P5课下-Verilog搭建流水线CPU-1

流水线架构

数据通路

CPU 的 5 级流水线阶段为：

阶段	简称	功能概述
取指阶段（Fetch）	F	从指令存储器中读取指令
译码阶段（Decode）	D	从寄存器文件中读取源操作数并对指令译码以便得到控制信号
执行阶段（Execute）	E	使用 ALU 执行计算
存储阶段（Memory）	M	读或写数据存储器
写回阶段（Writeback）	W	将结果写回到寄存器文件

梳理一下所有指令的数据通路：（无冒险，仅流水）
`add`：

- F 级的 MUX 选择 0，PC+4；
- D 级的两个 MUX 都选择 0，ID/EX 存入 GPR[rs], GPR[rt]；
- E 级的上面三个 MUX 都选择 0，GPR[rs], GPR[rt] 进入 ALU，进行 0 运算（加法），下方的 MUX 选择 1，EX/MEM 中存入 rd；
- W 级的 MUX 选择 1，将 AO 回写，地址为 rd；

`sub`：

- 与 `add` 完全一致，仅有 ALU 选择不同；
- ALU 选择 1 运算（减法）；

`ori`：

- F 级的 MUX 选择 0，PC+4；
- D 级的第一个 MUX 选择 0，ID/EX 存入 GPR[rs]；
- E 级的第一个 MUX 选择 0，第三个 MUX 选择 1，GPR[rs], IMM 进入 ALU，进行 2 运算（或），下方 MUX 选择 0，EX/MEM 中存入 rt；
- W 级的 MUX 选择 1，将 AO 回写，地址为 rt；

`lw`：

- F 级的 MUX 选择 0，PC+4；
- D 级的第一个 MUX 选择 0，ID/EX 存入 GPR[rs]；

- E 级的第一个 MUX 选择 0，第三个 MUX 选择 1，GPR[rs], IMM 进入 ALU，进行 0 运算，下方 MUX 选择 0，EX/MEM 存入 rt；
- M 级允许 DM 可读不可写，MEM/WB 存入 Memory[GPR[rs]+IMM]；
- W 级 MUX 选择 0，将 Memory 回写，地址为 rt；

`sw`：

- F 级的 MUX 选择 0， PC+4；
- D 级第一个 MUX 选择 0，第二个 MUX 选择 0， GPR[rs], GPR[rt] 存入 ID/EX，GRF 可读不可写；
- E 级的第一个和第二个 MUX 选择 0，第三个 MUX 选择 1，ALU 输入 GPR[rs], IMM，进行 0 运算，将 AO 和 GPR[rt]存入 EX/MEM；
- M 级允许 DM 可写，写入 GPR[rt]，地址为 Memory[GPR[rs]+IMM]；

`beq`：

- F 级 MUX 选择 1，跳转到 D 级计算出的 NPC；
- D 级两个 MUX 选择 0，ID/EX 存入 GPR[rs], GPR[rt], IMM，NPC 模块选择 0 （beq 计算），进行 PC + 4 + IMM<<2 计算；COMP 模块对 GPR[rs], GPR[rt] 进行判等，控制信号输入 NPC 模块，决定是否输出跳转；

`lui`：

- F 级 MUX 选择 0，PC+4；
- D 级 ID/EX 存入 IMM；
- E 级第三个 MUX 选择 1，ALU 输入 IMM，选择 3 运算（立即数加载到最高位），最下方 MUX 选择 0，将 rt 存入 EX/MEM；
- W 级选择 1，将 AO 回写，地址为 rt；

`nop`：

- F 级 PC+4；
- 其他级不进行任何操作，指令向下流水；

`jal`：

- F 级选择 2，进行 jal 跳转；
- D 级 NPC 模块选择 1，进行 jal 计算，即 {PC[31:28], IMM26, 2'b00}；
- E 级最下方 MUX 选择 2，将 31 作为写入地址；
- W 级选择 2， 将 PC+8 回写，地址为 31。

`jr`：

- F 级选择 3，进行 jr 跳转；
- D 级第一个 MUX 选择 0，将 GPR[rs] 作为 NPC；

译码器

各指令 OpCode 及 Func：

指令	op5	op4	op3	op2	op1	op0
add	0	0	0	0	0	0
sub	0	0	0	0	0	0

指令	op5	op4	op3	op2	op1	op0
ori	0	0	1	1	0	1
lui	0	0	1	1	1	1
lw	1	0	0	0	1	1
sw	1	0	1	0	1	1
beq	0	0	0	1	0	0
nop	0	0	0	0	0	0
jal	0	0	0	0	1	1
jr	0	0	0	0	0	0

指令	func5	func4	func3	func2	func1	func0
add	1	0	0	0	0	0
sub	1	0	0	0	1	0
nop	0	0	0	0	0	0
jr	0	0	1	0	0	0

译码方式：
Logisim 中拟采用集中式译码；Verilog 中拟采用分布式译码。
译码风格：
再说。
控制信号：**!!重点!!**

指令	add	sub	ori	lui	lw	sw	beq	jal	jr
WDSel	01	01	01	01	00			10	
RegWrite	1	1	1	1	1			1	
MemWrite						1			
MemRead?					1				
PCSel	000	000	000	000	000	000	001	010	011
ALUCtrl	000	001	010	011	000	000	001		
ALUSrc	0	0	1	1	1	1			
RegDst	01	01	00	00	00			10	

