A^B;
    else if(ALUctr==7) begin
        ALUresult<=~(A|B);
    end
    else if(ALUctr==8) begin
        if($signed(A)<$signed(B)) ALUresult<=1;
        else ALUresult<=0;
    end
    else if(ALUctr==9) begin
        if((A)<(B)) ALUresult<=1;
        else ALUresult<=0;
    end
    else ALUresult<=0;
end
```

图 4：ALU 模块代码图

4. DM：（数据存储器）

序号	功能名称	方向	功能描述
1	addr[11:0]	I	进行操作的数据的地址

2	opcode[5:0]	I	当前执行的指令
3	byteM[1:0]	I	存储的时候选择怎么存储
4	din[31:0]	I	数据
6	clk	I	时钟信号
7	dout[31:0]	O	读出的数据
8	pc	I	当前 PC 值

模块解释：

1.此模块通过一组寄存器，此模块在 M 级。

2.因为模块容量为 4kb，所以进行操作的地址为 M/W 级 ALUoutM 计算结果的 11-2 位。且 PC 值为 M 级的 PC 值，会在前面进行传输下来。

3.用 ans 代表当前 addr 的 32 位表达值。

```
reg [31:0] a [4095:0];
wire [31:0] ans;
initial begin
   for(i=0;i<=4095;i=i+1) a[i]=0;
end
assign dout=a[addr];
assign ans={18'b0,addr[11:0],2'b0};
always@(posedge clk) begin
   if(reset==1) begin
      for(i=0;i<=4095;i=i+1) a[i]<=0;
   end
   else begin
      if(MemWrite==1&&opcode==43) begin
         $display("%d@%h: *%h <= %h",$time,pc, ans,din);
         a[addr]<=din;
      end
      else if(MemWrite==1&&opcode==40) begin //sb
         if(byteM==0) a[addr][7:0]=din[7:0];
         else if(byteM==1) a[addr][15:8]=din[7:0];
         else if(byteM==2) a[addr][23:16]=din[7:0];
         else if(byteM==3) a[addr][31:24]=din[7:0];
         $display("%d@%h: *%h <= %h",$time,pc, ans,a[addr]);
      end
      else if(MemWrite==1&&opcode==41) begin //sh
         if(byteM==0) a[addr][15:0]=din[15:0];
         else if(byteM==2) a[addr][31:16]=din[15:0];
         $display("%d@%h: *%h <= %h",$time,pc, ans,a[addr]);
      end
   end
end
```

图 5：DM 模块代码图

6.EXT：（数据扩展器）

序号	功能名称	方向	功能描述

1	IMM[15:0]	I	进行扩展的数
2	ExtOp[1:0]	I	扩展的方式
3	ExtIMM[31:0]	O	扩展后的数据

模块解释：

1. ExtOp=0：前 16 位 0 扩展

   ExtOp=1：符号扩展至 32 位

   ExtOp=2：后 16 位补 0 扩展
2. 用 ans（一个寄存器）以及 for 循环的利用实现。
3. 此模块存在于 D/E 级，会在 D/E 级

```
    reg [31:0] ans;
    integer i;
    initial  begin
    ans=0;
    end
    assign ExtIMM=ans;
    always@(*) begin
        if(ExtOp==0) begin
            for(i=31;i>=0;i=i-1) begin
                if(i>=16) begin
                    ans[i]<=0;
                end
                else ans[i]<=IMM[i];
            end
        end
