计算机组成原理　P5 实验报告

一、CPU 设计方案综述

本 CPU 为 Verilog 实现的流水线 MIPS32 - CPU，支持的指令集包含 MIPS-lite2 **{addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop}**。,采用模块化和层次化设计。顶层有效的驱动信号仅有内置的 clk 时钟信号和同步复位 reset 信号。
模块定义

二．模块接口

1.PC_IM

信号名	方向	描述
clk	I	时间信号
reset	I	复位信号
WE	I	使能
npc	I	下一周期待执行指令的 32 位地址
pc	O	本周起待执行指令的 32 位地址
Instr	O	当前执行的 32 位机器码指令

2. REG_D (IF-ID 寄存器)

文件	模块接口定义
REG_D.v	module REG_D(input clk, input reset, input en, input [31:0] Instr, input [31:0] pc, output reg[31:0] Instr_out, output reg[31:0] pc_out);

3.GRF

信号名	方向	描述
pc	I	当前执行的机器码指令
clk	I	时钟信号
reset	I	异步复位信号，将 32 个寄存器中的值全部清零 1：复位　0：无效

		写使能信号
WE	I	1：可向 GRF 中写入数据　　0：不可向 GRF 中写入数据
a1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
a2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
a3	I	5 位地址输入信号，指定 32 个寄存器中的一个，作为写入的目标寄存器
WD	I	32 位数据输入信号
RD1	O	输出 a1 指定的寄存器中的 32 位数据
RD2	O	输出 a2 指定的寄存器中的 32 位数据

4. CMP

信号名	方向	功能描述
rsRD1[31:0]	I	Rs 指向寄存器的值
rtRD2[31:0]	I	Rt 指向寄存器的值
B_type[2:0]	I	B 跳转指令的类型
B_jump	O	是否执行 B 跳转指令

5. EXT

信号名	方向	描述
EXTin	I	J 型指令 25-0 位中存储的 26 位立即数
		1 位控制信号，控制 EXT 对 16 位立即数的位扩展方式
EXTop	O	0：零扩展　　1：符号扩展
EXTout	O	32 位的立即数的扩展结果

6. NPC

信号名	方向	功能描述
D_PC	I	D 级 PC
F_PC	I	F 级 PC
I26	I	26 位立即数
MFRSD	I	转发 PC 的 MUX 结果（jr jalr 需要转发）
Br	I	选择下一条指令地址的类型
B_jump	I	判断 BRANCH 类指令条件是否成立
Npc	O	更新的 pc 值

7. E_REG(ID/EX)

文件	模块接口定义
E_REG.v	module E_REG (input clk, input reset, input en, input [31:0] Instr, input [31:0] PC,

	input [31:0] RF_RD1, input [31:0] RF_RD2, input [31:0] EXT, output reg[31:0] Instr_out, output reg[31:0] PC_out, output reg[31:0] RS_E, output reg[31:0] RT_E, output reg[31:0] EXT_E);

8.ALU

信号名	方向	功能描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
ALUCtrl[2:0]	I	控制信号 000：与 001：或 010：加 011：减 100：移位
Shamt[4:0]	I	位移量
ALUout	O	ALU 计算结果

9.M_REG（EX/MEM）

文件	模块接口定义
M_REG.v	module M_REG (input clk, input reset, input en, input [31:0] Instr, input [31:0] PC, input [31:0] ALU, input [31:0] RT_E, output reg[31:0] Instr_out, output reg[31:0] PC_out, output reg[31:0] ALU_out, output reg[31:0] RT_M);

