计算机组成原理 实验报告

姓名：魏钊　学号：PB18111699　实验日期：2020-6-7

一、实验题目：

实验五 流水线 CPU

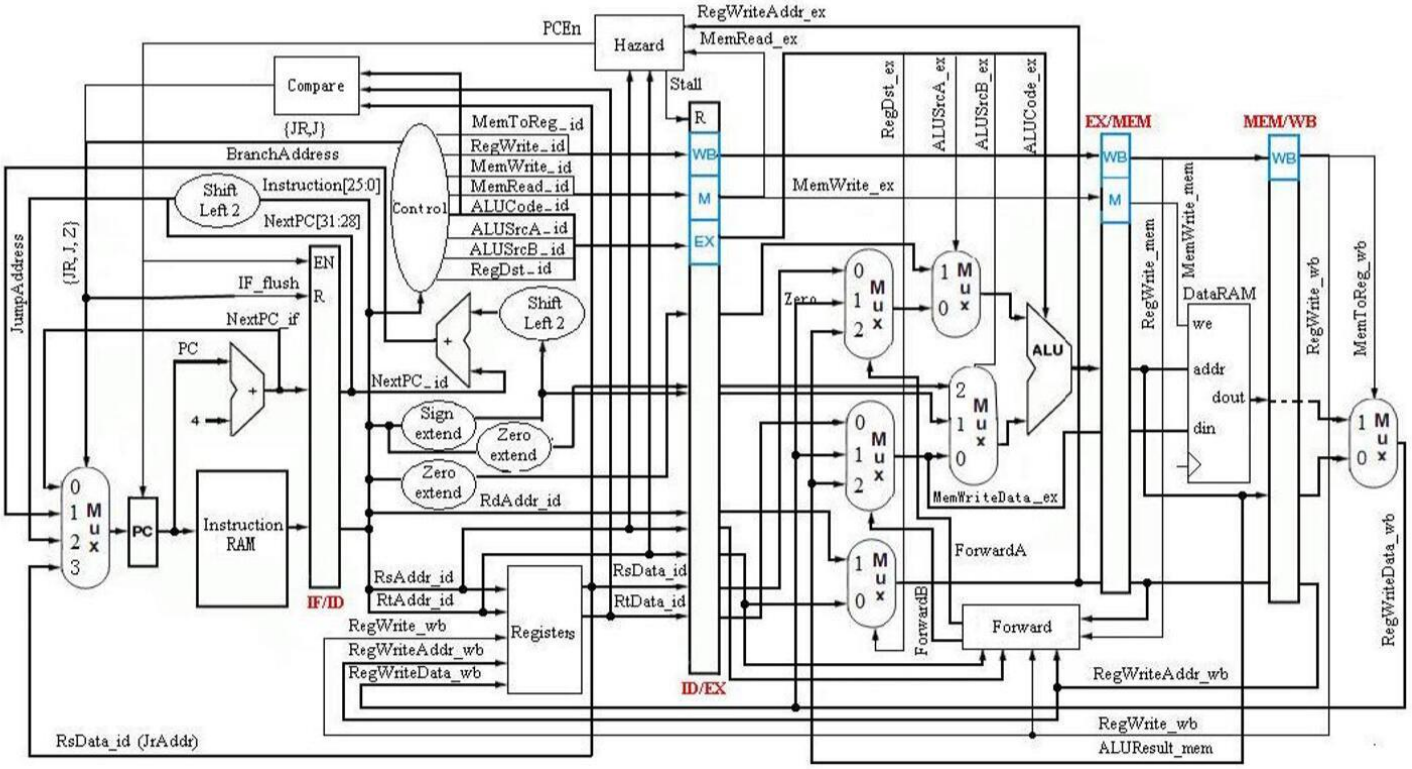
二、实验目的：

1. 理解流水线 CPU 的组成结构和工作原理；
2. 掌握数字系统的设计和调试方法；
3. 熟练掌握数据通路和控制器的设计和描述方法。

三、实验平台：

ISE / Vivado（暂不支持其他 Verilog HDL 开发环境的检查）

四、实验过程：



指令存储器：256 x 32 位 ROM，IP 例化，分布式存储器

数据存储器：256 x 32 位 RAM，IP 例化，分布式存储器

相关信号说明：

IFID.v			
输入/输出	宽度	信号名	说明
input	[0:0]	clk	时钟沿
input	[0:0]	en	使能信号，高电平有效
input	[0:0]	flush	清空信号，高电平有效
input	[31:0]	PCPlus_in	IF 段的 PC+4
input	[31:0]	IMemout_in	IF 段的 Imem 输出
output	[31:0]	PCPlus_out	ID 段的 PC+4
output	[31:0]	IMemout_out	ID 段的 Imem 输出

IDEX.v			
输入/输出	宽度	信号名	说明
input	[0:0]	clk	时钟沿
input	[0:0]	en	使能信号高电平有效
input	[0:0]	flush	清空信号高电平有效
input	[31:0]	PCPlus_in	ID 段的 PCPlus
input	[31:0]	RegRdout1_in	ID 段的 RegRdout1
input	[31:0]	RegRdout2_in	ID 段的 RegRdout2
input	[31:0]	IMMSignExtended_in	ID 段的 IMMSignExtended
input	[31:0]	IMMZeroExtended_in	ID 段的 IMMZeroExtended
input	[31:0]	ShamtZeroExtended_in	ID 段的 ShamtZeroExtended
input	[4:0]	Rs_in	ID 段的 Rs
input	[4:0]	Rt_in	ID 段的 Rt
input	[4:0]	RegWtaddr_in	ID 段的 RegWtaddr
output	[31:0]	PCPlus_out	EX 段的 PCPlus
output	[31:0]	RegRdout1_out	EX 段的 RegRdout1
output	[31:0]	RegRdout2_out	EX 段的 RegRdout2
output	[31:0]	IMMSignExtended_out	EX 段的 IMMSignExtended
output	[31:0]	IMMZeroExtended_out	EX 段的 IMMZeroExtended
output	[31:0]	ShamtZeroExtended_out	EX 段的 ShamtZeroExtended
output	[4:0]	Rs_out	EX 段的 Rs
output	[4:0]	Rt_out	EX 段的 Rt
output	[4:0]	RegWtaddr_out	EX 段的 RegWtaddr
input	[0:0]	RegDst_in	ID 段的 RegDst
input	[0:0]	ALUSrcASel_in	ID 段的 ALUSrcASel
input	[1:0]	ALUSrcBSel_in	ID 段的 ALUSrcBSel
input	[4:0]	ALUControl_in	ID 段的 ALUControl
input	[0:0]	DMemRead_in	ID 段的 DMemRead
input	[0:0]	DMemWrite_in	ID 段的 DMemWrite
input	[0:0]	DMemtoReg_in	ID 段的 DMemtoReg
input	[0:0]	RegWrite_in	ID 段的 RegWrite
output	[0:0]	RegDst_out	EX 段的 RegDst
output	[0:0]	ALUSrcASel_out	EX 段的 ALUSrcASel
output	[1:0]	ALUSrcBSel_out	EX 段的 ALUSrcBSel
output	[4:0]	ALUControl_out	EX 段的 ALUControl
output	[0:0]	DMemRead_out	EX 段的 DMemRead
output	[0:0]	DMemWrite_out	EX 段的 DMemWrite
output	[0:0]	DMemtoReg_out	EX 段的 DMemtoReg
output	[0:0]	RegWrite_out	EX 段的 RegWrite