        else if(ExtOp==1) begin
            for(i=31;i>=0;i=i-1) begin
                if(i>=16) begin
                    ans[i]<=IMM[15];
                end
                else ans[i]<=IMM[i];
            end
        end
        else if(ExtOp==2) begin
            for(i=31;i>=0;i=i-1) begin
                if(i>=16) begin
                    ans[i]<=IMM[i-16];
                end
                else ans[i]<=0;
            end
        end
    end


endmodule
```

图 6：EXT 模块代码图

7. Control：（控制器）

序号	功能名称	方向	功能描述
1	opcode[5:0]	I	Special
2	funct[5:0]	I	Function
3	RegDst[1:0]	O	选择需要在 GRF 中写入数据的寄存器
4	ALUSrc[1:0]	O	选择进行 ALU 操作的第二个操作数
5	MemtoReg[2:0]	O	选择存入 GRF 中寄存器的数据
6	RegWrite	O	是否在 GRF 中写入控制信号
7	MemWrite	O	DM 中的 RAM 使能端控制信号
8	nPC_sel[1:0]	O	选择下一 PC 值信号
9	ExtOp[1:0]	O	数据扩展方式
10	ALUctr[3:0]	O	ALU 进行操作的方式

模块解释：

1. 首先通过 opcode 与 funct 选择出此时进行操作的指令方式。

2.通过 if_else 语句实现控制信号的选择。

3. RegDst==2 时选择 31 号寄存器，MemtoReg==2 是选择 PC+8(因为延迟槽)，为了实现 jal 指令。MemtoReg==3 是为了 Load 型指令

4. npc_sel==0: 其他指令

npc_sel==1: beq bne blez bgtz bltz bgez

npc_sel==2: jal or j

npc_sel==3: jr jalr

5.此模块在 D/E 级操作，后面会将这些产生的控制信号，在后面需要的地方进行传输，保证正确性。

funct	100000	100010					
opcode	000000	000000	001101	100011	101011	000100	001111
	add	sub	ori	lw	sw	beq	lui
RegDst[1:0]	1	1	0	0	x	x	0
ALUScr	0	0	1	1	1	0	1
Memto [1:0]	0	0	0	1	x	x	0

RegWrite	1	1	1	1	0	0	1
MemWrite	0	0	0	0	1	0	0
nPC_sel[1:0]	0	0	0	0	0	1	0
ExtOp[1:0]	x	x	0	1	1	x	2
ALUctr[2:0]	1	2	3	1	1	2	4

图七：信号输出真值表（Memto 代表 MemtoReg）

funct	**001000**	**000000**		
opcode	000000	000000	000011	000010
	jr	nop	jal	j
RegDst[1:0]	0	0	2	0
ALUScr	0	0	0	0
Memto [1:0]	0	0	2	0
RegWrite	0	0	1	0
MemWrite	0	0	0	0
nPC_sel[1:0]	3	0	2	2
ExtOp[1:0]	0	0	0	0
ALUctr[2:0]	0	0	0	0
npc_sel3	1	0	1	0
npc_sel4	1	0	0	0

图七（续）：信号输出真值表（Memto 代表 MemtoReg，部分，其余的同理）

8. ALU_2(HI,LO 乘除模块)

序号	功能名称	方向	功能描述
1	clk	I	时钟信号
2	reset	I	复位信号
3	start	I	是否在开始工作
4	A[31:0]	I	乘除的第一个数
5	B[31:0]	I	乘除的第二个数

6	sel[3:0]	I	选择此模块干什么
7	out[31:0]	O	输出信号
8	busy	O	是否在工作的信号

模块解释：

1.当其中的信号 start 为 1 的时候开始工作，工作方式如下：

sel==1:进行有符号乘操作（mult），置其中的 Busy=6
sel==2:进行无符号乘操作（multu），置其中的 Busy=6
sel==3:进行有符号除操作（div）,置其中的 Busy=11
sel==4:进行无符号除操作（divu），置其中的 Busy=11
sel==5:取出 HI 寄存器中的值（mfhi）