10.DM

信号名	方向	功能描述
Addr[31:0]	I	ALUout,写入地址

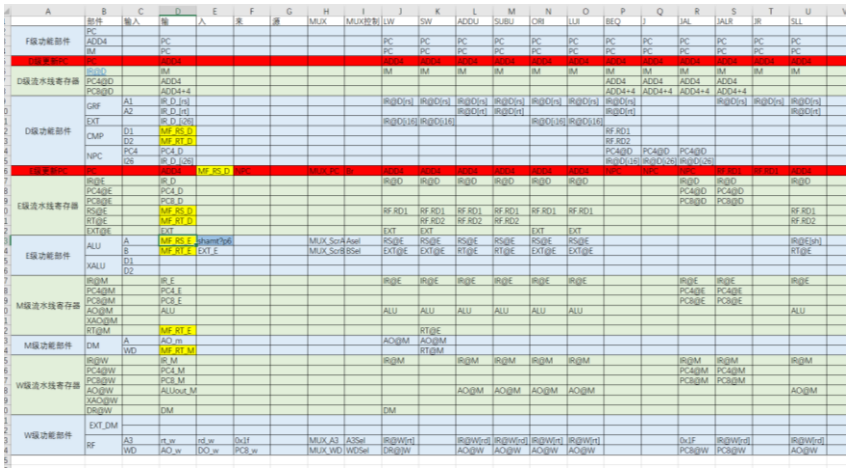
Bit_type[2:0]	I	Store/load 的类型
Newdata	I	RD2
PC[31:0]	I	当前 PC
WE	I	使能
Reset	I	复位信号 1：复位 0：无效
Clk	I	时钟信号
Out_DM[31:0]	O	加载内存的结果

11.W_REG(MEM/WB)

文件	模块接口定义
W_REG	module W_REG (input clk, input reset, input en, input [31:0] Instr, input [31:0] PC, input [31:0] AO_M, input [31:0] DM, output reg[31:0] Instr_out, output reg[31:0] PC_W, output reg[31:0] AO_W, output reg[31:0] DR_W);

三．控制器设计(CU)

含有转发数据的数据通路如表格



CU （采取分布式译码）

```
module CU (
    input [31:0] Instr,

    // meaningful parts of instructions
    output [4:0] rs,
    output [4:0] rt,
    output [4:0] rd,
    output [4:0] shamt,
    output [15:0] imm16,
    output [25:0] imm26,

    // controlling
    output [2:0] Br,
    output [3:0] b_type,

    output EXTop,

    output [3:0] ALUControl,
    output BSel,

    output [2:0] DMtype,
    output DMWr,
```

```
    output [2:0] A3Sel,
    output [2:0] WDSel,
    output RFWr,

    output [4:0] A3,

    //type
    output BRANCH,
    output LOAD,
    output STORE,
    output CAL_R,
    output CAL_I,
    output j_26,
    output j_reg,
    output j_link
```

具体控制信号的生成，采取先对指令进行指令类型判断，再根据指令类型得到控制型号取值的方法。具体实现见 CU.v。

在流水线阶段，根据使用需要，将 CU 实例化四次。

四. 冒险处理单元设计(FWD,STALL)————使用**需求时间——供给时间模型**。

1．Stall 暂停信号的生成

```
module STALL (
    input [31:0] D_Instr,
    input [31:0] E_Instr,
    input [31:0] M_Instr,
    output stall
);
```

通过 D、E、M 级流水线寄存器的指令类型来判断 stall 信号，从而完成流水线的暂停（即寄存器使能置零和值清空）

IF/ID阶段 Tuse分析		
指令类型	寄存器	Tuse
BRANCH	rs/rt	0
LOAD	rs	1
STORE	rs	1
	rt	2
jr	rs	0
jalr	rs	0
cal_r	rs/rt	1
cal_i	rs	1

Tnew														
ID/EX					EX/MEM					MEM/WB				
cal_r	cal_i	LOAD	jal	jalr	cal_r	cal_i	LOAD	jal	jalr	cal_r	cal_i	LOAD	jal	jalr
rd-1	rt-1	rt-2	31-0	rd-0	rd-0	rt-0	rt-1	31-0	rd-0	rd-0	rt-0	rt-0	31-0	rd-0