EXMEM.v			
输入/输出	宽度	信号名	说明
input	[0:0]	clk	时钟沿
input	[0:0]	en	使能信号高电平有效
input	[0:0]	flush	清空信号高电平有效
input	[31:0]	ALUResult_in	EX 段的 ALUResult
input	[31:0]	DMemin_in	EX 的 Dmemin
input	[4:0]	RegWtaddr_in	EX 的 RegWtaddr
output	[31:0]	ALUResult_out	MEM 段的 ALUResult
output	[31:0]	DMemin_out	MEM 的 Dmemin
output	[4:0]	RegWtaddr_out	MEM 的 RegWtaddr
input	[0:0]	DMemRead_in	EX 段的 DMemRead
input	[0:0]	DMemWrite_in	EX 段的 DMemWrite
input	[0:0]	DMemtoReg_in	EX 段的 DMemtoReg
input	[0:0]	RegWrite_in	EX 段的 RegWrite
output	[0:0]	DMemRead_out	MEM 段的 DMemRead
output	[0:0]	DMemWrite_out	MEM 段的 DMemWrite
output	[0:0]	DMemtoReg_out	MEM 段的 DMemtoReg
output	[0:0]	RegWrite_out	MEM 段的 RegWrite

MEMWB.v			
输入/输出	宽度	信号名	说明
input	[0:0]	clk	时钟沿
input	[0:0]	en	使能信号高电平有效
input	[0:0]	flush	清空信号高电平有效
input	[31:0]	ALUResult_in	MEM 段的 ALUResult
input	[31:0]	DMemout_in	MEM 段的 Dmemout
input	[4:0]	RegWtaddr_in	MEM 段的 RegWtaddr
output	[31:0]	ALUResult_out	WB 段的 ALUResult
output	[31:0]	DMemout_out	WB 段的 DMemout
output	[4:0]	RegWtaddr_out	WB 段的 RegWtaddr
input	[0:0]	DMemtoReg_in	MEM 段的 DMemtoReg
input	[0:0]	RegWrite_in	MEM 段的 RegWrite
output	[0:0]	DMemtoReg_out	WB 段的 DMemtoReg
output	[0:0]	RegWrite_out	WB 段的 RegWrite

forward.v			
输入/输出	宽度	信号名	说明
input	[4:0]	Rs_EX	EX 段的 Rs
input	[4:0]	Rt_EX	EX 段的 Rt
input	[0:0]	RegWrite_MEM	MEM 段的 RegWrite
input	[0:0]	RegWrite_WB	WB 段的 RegWrite
input	[4:0]	RegWtaddr_MEM	MEM 段的 RegWtaddr
input	[4:0]	RegWtaddr_WB	WB 段的 RegWtaddr
output	[1:0]	RegRdout1Sel_Forward_EX	EX 段的 RegRdout1Sel_Forward
output	[1:0]	RegRdout2Sel_Forward_EX	EX 段的 RegRdout2Sel_Forward

hazard.v			
输入/输出	宽度	信号名	说明
input	[4:0]	Rs_ID	ID 段的 Rs
input	[4:0]	Rt_ID	ID 段的 Rt
input	[4:0]	RegWtaddr_EX	EX 段的 RegWtaddr
input	[0:0]	DMemRead_EX	EX 段的 DMemRead
output	[0:0]	PCEn	允许 PC 更新，高电平有效
output	[0:0]	IF_ID_En	允许 IFID 更新，高电平有效
output	[0:0]	ID_EX_Flush	IDEX 清空，高电平有效

control.v			
输入/输出	宽度	信号名	说明
input	[0:0]	clk	时钟沿
input	[0:0]	rst	复位信号高电平有效
input	[5:0]	Op	
input	[4:0]	Rt	
input	[5:0]	Funct	
input	[1:0]	RsCMPRt	Rs 和 Rt 寄存器比较结果
input	[1:0]	RsCMPZero	Rs 寄存器和 0 比较结果
output	[1:0]	PCSrc	0:+4，1:Branch，2:J，3:JR
output	[0:0]	RegDst	0:RegWtaddr=rt，1:RegWtaddr=rd
output	[0:0]	ALUSrcASel	0:RegRdout1，1:ShamtZeroExtended
output	[1:0]	ALUSrcBSel	0:RegRdout2，1:IMMSignExtended，2:IMMZeroExtended
output	[4:0]	ALUControl	
output	[0:0]	DMemRead	1:En
output	[0:0]	DMemWrite	1:En
output	[0:0]	DMemtoReg	0:Aluout，1:DMemout
output	[0:0]	RegWrite	1:En