流水线冒险

冒险的种类

结构冒险：不需要考虑
控制冒险：主要是 beq 指令的判断结果影响接下来的指令执行流
数据冒险：指令需要的数据还未进入寄存器堆的情况

冒险的解决

阻塞：两条指令发生数据依赖时，将前一条指令阻塞在 D 级，插入 nop 指令气泡。
提前分支判断：在 D 级进行分支判断，如果发生跳转，F 级指令作废。
延迟槽：只要不考虑 F 级指令的作废问题，就是实现了延迟槽；对于 jal 指令，当指令位于 D 级时，应当向 $31 写入 D_PC+8 或 F_PC+4。
转发：这是对阻塞的进一步优化，例如 add 指令在 E 级就已经计算完成，那么在 M 级就可以将结果转发回去，从而解除 D 级的阻塞状态。可以处理部分数据冒险。

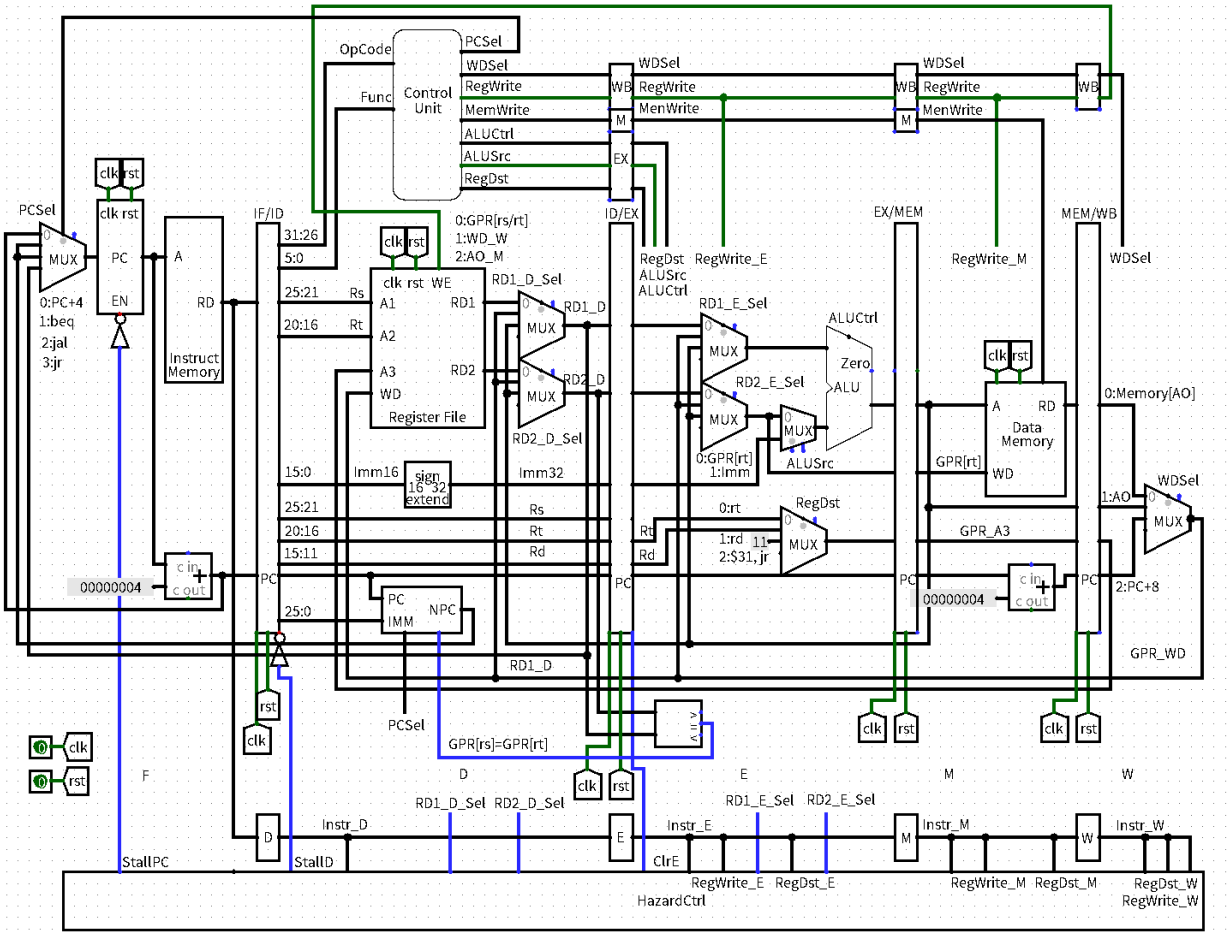
转发的实现

AT模型
T_{use} 表示数据到了 D 级还需要多少个周期要使用，T_{new} 表示数据还有多长时间产生；
对于永远不被使用的数据，设为一个大数（3）.
!!重点看表格!!