			Tnew			
IF/ID阶段 Tuse分析			ID/EX			EX/MEM
指令类型	寄存器	Tuse	cal_r	cal_i	LOAD	LOAD
			rd-1	rt-1	rt-2	rt-1
BRANCH	rs/rt	0	S	S	S	S
LOAD	rs	1			S	
STORE	rs	1			S	
	rt	2				
jr	rs	0	S	S	S	S
jalr	rs	0	S	S	S	S
cal_r	rs/rt	1			S	
cal_i	rs	1			S	

```
assign  Tuse_rs = (D_BRANCH | D_j_reg)? 3'd0
                : (D_LOAD | D_STORE | D_CAL_I |D_CAL_R) ?  3'd1
                : 3'd2 ;
assign  Tuse_rt = (D_BRANCH )? 3'd0
                : (D_CAL_R) ?  3'd1
                : (D_STORE) ?  3'd2
                : 3'd3 ;
```

```
wire [2:0] Tnew_E;          // 冲突并不在意写入的是rs、rt，只在意具体
assign Tnew_E = (E_CAL_I | E_CAL_R) ? 3'd1
              : ( E_LOAD ) ?  3'd2
              : 3'd0 ;
wire stall_E_rs = ( Tuse_rs<Tnew_E ) && (D_rs == E_A3) && E_WE ;
wire stall_E_rt = ( Tuse_rt<Tnew_E ) && (D_rt == E_A3) && E_WE ;
```

```
wire [2:0] Tnew_M;          // 冲突并不在意写入的是rs、rt，只在意具体
assign Tnew_M =  ( M_LOAD ) ?  3'd1 : 3'd0 ;
wire stall_M_rs = ( Tuse_rs<Tnew_M ) && (D_rs == M_A3) && M_WE ;
wire stall_M_rt = ( Tuse_rt<Tnew_M ) && (D_rt == M_A3) && M_WE ;
```

```
assign stall = stall_E_rs | stall_E_rt | stall_M_rs | stall_M_rt ;
```

总的暂停信号通过各级 rs、rt 处发生的暂停信号进行按位与后得到。

2．转发 MUX 的选择信号的生成

```
module FWD (
    input [31:0] D_Instr,
    input [31:0] E_Instr,
    input [31:0] M_Instr,
    input [31:0] W_Instr,

    output [3:0] F_RS_D_Sel,
    output [3:0] F_RT_D_Sel,

    output [3:0] F_RS_E_Sel,
    output [3:0] F_RT_E_Sel,

    output [3:0] F_RT_M_Sel
);
```