ALU：

```
22  `define A_NOP 5'd00 //nop
23  `define A_ADD 5'd01 //signed_add
24  `define A_SUB 5'd02 //signed_sub
25  `define A_AND 5'd03 //and
26  `define A_OR  5'd04 //or
27  `define A_XOR 5'd05 //xor
28  `define A_NOR 5'd06 //nor
29  `define A_ADDU 5'd07 //unsigned_add
30  `define A_SUBU 5'd08 //unsigned_sub
31  `define A_SLT 5'd09 //slt
32  `define A_SLTU 5'd10 //unsigned_slt
33  `define A_SLL 5'd11 //sll
34  `define A_SRL 5'd12 //srl
35  `define A_SRA 5'd13 //sra
36  `define A_MOV 5'd14 //movz,movn
37  `define A_LUI 5'd15 //lui
38  module alu(
39      input [31:0] alu_a,//无符号型的，如果有负数，是以补码存储
40      input [31:0] alu_b,
41      input [4:0] alu_op,
42      output reg [31:0] alu_out
43
44      );
45      always@(*)
46          case (alu_op)
47              `A_NOP: alu_out = 0;
48              `A_ADD: alu_out = alu_a + alu_b;
49              `A_SUB: alu_out = alu_a - alu_b;
50              `A_AND: alu_out = alu_a & alu_b;
```

```
            `A_OR : alu_out = alu_a | alu_b;
            `A_XOR: alu_out = alu_a ^ alu_b;
            `A_NOR: alu_out = ~(alu_a | alu_b);
            `A_ADDU: alu_out = alu_a + alu_b;
            `A_SUBU: alu_out = alu_a - alu_b;
            `A_SLT: //a<b(signed) return 1 else return 0;
                begin
                    if(alu_a[31] == alu_b[31]) alu_out = (alu_a < alu_b) ? 32'b1 : 32'b0;
                    //对于不加signed的变量类型，运算和比较视为无符号，但依然可以存储有符号数，这里相当于自行根据首位判断
                    //首位相等，即同号情况，直接比较，如果同正，后面31位大的，原数就大，如果同负，后面31位（补码）大的，依然
                    else alu_out = (alu_a[31] < alu_b[31]) ? 32'b0 : 32'b1;//异号情况，直接比较符号
                end
            /*`A_SLT://法2：使用$signed()
                alu_out = ($signed(alu_a) < $signed(alu_b)) ? 32'b1 : 32'b0;*/
            `A_SLTU: alu_out = (alu_a < alu_b) ? 32'b1 : 32'b0;
            `A_SLL: alu_out = alu_b << alu_a;
            `A_SRL: alu_out = alu_b >> alu_a;
            `A_SRA: alu_out = $signed(alu_b) >>> alu_a;
            //使用>>>为算术右移，高位补符号，应该注意，如果是无符号数，>>>仍是逻辑右移，故应该$signed
            `A_MOV: alu_out = alu_b;//原样输出，相当于reg[rt],mov本不需要通过alu，但因为是RType格式，故统一
            `A_LUI: alu_out = alu_b << 16;
            default: ;
        endcase
endmodule
```

Regfile:

```
module regfile(
    input   clk,
    input rst_n,
    input [4:0] rAddr1,//读地址1
    output [31:0] rDout1,//读数据1
    input [4:0] rAddr2,//读地址2
    output [31:0] rDout2,//读数据2
    input [4:0] wAddr,//写地址
    input [31:0] wDin,//写数据
    input wEna//写使能
);
    reg [31:0] data [0:31];
    integer i;
    assign rDout1=data[rAddr1];//读1
    assign rDout2=data[rAddr2];//读2

    always@(posedge clk or negedge rst_n)//写和复位
        if(~rst_n)
        begin
            for(i=0; i<32; i=i+1) data[i]<=0;
        end
        else
        begin
            if(wEna)
                if(wAddr!=0)
                    data[wAddr]<=wDin;
        end
endmodule
```

2 路和 4 路选择器：

```
module mux #(parameter WIDTH = 32)(
    input sel,
    input [WIDTH-1:0] d0,
    input [WIDTH-1:0] d1,
    output [WIDTH-1:0] out
    );
    assign out = (sel == 1'b1 ? d1 : d0);
endmodule
```

```
module mux4 #(parameter WIDTH = 32)(
     input [1:0] sel,
     input [WIDTH-1:0] d0,
     input [WIDTH-1:0] d1,
     input [WIDTH-1:0] d2,
     input [WIDTH-1:0] d3,
     output reg [WIDTH-1:0] out
     );
     always@(*)
         case(sel)
             2'b00: out=d0;
             2'b01: out=d1;
             2'b10: out=d2;
             2'b11: out=d3;
             default:;
         endcase
endmodule
```

D 触发器：

```
module dff #(parameter WIDTH = 32) ( //Data Flip-Flop
    input clk,
    input en,
    input rst,
    input [WIDTH-1:0] datain,
    output reg [WIDTH-1:0] dataout
    );
    always@(posedge clk)
    begin
        if(rst)
            dataout <= 0;
        else if(en)
            dataout <= datain;
    end
endmodule
```

IFID:

```
module IFID(
    input clk,
    input en,
    input flush,
    input [31:0] PCPlus_in,
    input [31:0] IMemout_in,
    output [31:0] PCPlus_out,
    output [31:0] IMemout_out
    );
     dff dff1(clk,en,flush,PCPlus_in,PCPlus_out);
     dff dff2(clk,en,flush,IMemout_in,IMemout_out);
endmodule
```