指令	rs_T_{use}	rt_T_{use}	E_T_{new}	M_T_{new}	W_T_{new}
add	1	1	1	0	0
sub	1	1	1	0	0
ori	1	3	1	0	0
lui	1	3	1	0	0
lw	1	3	2	1	0
sw	1	1	0	0	0
beq	0	0	0	0	0
nop	3	3	0	0	0
jal	3	3	2	1	0
jr	0	3	0	0	0

当数据冒险时（ A 相等）：
$T_{use} \geq T_{new}$ ，转发；$T_{use} < T_{new}$ ，阻塞。
需要阻塞的情况：

- $E\_T_{new} = 2, T_{use} = 0$
- $E\_T_{new} = 1, T_{use} = 0$
- $M\_T_{new} = 1, T_{use} = 0$
- $E\_T_{new} = 2, T_{use} = 1$
  转发优先级问题：选择流水线靠前的数据。



reg	name	usage
$0	$zero	常量 0
$1	$at	保留给汇编器使用的临时变量
$2-$3	$v0-$v1	函数调用返回值
$4-$7	$a0-$a3	函数调用参数
$8-$15	$t0-$t7	临时变量
$16-$23	$s0-$s7	需要保存的变量
$24-$25	$t8-$t9	临时变量
$26-$27	$k0-$k1	留给操作系统使用
$28	$gp	全局指针
$29	$sp	堆栈指针
$30	$fp	帧指针
$31	$ra	返回地址

源码

constants.v

```
// ALUCtrl
`define aluAdd 3'd0
`define aluSub 3'd1
`define aluOri 3'd2
`define aluLui 3'd3

// PCSel
`define pc4 3'd0
`define pcBeq 3'd1
`define pcJal 3'd2
`define pcJr 3'd3

// WDSel
`define wdDM 2'd0
`define wdAO 2'd1
`define wdPC8 2'd2

// RegDst
`define a3rt 2'd0
`define a3rd 2'd1
`define a3ra 2'd2

// Controller
`define nop 6'b000000
`define add 6'b100000
// more

// Instr
`define OpCode Instr[31:26]
`define Rs Instr[25:21]
`define Rt Instr[20:16]
`define Rd Instr[15:11]
`define IMM16 Instr[15:0]
`define IMM26 Instr[25:0]
`define Func Instr[5:0]

// A-T
/********  add  ********/
`define add_rs_Tuse 2'd1
`define add_rt_Tuse 2'd1
`define add_E_Tnew 2'd1
`define add_M_Tnew 2'd0
`define add_W_Tnew 2'd0
/********  sub  ********/
// more
```

PC.v

```
module PC(
    input clk,
    input reset,
    input EN,
    input [31:0] NPC,
    output reg [31:0] PC
    );

        initial begin
                PC = 32'h0000_3000;
        end

        always @(posedge clk) begin
                if (reset) begin
                        PC <= 32'h0000_3000;
                end
                else if (EN) begin
                        PC <= NPC;
                end
                else begin
                        PC <= PC;
                end
        end

endmodule
```

IM.v

```
module IM(
    input [31:0] PC,
    output [31:0] RD
    );

        wire [31:0] addr;
        reg [31:0] im[0:4095];

        initial begin
                $readmemh("code.txt", im);
        end

        assign addr = PC - 32'h0000_3000;
        assign RD = im[(addr >> 2)];

endmodule
```

Controller.v

```
module Controller(
    input [31:0] Instr,
    output reg [1:0] WDSel,
    output reg RegWrite,
    output reg MemWrite,
    output reg [2:0] PCSel,
    output reg [2:0] ALUCtrl,
    output reg ALUSrc,
    output reg [1:0] RegDst
    );

        always @(*) begin
                case (`OpCode)
                        `nop: begin // OpCode = 6'b000000;
                                case (`Func)
                                        `add: begin
                                                WDSel = `wdAO;
                                                RegWrite = 1;
                                                MemWrite = 0;
                                                PCSel = `pc4;
                                                ALUCtrl = `aluAdd;
                                                ALUSrc = 0;
                                                RegDst = `a3rd;
                                        end
                                        `sub: begin
                                                // more
                                end
                                endcase
                        end
                        `ori: begin
                                WDSel = `wdAO;
                                RegWrite = 1;
                                MemWrite = 0;
                                PCSel = `pc4;
                                ALUCtrl = `aluOri;
                                ALUSrc = 1;
                                RegDst = `a3rt;
                        end
                        `lui: begin
                                // more
                endcase
        end

endmodule
```

GRF.v

```
module GRF(
    input clk,
```

```
    input reset,
    input WE,
    input [4:0] A1,
    input [4:0] A2,
    input [4:0] A3,
    input [31:0] WD,
         input [31:0] PC,
    output [31:0] RD1,
    output [31:0] RD2
    );

        integer i;
        reg [31:0] regs[0:31];

        initial begin
                for (i = 0; i < 32; i = i + 1) begin
                        regs[i] <= 32'h0000_0000;
                end
        end

        always @(posedge clk) begin
                if (reset) begin
                        for (i = 0; i < 32; i = i + 1) begin
                                regs[i] <= 32'h0000_0000;
                        end
                end
                else begin
                        if (WE && A3 != 5'd0) begin
                                $display("%d@%h: $%d <= %h", $time, PC, A3, WD);
                                regs[A3] <= WD;
                        end
                end
        end

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

endmodule
```