sel==6:取出 LO 寄存器中的值（mflo）
sel==7:给 HI 赋值（mthi）
sel==8:给 LO 赋值（mtlo）

2. 其输出的 busy 信号是为了暂停 D 级中的 mfhi，mflo，mtlo，mthi 着一些指令，因为乘除需要分别进行 5，10 个周期，所以需要等待。

```
assign busy=(Busy>=1)?1:
            0;
assign out=(sel==5)?HI:
           (sel==6)?LO:
            0;
always @(*) begin
   if(start==1) begin
      if(sel==1) begin  {TEMP_HI,TEMP_LO}=$signed($signed({{32{A[31]}},A[31:0]}))*$signed({{32{B[31]}},B[31:0]})); Busy=6;end
      else if(sel==2) begin   {TEMP_HI,TEMP_LO}=A*B; Busy=6;end
      else if(sel==3) begin    TEMP_LO=$signed(A)/$signed(B); TEMP_HI=$signed(A)%$signed(B); Busy=11; end
      else if(sel==4) begin   TEMP_LO=A/B; TEMP_HI=A%B; Busy=11; end
   end
   if(sel==7) begin
      HI<=A;
   end
   if(sel==8) begin
      LO<=A;
   end
end

always@(posedge clk) begin
   if(reset==1) begin
      HI<=0;
      LO<=0;
   end
   else begin
      if(Busy) begin Busy=Busy-1;
      if(Busy==0) begin
         HI<=TEMP_HI;
         LO<=TEMP_LO;
      end
      end
   end
end
```

图八：ALU_2 模块部分代码图

三． 流水线 CPU 各级传输及操作

1.D 级

1. 传输信号：InstrD（来自于 Instr@F）,pc4D（pc4@F）,pcD(pc@F)。

2. 使能端 En,当需要暂停的时候需要将 D 级寄存器锁住，不再传输，当 reset 的时候也需要将其锁住，以免使第一条指令工作多次。

3. 此级进行 GRF 读取操作以及 EXT 扩展操作和 Control 的操作。

4. 还需要进行 B/J 类型指令跳转的选择，通过 npc_sel 进行选择，在上

面已经解释了 npc_sel 的含义。

5.在进行 b 型,j 型指令的时候在这一级会进行一次比较,从而选择 npc_sel 的终值。

6.以后只用传输 Control 翻译出来的信号即可。(集中式译码)

2.E 级

1.传输信号：A1E(A1D@D),A2E(A2D@D),A3E(A3D@D), Data1E(Data1D@D),Data2E(Data2D@D),EXTIMME(EXTIMMD@D),RegDstE(RegDstD@D),ALUSrcE(ALUSrcD@D),MemtoRegE(MemtoRegD@D),RegWriteE(RegWriteD@D),MemWriteE(MemWriteD@D),ALUctrE(ALUctrD@D),opcodeE(opcodeD@D),functE(functD@D),pcE(pcD@D)。

2. 信号解释，其中 A1E,A2E,A3E 代表进行操作的寄存器编号，Data1E,Data2E 分别为 A1E,A2E 为转发以后的值（不一定为当前的准确值，因为可能在 D 级时这两个数据未被使用，于是可能在 E 级的转发器中更新）。其余的为 ControlD 控制信号传输。

3. 此级进行操作的有 ALU，ALU_2 模块。

4. 需要注意在此级就选择出来了会进行回写的寄存器编号，命名为 WriteRegE。还需要注意在此级仍需转发，以免进行操作的数据为错误数据。

3.M 级

1. 传输信号：WriteDataM(WriteDataE@E),WriteRegM(WriteRegE@E),MemtoRegM(MemtoRegE@E),RegWriteM(RegWriteE@E),MemWriteM(MemWriteE@E),opcodeM(opcodeE@E),functM(functE@E),pcM(pcE@E),ALUoutM(ALUoutE@E)。

2. 信号解释：WriteDataM 为向 DM 中写入的值，ALUoutM 为在 E 级中 ALU 中计算出来的值。其余的为控制信号传输。

3. 此级需要进行操作的是 DM 模块。

4.W 级