IDEX:

```
module IDEX(
    input clk,
    input en,
    input flush,//flush for stall or start
    input [31:0] PCPlus_in,
    input [31:0] RegRdout1_in,
    input [31:0] RegRdout2_in,
    input [31:0] IMMSignExtended_in,
    input [31:0] IMMZeroExtended_in,
    input [31:0] ShamtZeroExtended_in,
    input [4:0] Rs_in,
    input [4:0] Rt_in,
    input [4:0] RegWtaddr_in,
    output [31:0] PCPlus_out,
    output [31:0] RegRdout1_out,
    output [31:0] RegRdout2_out,
    output [31:0] IMMSignExtended_out,
    output [31:0] IMMZeroExtended_out,
    output [31:0] ShamtZeroExtended_out,
    output [4:0] Rs_out,
    output [4:0] Rt_out,
    output [4:0] RegWtaddr_out,
     //control
    input RegDst_in,
    input ALUSrcASel_in,
    input [1:0] ALUSrcBSel_in,
    input [4:0] ALUControl_in,
    input DMemRead_in,
    input DMemWrite_in,
```

```
    input DMemtoReg_in,
    input RegWrite_in,
    output RegDst_out,
    output ALUSrcASel_out,
    output [1:0] ALUSrcBSel_out,
    output [4:0] ALUControl_out,
    output DMemRead_out,
    output DMemWrite_out,
    output DMemtoReg_out,
    output RegWrite_out
    );
     dff dff1(clk, en, flush, PCPlus_in, PCPlus_out);
     dff dff2(clk, en, flush, RegRdout1_in, RegRdout1_out);
     dff dff3(clk, en, flush, RegRdout2_in, RegRdout2_out);
     dff dff4(clk, en, flush, IMMSignExtended_in, IMMSignExtended_out);
     dff dff5(clk, en, flush, IMMZeroExtended_in, IMMZeroExtended_out);
     dff dff6(clk, en, flush, ShamtZeroExtended_in, ShamtZeroExtended_out);
     dff #(5) dff7(clk, en, flush, Rs_in, Rs_out);
     dff #(5) dff8(clk, en, flush, Rt_in, Rt_out);
     dff #(5) dff9(clk, en, flush, RegWtaddr_in, RegWtaddr_out);
     dff #(1) dff10(clk, en, flush, RegDst_in, RegDst_out);
     dff #(1) dff11(clk, en, flush, ALUSrcASel_in, ALUSrcASel_out);
     dff #(2) dff12(clk, en, flush, ALUSrcBSel_in, ALUSrcBSel_out);
     dff #(5) dff13(clk, en, flush, ALUControl_in, ALUControl_out);
     dff #(1) dff14(clk, en, flush, DMemRead_in, DMemRead_out);
     dff #(1) dff15(clk, en, flush, DMemWrite_in, DMemWrite_out);
     dff #(1) dff16(clk, en, flush, DMemtoReg_in, DMemtoReg_out);
     dff #(1) dff17(clk, en, flush, RegWrite_in, RegWrite_out);
endmodule
```

EXMEM:

```
module EXMEM(
      input clk,
      input en,
      input flush,
      input [31:0] ALUResult_in,
      input [31:0] DMemin_in,
      input [4:0] RegWtaddr_in,
      output [31:0] ALUResult_out,
      output [31:0] DMemin_out,
      output [4:0] RegWtaddr_out,
      //control
      input DMemRead_in,
      input DMemWrite_in,
      input DMemtoReg_in,
      input RegWrite_in,
      output DMemRead_out,
      output DMemWrite_out,
      output DMemtoReg_out,
      output RegWrite_out
     );
      dff dff1(clk, en, flush, ALUResult_in, ALUResult_out);
      dff dff2(clk, en, flush, DMemin_in, DMemin_out);
      dff #(5) dff3(clk, en, flush, RegWtaddr_in, RegWtaddr_out);
      dff #(1) dff14(clk, en, flush, DMemRead_in, DMemRead_out);
      dff #(1) dff15(clk, en, flush, DMemWrite_in, DMemWrite_out);
      dff #(1) dff16(clk, en, flush, DMemtoReg_in, DMemtoReg_out);
      dff #(1) dff17(clk, en, flush, RegWrite_in, RegWrite_out);
endmodule
```

MEMWB:

```
module MEMWB(
      input clk,
      input en,
      input flush,
      input [31:0] ALUResult_in,
      input [31:0] DMemout_in,
      input [4:0] RegWtaddr_in,
      output [31:0] ALUResult_out,
      output [31:0] DMemout_out,
      output [4:0] RegWtaddr_out,
      //control
      input DMemtoReg_in,
      input RegWrite_in,
      output DMemtoReg_out,
      output RegWrite_out
    );
      dff dff1(clk,en,flush,ALUResult_in,ALUResult_out);
      dff dff2(clk,en,flush,DMemout_in,DMemout_out);
      dff #(5) dff3(clk,en,flush,RegWtaddr_in,RegWtaddr_out);
      dff #(1) dff16(clk,en,flush,DMemtoReg_in,DMemtoReg_out);
      dff #(1) dff17(clk,en,flush,RegWrite_in,RegWrite_out);
endmodule
```

compare:

```
`define LESS 2'b00
`define EQUAL 2'b01
`define GREATER 2'b10
module compare(
      input signed [31:0] a,
      input signed [31:0] b,
      output reg [1:0] res
      //output isEqual
    );
      always @(*)
         if(a == b) res = 2'b01;
         else if(a < b) res = 2'b00;
         else if(a > b) res = 2'b10;
    //assign isEqual = (a == b ? 1 : 0);
endmodule
```

SignExtended 模块——立即数符号扩展

过于简单，直接在 top 中实现了。

ZeroExtended 模块——立即数无符号扩展

过于简单，直接在 top 中实现了。

forward:

```
module forward(
    input [4:0] Rs_EX,
    input [4:0] Rt_EX,
    input RegWrite_MEM,
    input RegWrite_WB,
    input [4:0] RegWtaddr_MEM,
    input [4:0] RegWtaddr_WB,
    output reg [1:0] RegRdout1Sel_Forward_EX,
    output reg [1:0] RegRdout2Sel_Forward_EX
    );
    always @(*) begin
        RegRdout1Sel_Forward_EX[0] = RegWrite_WB && (RegWtaddr_WB != 0) && (RegWtaddr_MEM != Rs_EX) && (RegWtaddr_WB == Rs_EX);
        RegRdout1Sel_Forward_EX[1] = RegWrite_MEM && (RegWtaddr_MEM != 0) && (RegWtaddr_MEM == Rs_EX);
        RegRdout2Sel_Forward_EX[0] = RegWrite_WB && (RegWtaddr_WB != 0) && (RegWtaddr_MEM != Rt_EX) && (RegWtaddr_WB == Rt_EX);
        RegRdout2Sel_Forward_EX[1] = RegWrite_MEM && (RegWtaddr_MEM != 0) && (RegWtaddr_MEM == Rt_EX);
    end
endmodule
```