FWD.v 用于生成转发 MUX 的选择信号，在 mips.v 中实例化后使用

						ID/EX		EX/MEM				MEM/WB				
						jal	jalr	cal_r	cal_i	jal	jalr	cal_r	cal_i	load	jal	jalr
流水线级别	寄存器	涉及指令	MUX	控制信号	无转发值	31	rd	rd	rt	31	rd	rd	rt	rt	31	rd
D	rs	cal_r/cal_i/load/branch/store/j_reg	MF_RS_D	F_RS_D_Sel	D_RS	PC8_E	PC8_E	ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
D	rt	cal_r/branch/store	MF_RT_D	F_RT_D_Sel	D_RT	PC8_E	PC8_E	ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
E	rs	cal_r/cal_i/load/store	MF_RS_E	F_RS_E_Sel	E_RS			ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
E	rt	cal_r/store	MF_RT_E	F_RT_E_Sel	E_RT			ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
M	rt	store	MF_RT_M	F_RT_M_Sel	M_RT							MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
			优先级——	D	0	3	3	1	1	4	4	2	2	2	5	5
				E	0			1	1	3	3	2	2	2	4	4
				M	0							1	1	1	2	2

```
wire F_RS_D_instructions = D_BRANCH | D_LOAD | D_STORE | D_CAL_R | D_CAL_I | D_j_reg ;
assign  F_RS_D_Sel = (  (F_RS_D_instructions==1'b1) && (E_j_link==1'b1) && (D_rs == E_A3) && (D_rs != 5'd0) ) ?  4'd3  // E---D
                   : (  (F_RS_D_instructions==1'b1) && (M_CAL_R==1'b1) && (D_rs == M_A3) && (D_rs != 5'd0) )  ?  4'd1  // M---D
                   : (  (F_RS_D_instructions==1'b1) && (M_CAL_I==1'b1) && (D_rs == M_A3) && (D_rs != 5'd0) )  ?  4'd1
                   : (  (F_RS_D_instructions==1'b1) && (M_j_link==1'b1) && (D_rs == M_A3) && (D_rs != 5'd0) )  ?  4'd4
                   : (  (F_RS_D_instructions==1'b1) && (W_LOAD==1'b1) && (D_rs == W_A3) && (D_rs != 5'd0) )  ?  4'd2   // W---D
                   : (  (F_RS_D_instructions==1'b1) && (W_CAL_R==1'b1) && (D_rs == W_A3) && (D_rs != 5'd0) )  ?  4'd2
                   : (  (F_RS_D_instructions==1'b1) && (W_CAL_I==1'b1) && (D_rs == W_A3) && (D_rs != 5'd0) )  ?  4'd2
                   : (  (F_RS_D_instructions==1'b1) && (W_j_link==1'b1) && (D_rs == W_A3) && (D_rs != 5'd0) )  ?  4'd5
                   : 4'd0  ;

wire F_RT_D_instructions = D_BRANCH | D_STORE | D_CAL_R ;
assign  F_RT_D_Sel = (  (F_RT_D_instructions==1'b1) && (E_j_link==1'b1) && (D_rt == E_A3) && (D_rt != 5'd0) ) ?  4'd3  // E---D
                   : (  (F_RT_D_instructions==1'b1) && (M_CAL_R==1'b1) && (D_rt == M_A3) && (D_rt != 5'd0) )  ?  4'd1  // M---D
                   : (  (F_RT_D_instructions==1'b1) && (M_CAL_I==1'b1) && (D_rt == M_A3) && (D_rt != 5'd0) )  ?  4'd1
                   : (  (F_RT_D_instructions==1'b1) && (M_j_link==1'b1) && (D_rt == M_A3) && (D_rt != 5'd0) )  ?  4'd4
                   : (  (F_RT_D_instructions==1'b1) && (W_LOAD==1'b1) && (D_rt == W_A3) && (D_rt != 5'd0) )  ?  4'd2   // W---D
                   : (  (F_RT_D_instructions==1'b1) && (W_CAL_R==1'b1) && (D_rt == W_A3) && (D_rt != 5'd0) )  ?  4'd2
                   : (  (F_RT_D_instructions==1'b1) && (W_CAL_I==1'b1) && (D_rt == W_A3) && (D_rt != 5'd0) )  ?  4'd2
                   : (  (F_RT_D_instructions==1'b1) && (W_j_link==1'b1) && (D_rt == W_A3) && (D_rt != 5'd0) )  ?  4'd5
                   : 4'd0  ;
```

```
wire F_RS_E_instructions =  E_LOAD | E_STORE | E_CAL_R | E_CAL_I  ;
assign  F_RS_E_Sel = (  (F_RS_E_instructions==1'b1) && (M_CAL_R==1'b1) && (E_rs == M_A3) && (E_rs != 5'd0) )  ?  4'd1  // M---E
                   : (  (F_RS_E_instructions==1'b1) && (M_CAL_I==1'b1) && (E_rs == M_A3) && (E_rs != 5'd0) )  ?  4'd1
                   : (  (F_RS_E_instructions==1'b1) && (M_j_link==1'b1) && (E_rs == M_A3) && (E_rs != 5'd0) )  ?  4'd3
                   : (  (F_RS_E_instructions==1'b1) && (W_LOAD==1'b1) && (E_rs == W_A3) && (E_rs != 5'd0) )  ?  4'd2   // W---E
                   : (  (F_RS_E_instructions==1'b1) && (W_CAL_R==1'b1) && (E_rs == W_A3) && (E_rs != 5'd0) )  ?  4'd2
                   : (  (F_RS_E_instructions==1'b1) && (W_CAL_I==1'b1) && (E_rs == W_A3) && (E_rs != 5'd0) )  ?  4'd2
                   : (  (F_RS_E_instructions==1'b1) && (W_j_link==1'b1) && (E_rs == W_A3) && (E_rs != 5'd0) )  ?  4'd4
                   : 4'd0  ;

wire F_RT_E_instructions =  E_STORE | E_CAL_R  ;
assign  F_RT_E_Sel = (  (F_RT_E_instructions==1'b1) && (M_CAL_R==1'b1) && (E_rt == M_A3) && (E_rt != 5'd0) )  ?  4'd1  // M---E
                   : (  (F_RT_E_instructions==1'b1) && (M_CAL_I==1'b1) && (E_rt == M_A3) && (E_rt != 5'd0) )  ?  4'd1
                   : (  (F_RT_E_instructions==1'b1) && (M_j_link==1'b1) && (E_rt == M_A3) && (E_rt != 5'd0) )  ?  4'd3
                   : (  (F_RT_E_instructions==1'b1) && (W_LOAD==1'b1) && (E_rt == W_A3) && (E_rt != 5'd0) )  ?  4'd2   // W---E
                   : (  (F_RT_E_instructions==1'b1) && (W_CAL_R==1'b1) && (E_rt == W_A3) && (E_rt != 5'd0) )  ?  4'd2
                   : (  (F_RT_E_instructions==1'b1) && (W_CAL_I==1'b1) && (E_rt == W_A3) && (E_rt != 5'd0) )  ?  4'd2
                   : (  (F_RT_E_instructions==1'b1) && (W_j_link==1'b1) && (E_rt == W_A3) && (E_rt != 5'd0) )  ?  4'd4
                   : 4'd0  ;

wire F_RT_M_instructions =  M_STORE  ;
assign  F_RT_M_Sel = (  (F_RT_M_instructions==1'b1) && (W_LOAD==1'b1) && (M_rt == W_A3) && (M_rt != 5'd0) )  ?  4'd1   // W---M
                   : (  (F_RT_M_instructions==1'b1) && (W_CAL_R==1'b1) && (M_rt == W_A3) && (M_rt != 5'd0) )  ?  4'd1
                   : (  (F_RT_M_instructions==1'b1) && (W_CAL_I==1'b1) && (M_rt == W_A3) && (M_rt != 5'd0) )  ?  4'd1
                   : (  (F_RT_M_instructions==1'b1) && (W_j_link==1'b1) && (M_rt == W_A3) && (M_rt != 5'd0) )  ?  4'd2
```