1.传输信号：ALUoutW(ALUoutM@M),DMoutW(DMoutM@M),pcW(pcM@M),MemtoRegW(MemtoRegM@M),RegWriteW(RegWriteM@M),opcodeW(opcodeM@M),functW

(functM@M),WriteRegW(WriteRegM@M)。

2. 其中需要选择出回写的值，记即在 ALUoutW ,DMoutW, pcW+8 中进行选择出最后的值。(因为延迟槽所以需要 pcW+8)。选择出来值为 ResultW。

四. 转发与暂停机制的实现

1.转发机制：

1.实现的方法：暴力转发。即在每一级中找到需要的更新转发的值，找到在后面需要转发过来的值，用一个选择器进行选择，其中转发需要的条件是满足寄存器编号一样且不为 0，并且此时得到了该周期的值。

2.需要转发的地方：D 级的 GRF 出来的 RD1,RD2 值，这是因为这是读出了其中两个寄存器的值，当进行 beq 的值的时候需要在 D 级就进行比较，所以需要最新的值。RD1 转发以后的值记为 Data1D,RD2 转发以后的值记为 Data2D，按照数据通路可知转发来自于本身的值（RD1），E 级的 EXIMME，offoutE，M 级的 ALUoutM 以及 W 级的 ResultW。用 ForwardAD，ForwardBD 代表选择信号。E 级的 Data1E,Data2E 需要转发，转发至本身的值（Data1E），M 级的 ALUoutM 以及 W 级的 ResultW，用 ForwardAE,ForwarBE 表示选择信号。

```
module Forward_unit_1(opcodeE,functE,WriteRegE,RegWriteE,WriteRegM,RegWriteM,WriteRegW,RegWriteW,A,Forward);
    input [4:0] WriteRegM;
    input RegWriteM;
    input [4:0] WriteRegW;
    input RegWriteW;
    input [4:0] WriteRegE;
    input RegWriteE;
    input [5:0] opcodeE,functE;
    input [4:0] A;
    output reg [2:0] Forward;

    always@(*) begin
      if(opcodeE==15&&RegWriteE==1&&WriteRegE==A&&A!=0)  Forward<=1;
      else if(opcodeE==0&&(functE==0||functE==4||functE==2||functE==6||functE==7||functE==3)&&RegWriteE==1&&WriteRegE==A&&A!=0) Forward<=4;
      else if(RegWriteM==1&&WriteRegM==A&&A!=0) Forward<=2;
      else if(RegWriteW==1&&WriteRegW==A&&A!=0)Forward<=3;
      else Forward<=0;
    end

endmodule
```

图 7：Forward 选择信号示意图

2.暂停机制：

1.暂停机制是为了解决转发机制解决不了的问题。

2.首先用 Tuse 代表该条指令在进入 D 级以后的第几个周期需要用到寄存器中的值，Tnew 代表在流水线中的 E,M 级中需要几个周期才能得到最新的值（比如 b 型的 Tuse 等于 0，E 级的 load 型等于 2）。

3．当 Tuse<后面的任何一级的 Tnew 时需要暂停，暂停时进行的操作为 E 级注入一个空操作，D 级停止传输，pc 停止传输。

4.需要注意当此时在 E 级的为乘除操作时，D 级为 mfhi，mflo，mtlo，mthi 也需要暂停

	Tuse-A1	Tuse-A2	E-Tnew	M-Tnew
addu	1	1	1	0
subu	1	1	1	0
ori	1	1	1	0
lw	1		2	1
sw	1	1	0	0
beq	0	0		
lui			0	0
jal			2	1
jr	0			

```
assign stall_rs=(A1D==0)?0:
                (A1D==WriteRegE&&RegWriteE==1&&Tuse_rs<Tnew_E)?1:
                (A1D==WriteRegE&&RegWriteE==1&&Tuse_rs>=Tnew_E)?0:
                (A1D==WriteRegM&&RegWriteM==1&&Tuse_rs<Tnew_M)?1:
                0;

assign stall_rt=(A2D==0)?0:
                (A2D==WriteRegE&&RegWriteE==1&&Tuse_rt<Tnew_E)?1:
                (A2D==WriteRegE&&RegWriteE==1&&Tuse_rt>=Tnew_E)?0:
                (A2D==WriteRegM&&RegWriteM==1&&Tuse_rt<Tnew_M)?1:
                0;

assign stall_3=(busyE==1&&((opcodeD==0&&functD==16)||(opcodeD==0&&functD==18)||
                           (opcodeD==0&&functD==17)||(opcodeD==0&&functD==19)));
assign stall=stall_rs|stall_rt|stall_3;

endmodule
```

图九：Tuse，Tnew 代码（部分代码）

五． 测试代码：

1.R 型指令（addu 为例，乘除包含 mflo，mfhi，slt 等类型指令）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-M-RS	subu	E	rs	subu $1,$2,$3 addu $4,$1,$2	转发

R-M-RT	subu	E	rt	subu $1,$2,$3 addu $4,$2,$1	转发

R-W-RS	subu	E	rs	subu $1,$2,$3
				nop
				addu $4,$1,$2
					转发

R-W-RT	subu	E	rt	subu $1,$2,$3
				nop
				addu $4,$2,$1
					转发

I-M-RS	ori	E	rs	ori $1,$0,0x1111 addu $3,$1,$2	转发
I-M-RT	ori	E	rt	ori $1,$0,0x1111 addu $3,$2,$1	转发
I-W-RS	ori	E	rs	ori $1,$0,0x1111 nop addu $3,$1,$2	转发
I-W-RT	ori	E	rt	ori $1,$0,0x1111 nop addu $3,$2,$1	转发

LD-W-RS	lw	E	rs	lw $1,0($2)
				nop
				addu $3,$1,$2
					转发

LD-W-RT	lw	E	rt	lw $1,0($2)
				nop
				addu $3,$2,$1
					转发

LD-M-RS	lw	E	rs	lw $1,0($2) addu $3,$1,$2	暂停
LD-M-RT	lw	E	rt	lw $1,0($2) addu $3,$1,$2	暂停
LD-E-RS	lw	D	rs	lw $1,0($2) addu $3,$1,$2	暂停
LD-E-RT	lw	D	rt	lw $1,0($2) addu $3,$1,$2	暂停
JAL-M-RS	jal	D	rs	jal dfs nop dfs: addu $1,$31,$0	转发
JAL-M-RT	jal	D	rt	jal dfs nop dfs: addu $1,$0,$31	转发
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop	转发

				addu $1,$31,$0	
JAL-W-RT	jal	D	rt	jal dfs nop dfs: nop addu $1,$0,$31	转发

2.I 型指令（ori 为例，其中包含了 sll 等指令的测试）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-M-RS	subu	E	rs	subu $1,$2,$3 ori $4,$1,0x1111	转发
R-W-RS	subu	E	rs	subu $1,$2,$3 nop ori $4,$1,0x1111	转发
I-M-RS	ori	E	rs	ori $1,$0,0x1110 ori $2,$1,0x1111	转发
I-W-RS	ori	E	rs	ori $1,$0,0x1110 nop ori $3,$1,0x1111	转发
LD-W-RS	lw	E	rs	lw $1,0($2) nop ori $3,$1,0x0	转发
LD-M-RS	lw	D	rs	lw $1,0($2) nop ori $3,$1,$2	暂停
LD-E-RS	lw	D	rs	lw $1,0($2) ori $3,$1,$2	暂停
JAL-M-RS	jal	D	rs	jal dfs nop dfs: ori $1,$31,0x0	转发
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop ori $1,$31,0x0	转发

3.LD 型指令（lw 为例）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-M-RS	subu	E	rs	subu $1,$2,$3 lw $4,0($1)	转发