特别注意到，目的寄存器是 0 寄存器的话就不进行转发。

hazard:

```
module hazard(
     input clk,
     input rst,
     input [5:0] Op,
     input [4:0] Rs_ID,
     input [4:0] Rt_ID,
     input [4:0] RegWtaddr_EX,
     input [4:0] RegWtaddr_MEM,
     input DMemRead_EX,
     input DMemRead_MEM,
     output PCEn,
     output IF_ID_En,
     output ID_EX_Flush,
     output reg    [1:0]   count
    );
    assign ID_EX_Flush = ((((RegWtaddr_EX == Rs_ID) || (RegWtaddr_EX == Rt_ID)) && (DMemRead_EX || Op==6'b000100))
                          || (((RegWtaddr_MEM == Rs_ID) || (RegWtaddr_MEM == Rt_ID))
                          && (DMemRead_MEM || Op==6'b000100)))&&(count!=3);//条件成立则为1，清空
    assign IF_ID_En = ~ID_EX_Flush;//条件成立则为0，保持
    assign PCEn = (~ID_EX_Flush );//条件成立则为0，保持
    always@(posedge clk or posedge rst)
    begin
       if(rst)
           count<=0;
       else    if(((RegWtaddr_EX == Rs_ID) || (RegWtaddr_EX == Rt_ID)) && (Op==6'b000100) && (count!=1) && (count!=2))
                   count<=1;
       else    if(count==1)
                   count<=2;
       else    if(count==2)
                   count<=3;
       else    if(count==3)
                   count<=0;
    end
endmodule
```

这是冒险单元，用于：

（1）上一条指令时 LW 指令且目的寄存器是当前指令 ID 级读的寄存器那么插入气泡。

（2）特殊情况：如果当前指令是 beq 指令，且上条指令的目的寄存器（例如 lw,add…）是当且 beq 指令读的寄存器则插入两个气泡。

注：特别注意的是：如果当且 beq 指令读的寄存器中至少有一个是零寄存器。因为气泡译码后，rs,rt,rd 均为零寄存器，如果不进行计数的话会根据判断条件无限产生气泡，加入 count 进行计数（统计产生气泡的数目），防止产生无限气泡。

control：

```
module control(
    input clk,rst,
    input [5:0] Op, //instr[31:26]
    input [4:0] Rt, //instr[20:16]
    input [5:0] Funct,//instr[5:0]
    input [1:0] RsCMPRt,
    input [1:0] RsCMPZero,
    output reg [1:0] PCSrc, //0:+4,1:Branch,2:J,3:JR
    output reg RegDst,//0:RegWtaddr=rt,1:RegWtaddr=rd
    output reg ALUSrcASel,//0:RegRdout1,1:ShamtZeroExtended
    output reg [1:0] ALUSrcBSel, //0:RegRdout2,1:IMMSignExtended,2:IMMZeroExtended
    output reg [4:0] ALUControl,
    output reg DMemRead,//1:En
    output reg DMemWrite,//1:En
    output reg DMemtoReg,//0:Aluout,1:DMemout
    output reg RegWrite//1:En
    );
    reg [1:0] tmpsrc;
    always @(*)
    begin
    if(rst)
        begin
            {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b00_0_0_00},{`A_NOP},{4'b0001}};
        end
    else
                case(Op)
                    6'b000000: //R-Type
                        case(Funct)
                            6'b000000: //NOP
                                {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg },{RegWrite}}={{6'b00_1_1_00},{`A_SLL},{4'b0000}};
                            6'b100000: //ADD
                                {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b00_1_0_00},{`A_ADD},{4'b0001}};
                        endcase
                    6'b000010: //J,无条件跳转
                        {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b10_z_z_zz},{`A_NOP},{4'b00z0}};
                    6'b000100: //BEQ,Reg[rs]==Reg[rt]则跳转,RsCMPRt=01(==),则PCSrc=01，否则PCSrc=00(不跳转)
                        {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{{1'b0},{RsCMPRt[0]}},{4'bz_z_zz},{`A_NOP},{4'b00z0}};
                    6'b001000: //ADDI,注意RegDst=0,AluBSrcSel=01(IMMSignExtended),下面三个同理
                        {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b00_0_0_01},{`A_ADD},{4'b0001}};
                    6'b100011: //LW,注意RegDst=0(写到Reg[rt]),AluBSrcSel=01(IMMSignExtended),DMemtoReg=1(来自DMem),
                        {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b00_0_0_01},{`A_ADD},{4'b1011}};
                    6'b101011: //SW,注意RegDst=x(不写Reg),AluBSrcSel=01(IMMSignExtended)
                        {PCSrc},{RegDst},{ALUSrcASel},{ALUSrcBSel},{ALUControl},{DMemRead},{DMemWrite},{DMemtoReg},{RegWrite}}={{6'b00_z_0_01},{`A_ADD},{4'b01z0}};
                    default: ;
                endcase
    end
endmodule
```

开关去抖动：

```
module debounce(
    input clk,
    input in,
    output reg out=0
    );
    reg [31:0] cnt=0;
    always@(posedge clk)
        begin
            if(in!=out)
                begin
                    cnt=cnt+1;

                    if(cnt==100000)
                    begin
                        out=~out;
                        cnt=0;
                    end

                end
            else cnt=0;

        end
endmodule
```