五．测试程序

(1)转发机制覆盖测试

一．D级rs
1.D级rs与E级j_link(jal/jalr)
2.D级rs与M级cal_r
3.D级rs与M级cal_i
4.D级rs与M级jal
5.D级rs与W级cal_r
6.D级rs与W级cal_i
7.D级rs与W级load rt
8.D级rs与W级jal
每一项E级rs指令又分为：cal_r,cal_i,load,store,beq（branch）,jr,jalr
二．D级Rt
1.D级rt与E级j_link(jal/jalr)
2.D级rt与M级cal_r
3.D级rt与M级cal_i
4.D级rt与M级jal
5.D级rt与W级cal_r
6.D级rt与W级cal_i
7.D级rt与W级load rt
8.D级rt与W级jal
每一项又分为：cal_r,st,beq
三．E级Rs
1.E级Rs与M级cal_r
2.E级Rs与M级cal_i
3.E级Rs与M级jal
4.E级Rs与W级cal_r
5.E级Rs与W级cal_i
6.E级Rs与W级load
7.E级Rs与W级jal
每一项又分为：cal_r,cal_i,load,store
四．E级Rt
1.E级Rt与M级cal_r
2.E级Rt与M级cal_i
3.E级Rt与M级jal
4.E级Rt与W级cal_r
5.E级Rt与W级cal_i
6.E级Rt与W级load
7.E级Rt与W级jal