NPC.v

```
module NPC(
    input [31:0] PC,
    input [25:0] IMM,
    input [2:0] PCSel,
    input equ,
    output reg [31:0] NPC
```

```
    );

        always @(*) begin
                case (PCSel)
                        `pcBeq: begin
                                if (equ) begin
                                        NPC = PC + ({{16{IMM[15]}}, IMM[15:0]} <<
2);
                                end
                                else begin
                                        NPC = PC + 32'd4;
                                end
                        end
                        `pcJal: begin
                                NPC = {PC[31:28], IMM, 2'd0};
                        end
                endcase
        end

endmodule
```

ALU.v

```
module ALU(
    input [31:0] alu_A,
    input [31:0] alu_B,
    input [2:0] ALUCtrl,
    output reg [31:0] alu_out
    );

        always @(*) begin
                case (ALUCtrl)
                        `aluAdd:
                                alu_out = alu_A + alu_B;
                        `aluSub:
                                alu_out = alu_A - alu_B;
                        `aluOri:
                                alu_out = alu_A | {16'd0, alu_B[15:0]};
                        `aluLui:
                                alu_out = alu_B << 16;
                endcase
        end

endmodule
```

DM.v

```
module DM(
    input clk,
    input reset,
    input WE,
    input [31:0] A,
    input [31:0] WD,
    input [31:0] PC,
    output [31:0] RD
    );

        reg [31:0] dm[0:3071];
        wire [11:0] MemAddr;
        wire [31:0] Addr;
        integer i;

        always @(posedge clk) begin
                if (reset) begin
                        for (i = 0; i < 3072; i = i + 1) begin
                                dm[i] <= 32'h0000_0000;
                        end
                end
                else begin
                        if (WE) begin
                                dm[MemAddr] <= WD;
                                $display("%d@%h: *%h <= %h", $time, PC, A, WD);
                        end
                end
        end

        assign MemAddr = A >> 2;
        assign RD = dm[MemAddr];

endmodule
```