数码管：

```
module seg(
        input clk,
        input rst_n,
        input [31:0] data32,
        output reg [3:0] sel,
        output reg [6:0] segments
    );
    integer clk_25=0;//数码管循环显示用
    integer clk_50000000=0;//2hz，移动显示用
    reg [1:0] cnt;
    reg [3:0] cnt2;
    reg [15:0] data16;//data32的16bit
    reg [3:0] data4;//data16的4bit
    reg [3:0] empty;//空白位
    always@(*)//组合逻辑，控制数码管
        begin
            if(!rst_n)
                segments = 7'b000_0000;
            else
                case(data4)
                    0: segments =       ~7'b011_1111;//0
                    1: segments =       ~7'b000_0110;//1
                    2: segments =       ~7'b101_1011;//2
                    3: segments =       ~7'b100_1111;//3
                    4: segments =       ~7'b110_0110;//4
                    5: segments =       ~7'b110_1101;//5
                    6: segments =       ~7'b111_1101;//6
                    7: segments =       ~7'b000_0111;//7
                    8: segments =       ~7'b111_1111;//8
```

```
                    9: segments =       ~7'b110_1111;//9
                    10:segments =       ~7'b111_0111;//A
                    11:segments =       ~7'b111_1100;//b
                    12:segments =       ~7'b011_1001;//C
                    13:segments =       ~7'b101_1110;//d
                    14:segments =       ~7'b111_1001;//E
                    15:segments =       ~7'b111_0001;//F
                    default: segments = 7'b000_0000; // required
                endcase
        end

    always@(posedge clk)//时序逻辑，产生位选择信号段选择信号
        begin
                if(clk_25==400000)
                    begin
                        clk_25=0;
                        cnt = cnt + 2'b01;
                    end
                else
                    clk_25=clk_25+1;
                if(clk_50000000==50000000)//
                    begin
                        clk_50000000=0;
                        cnt2=cnt2+1;
                        if(cnt2==4'b1010) cnt2=4'b0000;
                    end
                else
                    clk_50000000=clk_50000000+1;
            end

    always@(*)//组合逻辑，选择当前显示段
        begin
            case(cnt2)
                4'b0000:begin data16={8'bzzzzzzzz,data32[31:24]}; empty=4'b1100; end

                4'b0001:begin data16={4'bzzzz,data32[31:20]}; empty=4'b1000; end

                4'b0010:begin data16=data32[31:16]; empty=4'b0000; end

                4'b0011:begin data16=data32[27:12]; empty=4'b0000; end

                4'b0100:begin data16=data32[23:8]; empty=4'b0000; end

                4'b0101:begin data16=data32[19:4]; empty=4'b0000; end

                4'b0110:begin data16=data32[15:0]; empty=4'b0000; end

                4'b0111:begin data16={data32[11:0],4'bzzzz}; empty=4'b0001; end

                4'b1000:begin data16={data32[7:0],8'bzzzzzzzz}; empty=4'b0011; end

                4'b1001:begin data16={data32[3:0],8'bzzzzzzzz,data32[31:28]}; empty=4'b0110; end
                default:;
            endcase
        end

    always@(*)//组合逻辑，选择当前显示位
        begin
            case(cnt)
```

```
                2'b00:sel=4'b1110 | empty;
                2'b01:sel=4'b1101 | empty;
                2'b10:sel=4'b1011 | empty;
                2'b11:sel=4'b0111 | empty;
                default:sel=4'b1110;
            endcase
        end

    always@(*)//组合逻辑，选择当前显示位的数据
        begin
            case(cnt)
                2'b00:data4=data16[3:0];
                2'b01:data4=data16[7:4];
                2'b10:data4=data16[11:8];
                2'b11:data4=data16[15:12];
                default:data4=16'b0;
            endcase
        end
endmodule
```

TOP：

```
module top(
    input   clock,
    input   [7:0] sw,
    output  [7:0] seg7,
    output  [7:0] an,
    output  Led,
    input   btns,//rst
    input   read
    );
    wire clk;
    assign  clk=(read==0)?clock:1'b0;
//_后缀表示该信号所在的流水段
    wire ALUSrcASel_ID;
    wire ALUSrcASel_EX;
    wire [1:0] ALUSrcBSel_ID;//alu B在regout2和imm之间选择
    wire [1:0] ALUSrcBSel_EX;
    wire [31:0] ALUSrcA_EX;
    wire [31:0] ALUSrcB_EX;
    wire [4:0] ALUControl_ID;
    wire [4:0] ALUControl_EX;
    wire [31:0] ALUResult_EX;
    wire [31:0] ALUResult_MEM;
    wire [31:0] ALUResult_WB;

    wire [1:0] RsCMPZero;
    wire [1:0] RsCMPRt;

    wire [31:0] IMMSignExtended_ID;
    wire [31:0] IMMSignExtended_EX;
```

```
52      wire [31:0] IMMZeroExtended_ID;
53      wire [31:0] IMMZeroExtended_EX;
54      wire [31:0] ShamtZeroExtended_ID;
55      wire [31:0] ShamtZeroExtended_EX;
56
57      wire [1:0] RegRdout1Sel_Forward_EX;//旁路单元产生的选择信号
58      wire [1:0] RegRdout2Sel_Forward_EX;
59      wire [31:0] RegRdout1_Forward_EX;//旁路数据
60      wire [31:0] RegRdout2_Forward_EX;
61
62      wire [4:0] RegRdaddr1_ID;
63      wire [31:0] RegRdout1_ID;
64      wire [31:0] RegRdout1_EX;
65      wire [4:0] RegRdaddr2_ID;