R-W-RS	subu	E	rs	subu $1,$2,$3
				nop
				lw $4,0($1)
					转发

I-M-RS	ori	E	rs	ori $1,$0,0x1110 lw $4,0($1)	转发
I-W-RS	ori	E	rs	ori $1,$0,0x1110 nop lw $4,0($1)	转发

LD-W-RS	lw	E	rs	lw $1,0($2)
				nop
				lw $4,0($1)
					转发

LD-M-RS	lw	D	rs	lw $1,0($2)
				nop
				lw $4,0($1)
					转发

LD-E-RS	lw	D	rs	lw $1,0($2) lw $4,0($1)	暂停
JAL-M-RS	jal	D	rs	jal dfs nop dfs: lw $4,0($31)	转发
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop lw $4,0($31)	转发

4.Store 型指令（sw 为例）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-M-RS	subu	E	rs	subu $1,$2,$3 sw $4,0($1)	转发
R-M-RT	subu	E	rt	subu $1,$2,$3 sw $1,0($0)	转发

R-W-RS	subu	E	rs	subu $1,$2,$3
				nop
				sw $4,0($1)
					转发

R-W-RS	subu	E	rt	subu $1,$2,$3
				nop
				sw $1,0($0)
					转发

I-M-RS	ori	E	rs	ori $1,$0,0x1110 sw $4,0($1)	转发
I-W-RS	ori	E	rs	ori $1,$0,0x1110 nop sw $4,0($1)	转发
I-M-RT	ori	E	rt	ori $1,$0,0x1110 sw $1,0($0)	转发
I-W-RT	ori	E	rt	ori $1,$0,0x1110 nop	转发

				sw $1,0($0)	

LD-W-RS	lw	E	rs	lw $1,0($2)
				nop
				sw $4,0($1)
					转发

LD-W-RT	lw	E	rt	lw $1,0($2)
				nop
				sw $1,0($0)
					转发

LD-M-RS	lw	D	rs	lw $1,0($2)
				nop
				sw $4,0($1)
					转发

LD-M-RT	lw	D	rt	lw $1,0($2)
				nop
				sw $1,0($0)
					转发

LD-E-RS	lw	D	rs	lw $1,0($2) sw $4,0($1)	暂停
LD-E-RT	lw	D	rt	lw $1,0($2) sw $1,0($0)	转发
JAL-M-RS	jal	D	rs	jal dfs nop dfs: sw $4,0($31)	转发
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop sw $4,0($31)	转发

5.B 型指令（beq 为例）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-E-RS	subu	D	rs	subu $1,$2,$3 beq $1,$2,loop	暂停
R-E-RT	subu	D	rt	subu $1,$2,$3 beq $2,$1,loop	暂停
R-M-RS	subu	D	rs	subu $1,$2,$3 nop beq $1,$2,loop	转发
R-M-RT	subu	D	rt	subu $1,$2,$3 nop beq $2,$1,loop	转发
I-E-RS	ori	D	rs	ori $1,$2,0x1111 beq $1,$2,loop	暂停
I-E-RT	ori	D	rt	ori $1,$2,0x1111 beq $2,$1,loop	暂停

I-M-RS	ori	D	rs	ori $1,$2,0x1111 nop beq $1,$2,loop	转发
I-M-RT	ori	D	rt	ori $1,$2,0x1111 nop beq $2,$1,loop	转发
LD-E-RS	lw	D	rs	lw $1,0($2) beq $1,$2,loop	暂停
LD-E-RT	lw	D	rt	lw $1,0($2) beq $2,$1,loop	暂停
LD-M-RS	lw	D	rs	lw $1,0($2) nop beq $1,$2,loop	暂停
LD-M-RT	lw	D	rt	lw $1,0($2) nop beq $2,$1,loop	暂停
LD-W-RS	lw	D	rs	lw $1,0($2) nop nop beq $1,$2,loop	转发
LD-W-RT	lw	D	rt	lw $1,0($2) nop nop beq $2,$1,loop	转发