HazardCtrl.v

```
module HazardCtrl(
    input [31:0] Instr_D,
    input [31:0] Instr_E,
    input [31:0] Instr_M,
    input [31:0] Instr_W,
         input RegWrite_E,
         input RegWrite_M,
         input RegWrite_W,
         input [1:0] RegDst_E,
         input [1:0] RegDst_M,
         input [1:0] RegDst_W,
    output StallPC,
```

```
    output StallD,
    output [1:0] RD1_D_Sel,
    output [1:0] RD2_D_Sel,
    output ClrE,
    output [1:0] RD1_E_Sel,
    output [1:0] RD2_E_Sel
    );

        /********** Declarations  **********/
        reg [1:0] rs_Tuse, rt_Tuse, E_Tnew, M_Tnew, W_Tnew;
        wire [4:0] A1_D = Instr_D[25:21], A2_D = Instr_D[20:16];
        wire [4:0] A1_E = Instr_E[25:21], A2_E = Instr_E[20:16];
        wire [4:0] A1_M = Instr_M[25:21], A2_M = Instr_M[20:16];
        wire [4:0] A1_W = Instr_W[25:21], A2_W = Instr_W[20:16];
        wire [4:0] A3_E, A3_M, A3_W;
        assign A3_E = (RegDst_E == 2'd0) ? Instr_E[20:16] :
                                           (RegDst_E == 2'd1) ? Instr_E[15:11] :
5'd31;
        assign A3_M = (RegDst_M == 2'd0) ? Instr_M[20:16] :
                                           (RegDst_M == 2'd1) ? Instr_M[15:11] :
5'd31;
        assign A3_W = (RegDst_W == 2'd0) ? Instr_W[20:16] :
                                           (RegDst_W == 2'd1) ? Instr_W[15:11] :
5'd31;

        // StallCtrl
        wire Stall_RS0_E2, Stall_RS0_E1, Stall_RS0_M1, Stall_RS1_E2;
        wire Stall_RT0_E2, Stall_RT0_E1, Stall_RT0_M1, Stall_RT1_E2;
        wire Stall_RS, Stall_RT, Stall;

        /**********  Stage_D  **********/
        always @(*) begin
                case (Instr_D[31:26])
                        `nop: begin
                                case (Instr_D[5:0])
                                        `add: begin
                                                rs_Tuse = `add_rs_Tuse;
                                                rt_Tuse = `add_rt_Tuse;
                                        end
                                        `sub: begin
                                                // more
                                endcase
                        end
                        `ori: begin
                                rs_Tuse = `ori_rs_Tuse;
                                rt_Tuse = `ori_rt_Tuse;
                        end
                        `lui: begin
                                // more
                endcase
```

```
        end

        /**********  Stage_E  **********/
        always @(*) begin
                case (Instr_E[31:26])
                        `nop: begin
                                case (Instr_E[5:0])
                                        `add: begin
                                                E_Tnew = `add_E_Tnew;
                                        end
                                        `sub: begin
                                                // more
                                endcase
                        end
                        `ori: begin
                                E_Tnew = `ori_E_Tnew;
                        end
                        `lui: begin
                                // more
                endcase
        end

        /**********  Stage_M  **********/
        always @(*) begin
                case (Instr_M[31:26])
                        `nop: begin
                                // more

        /**********  Stage_W  **********/
        // more

        /**********  HazardCtrl  **********/
        // StallCtrl
        assign Stall_RS0_E2 = (rs_Tuse == 2'd0) & (E_Tnew == 2'd2) & (A1_D ==
A3_E) & (A1_D != 5'd0) & RegWrite_E;
        assign Stall_RS0_E1 = (rs_Tuse == 2'd0) & (E_Tnew == 2'd1) & (A1_D ==
A3_E) & (A1_D != 5'd0) & RegWrite_E;
        assign Stall_RS0_M1 = (rs_Tuse == 2'd0) & (M_Tnew == 2'd1) & (A1_D ==
A3_M) & (A1_D != 5'd0) & RegWrite_M;
        assign Stall_RS1_E2 = (rs_Tuse == 2'd1) & (E_Tnew == 2'd2) & (A1_D ==
A3_E) & (A1_D != 5'd0) & RegWrite_E;
        assign Stall_RS = Stall_RS0_E2 | Stall_RS0_E1 | Stall_RS0_M1 |
Stall_RS1_E2;

        assign Stall_RT0_E2 = (rt_Tuse == 2'd0) & (E_Tnew == 2'd2) & (A2_D ==
A3_E) & (A2_D != 5'd0) & RegWrite_E;
        assign Stall_RT0_E1 = (rt_Tuse == 2'd0) & (E_Tnew == 2'd1) & (A2_D ==
A3_E) & (A2_D != 5'd0) & RegWrite_E;
        assign Stall_RT0_M1 = (rt_Tuse == 2'd0) & (M_Tnew == 2'd1) & (A2_D ==
A3_M) & (A2_D != 5'd0) & RegWrite_M;
```

```
        assign Stall_RT1_E2 = (rt_Tuse == 2'd1) & (E_Tnew == 2'd2) & (A2_D ==
A3_E) & (A2_D != 5'd0) & RegWrite_E;
        assign Stall_RT = Stall_RT0_E2 | Stall_RT0_E1 | Stall_RT0_M1 |
Stall_RT1_E2;

        assign Stall = Stall_RS | Stall_RT;
        assign StallPC = Stall;
        assign StallD = Stall;
        assign ClrE = Stall;

        // ForwardCtrl
        assign RD1_D_Sel = (A1_D == A3_M && A1_D != 5'd0 && M_Tnew == 2'd0 &&
RegWrite_M == 1'b1) ? 2'd2 :
                                                        (A1_D == A3_W && A1_D !=
5'd0 && W_Tnew == 2'd0 && RegWrite_W == 1'b1) ? 2'd1 : 2'd0;
        assign RD2_D_Sel = (A2_D == A3_M && A2_D != 5'd0 && M_Tnew == 2'd0 &&
RegWrite_M == 1'b1) ? 2'd2 :
                                                        (A2_D == A3_W && A2_D !=
5'd0 && W_Tnew == 2'd0 && RegWrite_W == 1'b1) ? 2'd1 : 2'd0;
        assign RD1_E_Sel = (A1_E == A3_M && A1_E != 5'd0 && M_Tnew == 2'd0 &&
RegWrite_M == 1'b1) ? 2'd2 :
                                                        (A1_E == A3_W && A1_E !=
5'd0 && W_Tnew == 2'd0 && RegWrite_W == 1'b1) ? 2'd1 : 2'd0;
        assign RD2_E_Sel = (A2_E == A3_M && A2_E != 5'd0 && M_Tnew == 2'd0 &&
RegWrite_M == 1'b1) ? 2'd2 :
                                                        (A2_E == A3_W && A2_E !=
5'd0 && W_Tnew == 2'd0 && RegWrite_W == 1'b1) ? 2'd1 : 2'd0;
endmodule
```