66      wire [31:0] RegRdout2_ID;
67      wire [31:0] RegRdout2_EX;
68      wire [4:0] RegWtaddr_ID;
69      wire [4:0] RegWtaddr_EX;
70      wire [4:0] RegWtaddr_MEM;
71      wire [4:0] RegWtaddr_WB;
72      wire [31:0] RegWtin_WB;
73      wire RegWrite_ID;
74      wire RegWrite_EX;
75      wire RegWrite_MEM;
76      wire RegWrite_WB;
77      wire RegDst_ID;
78      wire RegDst_EX;
79
80      wire [31:0] IMemaddr;
81      wire [31:0] IMemout;
82
83      wire [31:0] DMemaddr_MEM;
84      wire [31:0] DMemin_MEM;
85      wire DMemRead_MEM;
86      wire [31:0] DMemout_MEM;
87      wire [31:0] DMemout_WB;
88      wire DMemWrite_MEM;
89      wire DMemtoReg_EX;
90      wire DMemtoReg_MEM;
91      wire DMemtoReg_WB;
92      wire DMemRead_ID;
93      wire DMemWrite_ID;
94      wire DMemtoReg_ID;
95      wire DMemRead_EX;
96      wire DMemWrite_EX;
97
98      wire [31:0] PC;
99      wire [31:0] PCPlus_IF;
100     wire [31:0] PCPlus_ID;
101     wire [31:0] PCPlus_EX;
102     wire [31:0] EPC;
103     wire [31:0] nextPC;
104     wire PCEn;
105     wire [1:0] PCSrc_ID;//Control输出的，0:+4,1:Branch,2:J
106     wire IF_ID_En;
107     wire IF_ID_Flush;
108     wire ID_EX_Flush;
```

```
    wire [31:0] PCJump_ID;
    wire [31:0] PCJR_ID;
    wire [31:0] PCBranch_ID;

    wire [31:0] Instr;
    wire [5:0] Funct;
    wire [4:0] Shamt;
    wire [15:0] IMM16;
    wire [4:0] Rd;
    wire [4:0] Rt;
    wire [4:0] Rs;
    wire [5:0] Op;
    wire [4:0] Rt_EX;//为了旁路判断
    wire [4:0] Rs_EX;//为了旁路判断

    wire [25:0] JumpIMM;
    wire [31:0] IMMSignExtendedShiftLeft2;
    wire btns_d;
    //debounce  debounce(clk,btns,btns_d);//中键去抖动

    //reg rst;
    //assign Led = rst;
    //always @(posedge btns_d) rst=~rst;
    wire rst;
    wire    [1:0]   count;
    assign rst=btns;

    mux4 MUXPC(
        .sel(PCSrc_ID),
        .d0(PCPlus_IF),//+4直接用IF的
        .d1(PCBranch_ID),
        .d2(PCJump_ID),
        .d3(PCJR_ID),
        .out(nextPC)
    );

    dff DFFPC(
        .clk(~clk),
        .en(PCEn),
        .rst(rst),
        .datain(nextPC),
        .dataout(PC)
    );

    alu ALUPCPlus(PC,4,5'd01,PCPlus_IF);

    assign IMemaddr = PC >> 2;//>>2是因为这里IMem是每个地址存储4字节，和实际上的（一地址一字节）不一样
    IMem IMem(.a(IMemaddr),.spo(IMemout));//上升沿读指令

    //======================IFID=======================

    IFID IFID(
        .clk(~clk),
        .en(IF_ID_En),
        .flush((IF_ID_Flush  ) || rst),
        .PCPlus_in(PCPlus_IF),
        .IMemout_in(IMemout),
        .PCPlus_out(PCPlus_ID),
```

```
168             .IMemout_out(Instr)
169         );
170 
171         //======================ID=========================
172 
173         assign JumpIMM = Instr[25:0];
174         assign Funct = Instr[5:0];
175         assign Shamt = Instr[10:6];
176         assign IMM16 = Instr[15:0];
177         assign Rd = Instr[15:11];
178         assign Rt = Instr[20:16];
179         assign Rs = Instr[25:21];
180         assign Op = Instr[31:26];
181 
182         //*******Control*******
183 
184         control control(
185             //in
186             .clk(clk),
187             .rst(rst),
188             .Op(Op),
189             .Rt(Rt),
190             .Funct(Funct),
191             .RsCMPRt(RsCMPRt),
192             .RsCMPZero(RsCMPZero),
193             //out
194             .PCSrc(PCSrc_ID),
195             //ID
196             .RegDst(RegDst_ID),
197             //EX
198             .ALUSrcASel(ALUSrcASel_ID),
199             .ALUSrcBSel(ALUSrcBSel_ID),
200             .ALUControl(ALUControl_ID),
201             //MEM
202             .DMemRead(DMemRead_ID),
203             .DMemWrite(DMemWrite_ID),
204             //WB
205             .DMemtoReg(DMemtoReg_ID),
206             .RegWrite(RegWrite_ID)
207         );
208         //*******Control*******
209 
210         assign RegRdaddr2_ID = Rt;
211         mux #(5) MUXRegWtaddr(RegDst_ID,Rt,Rd,RegWtaddr_ID);
212 
213         assign ShamtZeroExtended_ID = {{27{1'b0}},Shamt};
214         assign IMMSignExtended_ID = {{16{IMM16[15]}},IMM16};
215         assign IMMZeroExtended_ID = {{16{1'b0}},IMM16};
216         assign IMMSignExtendedShiftLeft2 = IMMSignExtended_ID << 2;
217         alu BranchALU(PCPlus_ID,IMMSignExtendedShiftLeft2,5'd01,PCBranch_ID);
218         assign PCJump_ID = {{PCPlus_ID[31:28]},{{2'b00,JumpIMM}<<2}};
219         assign PCJR_ID = RegRdout1_ID;
220         assign IF_ID_Flush = (PCSrc_ID == 2'b00)?1'b0:(PCSrc_ID == 2'b10)?1'b1:(count==3)?1'b1:1'b0;//有跳转则清空IF_ID寄存器
221         assign RegRdaddr1_ID = (read==1'b0)?(Rs):sw;
222         regfile regfile(clk,~rst,RegRdaddr1_ID,RegRdout1_ID,RegRdaddr2_ID,RegRdout2_ID,RegWtaddr_WB,RegWtin_WB,RegWrite_WB);
223 
224         compare compare1(RegRdout1_ID,RegRdout2_ID,RsCMPRt);//for beq,bne
225         hazard hazard(clk,rst,Op,Rs,Rt,RegWtaddr_EX,RegWtaddr_MEM,DMemRead_EX,DMemRead_MEM,PCEn,IF_ID_En,ID_EX_Flush,count);
```