JAL-M-RS	jal	D	rs	jal dfs nop dfs: beq $4,$31,loop	暂停
JAL-M-RT	jal	D	rt	jal dfs nop dfs: beq $31,$4,loop	暂停
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop beq $4,$31,loop	转发
JAL-W-RT	jal	D	rt	jal dfs nop dfs: nop beq $31,$4,loop	转发

6.jr 指令：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-E-RS	subu	D	rs	subu $1,$2,$3 jr $1	暂停
R-M-RS	subu	D	rs	subu $1,$2,$3 nop jr $1	转发
I-E-RS	ori	D	rs	ori $1,$2,0x1111 jr $1	暂停
I-M-RS	ori	D	rs	ori $1,$2,0x1111 nop jr $1	转发
LD-E-RS	lw	D	rs	lw $1,0($2) jr $1	暂停
LD-M-RS	lw	D	rs	lw $1,0($2) nop jr $1	暂停
LD-W-RS	lw	D	rs	lw $1,0($2) nop nop jr $1	转发
JAL-M-RS	jal	D	rs	jal dfs nop dfs: jr $31	暂停
JAL-W-RS	jal	D	rs	jal dfs nop dfs: nop jr $31	转发

6.乘除指令（mult 为例）：

测试类型	前序指令	冲突位置	冲突寄存器	测试序列	解决方法
R-E-RS	mfhi	D	HI	mfhi $1 mult $2,$3	暂停

六．思考题：

乘除模块:

1.为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

答：因为乘除模块需要进行的时钟周期为多个周期，而 ALU 模块进行的时钟周期为一个周期，而且单独用一个乘除模块不会影响不用乘除模块的指令，加快 CPU 进程，因为单独的 HI，LO 模块可以更高效的实现 cpu 性能，因为其他的指

令就可以继续进行，而不会受到影响，而且如果放在 GRF 中，没有办法在我的设计中直接改变两个寄存器的值，所以单独是最好的选择。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

答：这是为了不运用乘除模块的指令可以正常进行操作而不受乘除模块的影响，这样可以就像延迟槽一样，可以使得 CPU 的效率增加，并且使 CPU 设计不受到更多的影响

扩展模块：

1. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

答：C 语言中的字符串可以知道，当按字节访问的时候，就像相当于访问了一个 char 字符，此时一个字符可以知道可以直接寻址，而你一个字，可以知道，必须要进行扩展等操作，所以按字节访问更有优势。

复杂性控制与设计风格：

1. 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段?

答：首先在译码阶段，我才用了集中式译码的手段，运用了 if_else,我自认为这种方法对我来说比宏定义更加的让我喜欢，因为这种方式虽然代码量大，但是这种方式我觉得更加易于 debug，因为 opcode，funct 都是具体的值，方便观察。而且控制信号更加容易修改。

在选择器方面，我也选择两种方式，当选择器需要输出多个信号的时候我选择了用模块的方式实现，当只用输出一个信号的时候，我选择的就是用 assign 语句进行选择，我觉得这样可以是我的思路更加清晰，因为用模块进行选择的时候容易打错值。而直接用 assign 可以更加容易修改，但是局限性比较强。

在转发模块方面，我选择了侦测者方式，即用模块的方式。

在暂停方面我又选择了规划者方式，因为就像我刚才所说，因为输出的信号只有一位，需要的信号也不多，所以我选择了这种方式。

总的来说我选择了两种方式的综合

2. 你对流水线 CPU 设计风格有何见解?

答：我认为应该选择一种适合自己的方式，并且能够很好的加入指令，而且再转发等方面应该考虑各种各样的情况，以免在暂停转发方面出现问题，应该多写一些模块，比如各种的 mux，这样在需要的时候能够更加方便简便，最重要的一点就是需要很好的命名格式，要不然会发生惨剧，两个小时一个 bug，所以最重要的我觉得使命名格式，设计风格更命名模式有很大的关系。

在线测试：

1. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的?

答：b,j 指令最复杂，因为这些需要更多的转发，其实就是用了更多的模块，其实冲突最复杂就是用的，我是通过手动一条一条分类，通过器件进行分类，比如通过 ALU 得到最终结果的，其实最后的转发暂停条件就是一样的，所以就进行分类以后进行测试，这样能覆盖所以的冲突。