mips.v

```
module mips(
    input clk,
    input reset
    );

/******** Declarations *********/
// F
wire StallPC;
wire [31:0] Instr_F;
wire [31:0] PC_F;
wire [31:0] PC;
// D
wire StallD;
// more
/**********  Stage_F  **********/
PC F_PC(
```

```
    .clk(clk),
    .reset(reset),
    .EN(~StallPC),
    .NPC(PC),
    .PC(PC_F)
    );

IM F_IM(
    .PC(PC_F),
    .RD(Instr_F)
    );

assign PC = (PCSel_D == 3'd0) ? PC_F + 32'd4 :
                                (PCSel_D == 3'd1 || PCSel_D == 3'd2) ? NPC_D :
                                (PCSel_D == 3'd3) ? RD1_D : 0;

/**********  Stage_D  **********/
always @(posedge clk) begin
        if (reset) begin
                Instr_D <= 32'd0;
                PC4_D <= 32'd0;
        end
        else if (StallD) begin
                Instr_D <= Instr_D;
                PC4_D <= PC4_D;
        end
        else begin
                Instr_D <= Instr_F;
                PC4_D <= PC_F + 32'd4;
        end
end

Controller D_Controller(
    .Instr(Instr_D),
    .PCSel(PCSel_D)
    );

GRF D_GRF(
    .clk(clk),
    .reset(reset),
    .WE(RegWrite_W),
    .A1(Instr_D[25:21]),
    .A2(Instr_D[20:16]),
    .A3(GPR_A3_W),
    .WD(GPR_WD_W),
         .PC(PC_D),
    .RD1(GPR_rs),
    .RD2(GPR_rt)
    );
NPC D_NPC(
```

```
    .PC(PC4_D),
    .IMM(Instr_D[25:0]),
    .PCSel(PCSel_D),
    .equ(equ_D),
    .NPC(NPC_D)
    );
assign RD1_D = (RD1_D_Sel == 2'd0) ? GPR_rs :
                                     (RD1_D_Sel == 2'd1) ? GPR_WD_W :
                                     (RD1_D_Sel == 2'd2) ? AO_M : 0;
assign RD2_D = (RD2_D_Sel == 2'd0) ? GPR_rt :
                                     (RD2_D_Sel == 2'd1) ? GPR_WD_W :
                                     (RD2_D_Sel == 2'd2) ? AO_M : 0;
assign equ_D = (RD1_D == RD2_D);
assign PC_D = PC8_W - 32'd8;

/**********  Stage_E  **********/
always @(posedge clk) begin
        if (reset) begin
                Instr_E <= 32'd0;
                PC4_E <= 32'd0;
                RD1_E <= 32'd0;
                RD2_E <= 32'd0;
        end
        else if (ClrE) begin
                Instr_E <= 32'd0;
        end
        else begin
                Instr_E <= Instr_D;
                PC4_E <= PC4_D;
                RD1_E <= RD1_D;
                RD2_E <= RD2_D;
        end
end

Controller E_Controller(
    .Instr(Instr_E),
         .RegWrite(RegWrite_E),
    .ALUCtrl(ALUCtrl_E),
    .ALUSrc(ALUSrc_E),
    .RegDst(RegDst_E)
    );

assign Op1_E = (RD1_E_Sel == 2'd0) ? RD1_E :
                                     (RD1_E_Sel == 2'd1) ? GPR_WD_W :
                                     (RD1_E_Sel == 2'd2) ? AO_M : 0;
assign Op2_E = (ALUSrc_E == 1'd1) ? {{20{Instr_E[15]}}, Instr_E[15:0]} :
                                     (RD2_E_Sel == 2'd0) ? RD2_E :
                                     (RD2_E_Sel == 2'd1) ? GPR_WD_W :
                                     (RD2_E_Sel == 2'd2) ? AO_M : 0;
assign RD2_to_M = (RD2_E_Sel == 2'd0) ? RD2_E :
```

```
                                        (RD2_E_Sel == 2'd1) ? GPR_WD_W :
                                        (RD2_E_Sel == 2'd2) ? AO_M : 0;
ALU E_ALU(
    .alu_A(Op1_E),
    .alu_B(Op2_E),
    .ALUCtrl(ALUCtrl_E),
    .alu_out(AO_E)
    );

assign GPR_A3_E = (RegDst_E == 2'd0) ? Instr_E[20:16] :
                                                (RegDst_E == 2'd1) ?
Instr_E[15:11] :
                                                (RegDst_E == 2'd2) ? 5'd31 : 0;

/**********  Stage_M  **********/
always @(posedge clk) begin
        if (reset) begin
                Instr_M <= 32'd0;
                PC4_M <= 32'd0;
                AO_M <= 32'd0;
                RD2_M <= 32'd0;
                GPR_A3_M <= 5'd0;
        end
        else begin
                Instr_M <= Instr_E;
                PC4_M <= PC4_E;
                AO_M <= AO_E;
                RD2_M <= RD2_to_M;
                GPR_A3_M <= GPR_A3_E;
        end
end

Controller M_Controller(
    .Instr(Instr_M),
         .RegWrite(RegWrite_M),
    .MemWrite(MemWrite_M),
    .PCSel(PCSel_M),
         .RegDst(RegDst_M)
    );

DM M_DM(
    .clk(clk),
    .reset(reset),
    .WE(MemWrite_M),
    .A(AO_M),
    .WD(RD2_M),
    .PC(PC_M),
    .RD(DO_M)
    );
assign PC_M = PC4_M - 32'd4;
```

```
/**********  Stage_W  **********/
always @(posedge clk) begin
        if (reset) begin
                Instr_W <= 32'd0;
                PC8_W <= 32'd0;
                AO_W <= 32'd0;
                DO_W <= 32'd0;
                GPR_A3_W <= 5'd0;
        end
        else begin
                Instr_W <= Instr_M;
                PC8_W <= PC4_M + 32'd4;
                AO_W <= AO_M;
                DO_W <= DO_M;
                GPR_A3_W <= GPR_A3_M;
        end
end

Controller W_Controller(
    .Instr(Instr_W),
    .WDSel(WDSel_W),
    .RegWrite(RegWrite_W),
         .RegDst(RegDst_W)
    );
assign GPR_WD_W = (WDSel_W == 2'd0) ? DO_W :
                                                (WDSel_W == 2'd1) ? AO_W :
                                                (WDSel_W == 2'd2) ? PC8_W : 0;

/********** HazardCtrl **********/
HazardCtrl HazardCtrl(
    .Instr_D(Instr_D),
    // more
    );

endmodule
```