每一项又分为：cal_r,store
五. M级Rt
1.M级Rt与W级cal_r
2.M级Rt与W级cal_i
3.M级Rt与W级load
4.M级Rt与W级jal
每一项又分为：store

(2)暂停机制覆盖测试(按照指令来取的覆盖性测试)

测试目录：

一. Beq_rs/rt
 (1) E级cal_r_rd
 (2) E级cal_i_rt
 (3) E级load_rt
 (4) M级load_rt
二. Cal_r_rs/rt
 E级load_rt
三. Cal_i_rs
 E级load_rt
四. load_rs
 E级load_rt
五. store_rs
 E级load_rt
六. jr_rs
 (1)E级cal_r_rd
 (2) E级cal_i_rt
 (3) E级load_rt
 (4) M级load_rt

测试程序（Mars 与 CPU 的输出对拍对比直接使用 python 正则表达式匹配程序实现）

```
ori $1, $1, 1

nop
```

```
nop
nop
nop
addu $2, $1, $1
addu $2, $1, $2
subu $2, $2, $1
subu $2, $1, $2
addu $2, $1, $2
nop
nop
nop
nop
lui $1, 0
lui $2, 0
nop
nop
ori $1, $1, 1
nop
nop
nop
nop
addu $2, $1, $1
nop
addu $2, $1, $2
nop
subu $2, $2, $1
nop
```

```
subu $2, $1, $2
nop
addu $2, $1, $2
nop
nop
nop
nop
lui $1, 0
lui $2, 0
nop
nop
nop
ori $1, $1, 1
nop
nop
nop
nop
addu $2, $1, $1
nop
nop
addu $2, $1, $2
nop
nop
subu $2, $2, $1
nop
nop
subu $2, $1, $2
```

```
nop
nop
addu $2, $1, $2
nop
nop
nop
nop
ori $1, 123
addu $2, $1, $1
subu $3, $1, $1
nop
nop
nop
nop
ori $1, 321
subu $2, $1, $1
addu $3, $1, $1
nop
nop
nop
ori $1, 4321
nop
addu $2, $1, $1
subu $3, $1, $1
nop
nop
nop
```

```
nop
ori $1, 54212
nop
subu $2, $1, $1
addu $3, $1, $1
nop
nop
nop
nop
lui $1, 123
addu $2, $1, $1
subu $3, $2, $1
nop
nop
nop
nop
lui $1, 223
subu $3, $2, $1
addu $2, $1, $1
nop
nop
nop
nop
lui $1, 321
nop
addu $2, $1, $2
subu $3, $1, $1
```

```
nop

nop

nop

nop

lui $1, 432

nop

subu $3, $1, $1

addu $2, $1, $3

nop

nop

nop

jal label00

addu $2, $1, $31

label00: subu $1, $31, $1

jal label01

subu $1, $31, $1

label01: addu $2, $1, $31

jal label02

nop

label02: addu $2, $1, $31

subu $1, $31, $1

ori $2, $1, 123

ori $3, $1, 432

ori $4, $1, 543

jal label03

nop

label03: subu $1, $31, $1
```

```
addu $2, $1, $31

ori $3, $2, 123

ori $4, $2, 234

ori $5, $2, 345

ori $1, $5, 12345

ori $2, $5, 43121

ori $3, $5, 12313

ori $4, $5, 12421

lui $1, 12

ori $2, $1, 123

ori $2, $1, 121

ori $2, $1, 122

ori $2, $1, 143

jal label04

ori $1, $31, 123

label04 : ori $1, $31, 124

ori $1, $31, 125

ori $1, $31, 126

nop

nop

nop

addu $2, $1, $31

beq $2, $1, label05

nop

label05:

addu $3, $1, $31

nop
```

```
beq $3, $1, label06
nop
label06:
addu $4, $1, $31
nop
nop
beq $4, $1, label07
nop
label07:
subu $2, $1, $31
beq $2, $1, label08
nop
label08:
subu $3, $1, $31
nop
beq $3, $1, label09
nop
label09:
subu $4, $1, $31
nop
nop
beq $4, $1, label10
nop
label10:
ori $1, $4, 12312
beq $1, $31, label11
ori $2, $4, 12312
```

```
label11:nop
beq $2, $31, label12
ori $3, $4, 12312
label12: nop
nop
beq $3, $31, label13
ori $4, $3, 43121
label13: nop
nop
nop
beq $4, $31, label14
nop
label14:nop

lui $1, 12312
beq $1, $31, label15
lui $2, 12312
label15:nop
beq $2, $31, label16
lui $3, 12312
label16: nop
nop
beq $3, $31, label17
lui $4, 43121
label17: nop
nop
nop
```

```
beq $4, $31, label18

nop

label18:nop

jal label19

nop

label19: beq $31, $1, label20

nop