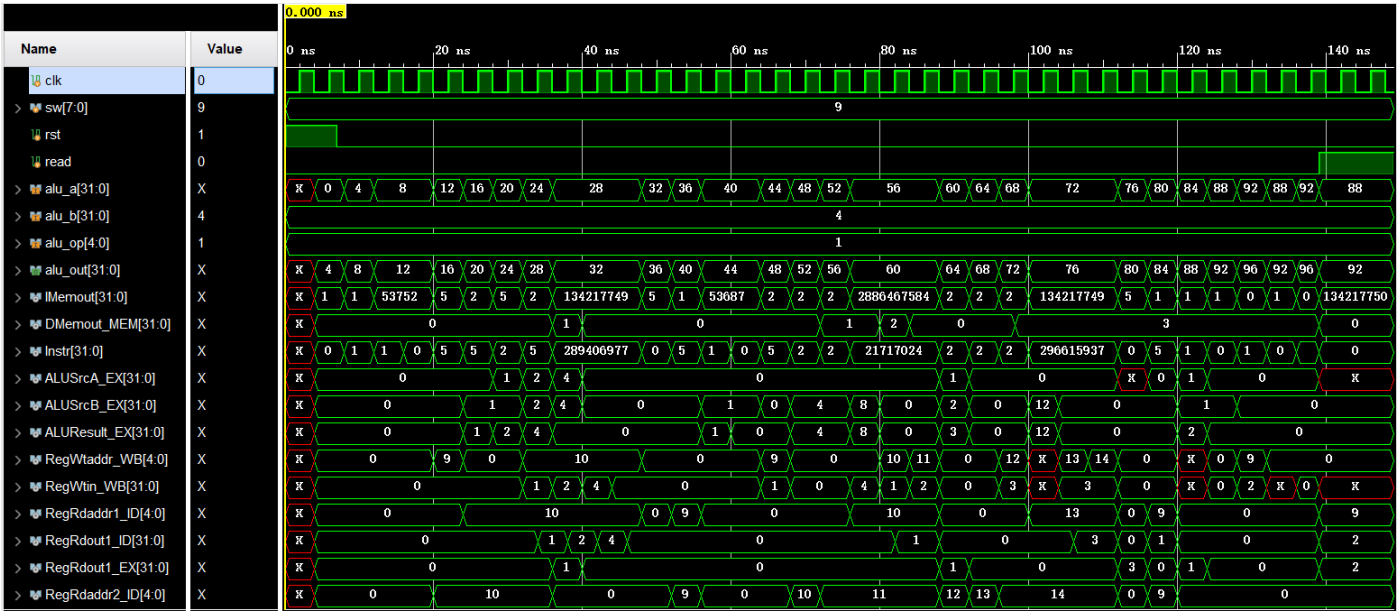
```
    //======================IDEX=========================

    IDEX IDEX(
        .clk(~clk),
        .en(1'b1),
        .flush(ID_EX_Flush || rst),
        .PCPlus_in(PCPlus_ID),
        .RegRdout1_in(RegRdout1_ID),
        .RegRdout2_in(RegRdout2_ID),
        .IMMSignExtended_in(IMMSignExtended_ID),
        .IMMZeroExtended_in(IMMZeroExtended_ID),
        .ShamtZeroExtended_in(ShamtZeroExtended_ID),
        .Rs_in(Rs),
        .Rt_in(Rt),
        .RegWtaddr_in(RegWtaddr_ID),
        .PCPlus_out(PCPlus_EX),
        .RegRdout1_out(RegRdout1_EX),
        .RegRdout2_out(RegRdout2_EX),
        .IMMSignExtended_out(IMMSignExtended_EX),
        .IMMZeroExtended_out(IMMZeroExtended_EX),
        .ShamtZeroExtended_out(ShamtZeroExtended_EX),
        .Rs_out(Rs_EX),
        .Rt_out(Rt_EX),
        .RegWtaddr_out(RegWtaddr_EX),
        .RegDst_in(RegDst_ID),
        .ALUSrcASel_in(ALUSrcASel_ID),
        .ALUSrcBSel_in(ALUSrcBSel_ID),
        .ALUControl_in(ALUControl_ID),
        .DMemRead_in(DMemRead_ID),
        .DMemWrite_in(DMemWrite_ID),
        .DMemtoReg_in(DMemtoReg_ID),
        .RegWrite_in(RegWrite_ID),
        .RegDst_out(RegDst_EX),
        .ALUSrcASel_out(ALUSrcASel_EX),
        .ALUSrcBSel_out(ALUSrcBSel_EX),
        .ALUControl_out(ALUControl_EX),
        .DMemRead_out(DMemRead_EX),
        .DMemWrite_out(DMemWrite_EX),
        .DMemtoReg_out(DMemtoReg_EX),
        .RegWrite_out(RegWrite_EX)
    );

    //======================EX=========================

    forward forward(Rs_EX,Rt_EX,RegWrite_MEM,RegWrite_WB,RegWtaddr_MEM,RegWtaddr_WB,RegRdout1Sel_Forward_EX,RegRdout2Sel_Forward_EX);
    mux4 MUXRegRdout1FW(RegRdout1Sel_Forward_EX,RegRdout1_EX,RegWtin_WB,ALUResult_MEM,0,RegRdout1_Forward_EX);//forward
    mux4 MUXRegRdout2FW(RegRdout2Sel_Forward_EX,RegRdout2_EX,RegWtin_WB,ALUResult_MEM,0,RegRdout2_Forward_EX);//forward
    mux MUXALUSrcA(ALUSrcASel_EX,RegRdout1_Forward_EX,ShamtZeroExtended_EX,ALUSrcA_EX);
    mux4 MUXALUSrcB(ALUSrcBSel_EX,RegRdout2_Forward_EX,IMMSignExtended_EX,IMMZeroExtended_EX,0,ALUSrcB_EX);
    alu alu(ALUSrcA_EX,ALUSrcB_EX,ALUControl_EX,ALUResult_EX);

    //======================EXMEM=========================

    EXMEM EXMEM(
        .clk(~clk),
        .en(1'b1),
        .flush(rst),
        .ALUResult_in(ALUResult_EX),
```

```
285         .DMemin_in(RegRdout2_Forward_EX),
286         .RegWtaddr_in(RegWtaddr_EX),
287         .ALUResult_out(ALUResult_MEM),
288         .DMemin_out(DMemin_MEM),
289         .RegWtaddr_out(RegWtaddr_MEM),
290         .DMemRead_in(DMemRead_EX),
291         .DMemWrite_in(DMemWrite_EX),
292         .DMemtoReg_in(DMemtoReg_EX),
293         .RegWrite_in(RegWrite_EX),
294         .DMemRead_out(DMemRead_MEM),
295         .DMemWrite_out(DMemWrite_MEM),
296         .DMemtoReg_out(DMemtoReg_MEM),
297         .RegWrite_out(RegWrite_MEM)
298     );
299
300     //======================MEM=======================
301
302     assign DMemaddr_MEM = (read==1'b0)?(ALUResult_MEM>>2):sw;
303     DMem DMem(.clk(clk),.we(DMemWrite_MEM),.a(DMemaddr_MEM ),.d(DMemin_MEM),.spo(DMemout_MEM));
304
305     //======================MEMWB=======================
306
307     MEMWB MEMWB(
308         .clk(~clk),
309         .en(1'b1),
310         .flush(rst),
311         .ALUResult_in(ALUResult_MEM),
312         .DMemout_in(DMemout_MEM),
313         .RegWtaddr_in(RegWtaddr_MEM),
314         .ALUResult_out(ALUResult_WB),
315         .DMemout_out(DMemout_WB),
316         .RegWtaddr_out(RegWtaddr_WB),
317         .DMemtoReg_in(DMemtoReg_MEM),
318         .RegWrite_in(RegWrite_MEM),
319         .DMemtoReg_out(DMemtoReg_WB),
320         .RegWrite_out(RegWrite_WB)
321     );
322     //======================WB=======================
323     mux MUXDMemtoReg(DMemtoReg_WB,ALUResult_WB,DMemout_WB,RegWtin_WB);
324 endmodule
```

五、实验结果：

Test2 的测试结果：

（以十进制显示）



可以看到最终 9 号寄存器结果为 2.

六、心得体会：

通过本次实验理解了流水线 CPU 的组成结构和工作原理；掌握了数字系统的设计和调试方法；熟练掌握了数据通路和控制器的设计和描述方法。