思考题

> 1. 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

提前分支判断，意味着当前面的指令产生数据依赖时，需要阻塞更多时间。例如：

```
add $s0, $t1, $t2
sub $s1, $t3, $t4
beq $s0, $s1, lable
```

如果提前分支判断，当 beq 处于 D 级时，add 的计算结果 $s0 可以转发回来，而 sub 的计算结果无法转发，需要阻塞一周期；如果不提前分支判断，当 beq 处于 E 级时，add 和 sub 都可以转发回来。
因此，提前分支判断的优势是当 beq 条件成立时，需要作废的指令只有一条（延迟槽）；不提前分支判断，需要作废两条指令。

> 2. 因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

因为 jal 之后紧跟的指令是延迟槽，而延迟槽内的指令不作废，如果是 PC+4 ，则会再次跳转到延迟槽的位置，再一次执行该指令。

> 3. 我们要求所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），请思考为什么？

这样便于不同流水级的管理和统一，避免出错。

> 5. 我们为什么要使用 GPR 内部转发？该如何实现？

内部转发可以提高运行效率，减少阻塞的情况。在 GRF 的输出端进行转发即可。

> 7. 我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

在上述分析已提到。

> 8. 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

- 跳转型
  修改 equ 的判断即取值可能性，或者直接新定义一个信号，其实只需要给 NPC 模块增加一个 input 并不是很麻烦，这样不容易出错。
- 访存型
  这个感觉比较好加，类比 lw sw 指令。
- 计算型
  对 ALU 进行一些更改即可啦，注意 Tuse 和 Tnew 的设置，类比一下已有指令。

清空延迟槽：关键其实在如何清空延迟槽， CMP 如果时 bonall 指令并且不跳转，则生成一个 flush 信号，如果 flush 为 1 并且没有处于阻塞状态，则 flush 一下 FD 级寄存器即可（想一想：为什么可以直接清空 FD 级寄存器，为什么不能处于阻塞状态）

> 9. 简要描述你的译码器架构，并思考该架构的优势以及不足。

分布式译码，指令驱动型，便于添加指令。缺点是代码过长。
上机后发现并不是很便于添加指令……

测试数据

```
.text
ori $s0, $s0, 5     # 自调用是否有问题
ori $s1, $zero, 4       #
add $t0, $s0, $s1       # R型指令的两种转发是否有问题
lui $t1, 65535          # 与接下来的sub指令一起判断高位减法是否有问题
sw $s1, 0($s1)          # 判断line 3的指令写入GRF是否符合要求且无冲突
add $zero, $s1, $t0     # 给$0复制并观察是否转发数据为0
sub $t1, $zero, $t1     # 使用$0观察转发数据以及检查高位减法

lw $t1, 0($s1)          # 检查寄存器写入是否覆盖
sw $t1, 0($s1)          # 检查lw-sw的转发或阻塞是否实现

lw $t2, 0($s1)
sub $t2, $s0, $s1
add $t4, $s0, $t2       # 以上三行检查R型指令转发的优先级是否正确

sub $s3, $s0, $s1
ori $s4, $zero, 48
ori $s2, $zero, 12344   # 以上三行纯属为了凑数据()
lw $ra, 0($s1)
sub $ra, $ra, $s1
jal tag1
nop                     #添加延时槽

tag1:
beq $ra, $s2, end1      # 以上六行反复对$ra更改以检查beq转发的优先级是否正确
nop
sw $ra, 0($s1)
add $ra, $ra, $s1
end1:
sub $s0, $s0, $s3
beq $s0, $zero, end2    # 检查R型指令是否会对beq指令造成影响[笔者就在这里错了]
nop
jr $ra                  # 反复更改$ra后观察是否跳转到正确的pc值
nop

end2:
jal tag2
nop
tag2:
lw $ra, 0($s1)
add $ra, $ra, $s4
jr $ra                  #以上六行检查jr指令转发的优先级是否正确
nop
lui $zero, 1111         # 最后jr $ra指令会跳转到该行
```

机器码：

```
36100005
34110004
02114020
3c09ffff
ae310000
02280020
00094822
8e290000
ae290000
8e2a0000
02115022
020a6020
02119822
34140030
34123038
8e3f0000
03f1f822
0c000c13
00000000
13f20003
00000000
ae3f0000
03f1f820
02138022
12000003
00000000
03e00008
00000000
0c000c1e
00000000
8e3f0000
03f4f820
03e00008
00000000
3c000457
```