label20: jal label21

nop

label21: nop

beq $31, $1, label22

nop

label22: jal label23

nop

label23: nop

nop

beq $31, $1, label24

nop

label24:

ori $2, $0, 16

addu $1, $31, $31

sw $1, -16($2)

sw $1, -12($2)

sw $1, -8($2)

sw $1, 12($0)

subu $1, $1, $31

sw $1, 0($0)
```

```
sw $1, 4($0)
sw $1, 8($0)
sw $1, 12($0)
ori $1, $1, 5324
sw $1, 0($0)
sw $1, 4($0)
sw $1, 8($0)
sw $1, 12($0)
lui $1, 12
sw $1, 0($0)
sw $1, 4($0)
sw $1, 8($0)
sw $1, 12($0)
jal label25
sw $31, -16($2)
label25: sw $31, 4($0)
sw $31, 8($0)
sw $31, 12($0)
ori $1, $0, 4
jal label26
addu $31, $31, $1
nop
j label27
nop
label26: jr $31
nop
label27: jal label28
```

```
addu $31, $31, $1
nop
j label29
nop
label28: nop
jr $31
nop
label29: jal label30
addu $31, $31, $1
nop
j label31
nop
label30: nop
nop
jr $31
nop
label31: nop
nop

ori $1, $0, 4
jal label32
addu $31, $31, $1
j label33
nop
label32: subu $31, $31, $1
jr $31
nop
```

```
label33: jal label34
addu $31, $31, $1
j label35
nop
label34: subu $31, $31, $1
nop
jr $31
nop
label35: jal label36
addu $31, $31, $1
j label37
nop
label36: subu $31, $31, $1
nop
nop
jr $31
nop
label37: nop

ori $9, $9, 0x3000
jal label38
subu $31, $31, $9
j label39
nop
label38: ori $31, $31, 0x3000
jr $31
nop
```

```
label39: jal label40
subu $31, $31, $9
j label41
nop
label40: ori $31, $31, 0x3000
nop
jr $31
nop

label41: jal label42
subu $31, $31, $9
j label43
nop
label42: ori $31, $31, 0x3000
nop
nop
jr $31
nop

label43: jal label44
nop
j label45
nop
label44: jr $31
nop
```

```
label45: jal label46
nop
j label47
nop
label46: nop
jr $31
nop

label47: jal label48
nop
j label49
nop
label48: nop
nop
nop
jr $31

label49: nop

ori $1, $0, 0
lw $1, 0($0)
addu $2, $1, $1
addu $3, $2, $1
addu $4, $3, $1
addu $5, $4, $1
ori $1, $0, 0
nop
```

```
nop

nop

nop

lw $1, 0($0)

subu $2, $1, $1

subu $3, $2, $1

subu $4, $3, $1

subu $5, $4, $1

lw $2, 0($0)

ori $1, $2, 123

ori $1, $2, 321

ori $1, $2, 245

ori $1, $2, 1234

lw $3, 0($0)

beq $3, $1, label50

nop

beq $3, $1, label50

nop

beq $3, $1, label50

nop

beq $3, $1, label50

nop

label50: nop

lw $4, 0($0)

sw $4, 0x2000($0)

sw $4, 0x2004($0)

sw $4, 0x2008($0)
```

```
sw $4, 0x200c($0)
ori $9, $0, 0x3000
subu $4, $4, $9
sw $4, 0x0000($4)
jal label51
sw $31, 0($0)
jal label52
sw $31, 0($0)
jal label53
sw $31, 0($0)
j label54
label51: lw $1, 0($0)
jr $1
nop
label52: lw $1, 0($0)
nop
jr $1
nop
label53: lw $1, 0($0)
nop
nop
jr $1
label54: nop
ori $2, $0, 4
addu $1, $2, $0
lw $3, 0($1)
lw $3, 4($1)
```

```
lw $3, 8($1)

lw $3, -4($1)

subu $3, $1, $1

lw $4, 0($3)

lw $4, 4($3)

lw $4, 8($3)

lw $4, 12($3)

ori $3, $0, 8

lw $2, -8($3)

lw $2, -4($3)

lw $2, 0($3)

lw $2, 4($3)

sw $3, 0($0)

lw $1, 0($0)

lw $2, 0($1)

lw $2, -4($1)

lw $2, 4($1)

lw $2, -8($1)

ori $2, $0, 123

lui $2, 0

lw $3, 0($2)

lw $3, 4($2)

lw $3, 8($2)

lw $3, 12($2)

jal label55

lw $4, -0x3000($31)

label55: lw $4, -0x3000($31)
```