输出：

```
@00003000: $16 <= 00000005
@00003004: $17 <= 00000004
@00003008: $ 8 <= 00000009
@0000300c: $ 9 <= ffff0000
@00003010: *00000004 <= 00000004
@00003018: $ 9 <= 00010000
@0000301c: $ 9 <= 00000004
```

```
@00003020: *00000004 <= 00000004
@00003024: $10 <= 00000004
@00003028: $10 <= 00000001
@0000302c: $12 <= 00000006
@00003030: $19 <= 00000001
@00003034: $20 <= 00000030
@00003038: $18 <= 00003038
@0000303c: $31 <= 00000004
@00003040: $31 <= 00000000
@00003044: $31 <= 0000304c
@00003054: *00000004 <= 0000304c
@00003058: $31 <= 00003050
@0000305c: $16 <= 00000004
@00003054: *00000004 <= 00003050
@00003058: $31 <= 00003054
@0000305c: $16 <= 00000003
@00003054: *00000004 <= 00003054
@00003058: $31 <= 00003058
@0000305c: $16 <= 00000002
@00003058: $31 <= 0000305c
@0000305c: $16 <= 00000001
@0000305c: $16 <= 00000000
@00003070: $31 <= 00003078
@00003078: $31 <= 00003054
@0000307c: $31 <= 00003084
```

二

```
ori $t1, $zero, 0
ori $t2, $zero, 10
ori $t0, $zero, 1
ori $t4, $zero, 0
ori $t5, $zero, 4
ori $a0, $zero, 0
ori $a1, $zero, 10
for_i_begin:
beq $t1, $t2, for_i_end
nop
add $t1, $t1, $t0
sw $t4, 0($t4)
add $t4, $t4, $t5
jal for_i_begin
nop
beq $a0, $a1, true_end
nop
```

```
add $a0, $a0, $t0
sw $t4, 0($t4)
add $t4, $t4, $t5
for_i_end:
jr $ra
nop
true_end:
```

机器码：

```
34090000
340a000a
34080001
340c0000
340d0004
34040000
3405000a
112a000b
00000000
01284820
ad8c0000
018d6020
0c000c07
00000000
10850006
00000000
00882020
ad8c0000
018d6020
03e00008
00000000
```

输出：

```
 55@00003000: $ 9 <= 00000000
 65@00003004: $10 <= 0000000a
 75@00003008: $ 8 <= 00000001
 85@0000300c: $12 <= 00000000
 95@00003010: $13 <= 00000004
105@00003014: $ 4 <= 00000000
115@00003018: $ 5 <= 0000000a
145@00003024: $ 9 <= 00000001
145@00003028: *00000000 <= 00000000
165@0000302c: $12 <= 00000004
175@00003030: $31 <= 00003038
215@00003024: $ 9 <= 00000002
```

```
 215@00003028: *00000004 <= 00000004
 235@0000302c: $12 <= 00000008
 245@00003030: $31 <= 00003038
 285@00003024: $ 9 <= 00000003
 285@00003028: *00000008 <= 00000008
 305@0000302c: $12 <= 0000000c
 315@00003030: $31 <= 00003038
 355@00003024: $ 9 <= 00000004
 355@00003028: *0000000c <= 0000000c
 375@0000302c: $12 <= 00000010
 385@00003030: $31 <= 00003038
 425@00003024: $ 9 <= 00000005
 425@00003028: *00000010 <= 00000010
 445@0000302c: $12 <= 00000014
 455@00003030: $31 <= 00003038
 495@00003024: $ 9 <= 00000006
 495@00003028: *00000014 <= 00000014
 515@0000302c: $12 <= 00000018
 525@00003030: $31 <= 00003038
 565@00003024: $ 9 <= 00000007
 565@00003028: *00000018 <= 00000018
 585@0000302c: $12 <= 0000001c
 595@00003030: $31 <= 00003038
 635@00003024: $ 9 <= 00000008
 635@00003028: *0000001c <= 0000001c
 655@0000302c: $12 <= 00000020
 665@00003030: $31 <= 00003038
 705@00003024: $ 9 <= 00000009
 705@00003028: *00000020 <= 00000020
 725@0000302c: $12 <= 00000024
 735@00003030: $31 <= 00003038
 775@00003024: $ 9 <= 0000000a
 775@00003028: *00000024 <= 00000024
 795@0000302c: $12 <= 00000028
 805@00003030: $31 <= 00003038
 885@00003040: $ 4 <= 00000001
 885@00003044: *00000028 <= 00000028
 905@00003048: $12 <= 0000002c
 955@00003040: $ 4 <= 00000002
 955@00003044: *0000002c <= 0000002c
 975@00003048: $12 <= 00000030
1025@00003040: $ 4 <= 00000003
1025@00003044: *00000030 <= 00000030
```

```
1045@00003048: $12 <= 00000034
1095@00003040: $ 4 <= 00000004
1095@00003044: *00000034 <= 00000034
1115@00003048: $12 <= 00000038
1165@00003040: $ 4 <= 00000005
1165@00003044: *00000038 <= 00000038
1185@00003048: $12 <= 0000003c
1235@00003040: $ 4 <= 00000006
1235@00003044: *0000003c <= 0000003c
1255@00003048: $12 <= 00000040
1305@00003040: $ 4 <= 00000007
1305@00003044: *00000040 <= 00000040
1325@00003048: $12 <= 00000044
1375@00003040: $ 4 <= 00000008
1375@00003044: *00000044 <= 00000044
1395@00003048: $12 <= 00000048
1445@00003040: $ 4 <= 00000009
1445@00003044: *00000048 <= 00000048
1465@00003048: $12 <= 0000004c
1515@00003040: $ 4 <= 0000000a
1515@00003044: *0000004c <= 0000004c
1535@00003048: $12 <= 00000050
```