lw $4, -0x3000($31)

lw $4, -0x3000($31)

sw $1, 0($4)

六. 思考题

流水线冒险

1. 在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

   答:

```
assign  npc = (Br == `NPC_pc4 )? F_pc + 32'd4
            : (Br == `NPC_branch && b_jump == 1'b1 )? D_pc + 32'd4 + {{15{imm26[15]}},imm26[14:0],2'b00}
            : (Br == `NPC_26 )? { D_pc[31:28] , imm26 , 2'b00}
            : (Br == `NPC_reg )? rsRD1[31:0]
            : F_pc + 32'd4 ;
```

   首先，我们在 D 阶段根据 D_Instr 进行 NEW_PC 的判断。

   控制信号与单周期类似，分为顺序执行、branch 跳转、j 跳转到 imm26 对应的地址和 j 跳转到寄存器存储的地址这四种。其中顺序执行时基于流水线中最新的指令进行的（也就是 F 级指令），而其他各类跳转都是基于 D_Instr 执行的。跳转指令之后的 nop 延迟槽，可以保证跳转地址的正确性。

   关于数据通路：branch 类跳转指令的比较被提前到 D 级的 CMP 部件执行；j 跳转到寄存器存储的地址、地址采用 D 级 rs 经过转发后的量；j 跳转到 imm26 对应的地址的采用 D 级 Instr 中的立即数部分作为基准。

2. 对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8 ？

   答：因为 MIPS 是采用延时槽技术，在 D 段决定是否进行跳转，这个时候下一条指令已经进入了流水线，所以在跳转指令的下一条指令必须被执行，这就是延时槽技术。故采用的地址为：PC+8。

数据冒险的分析

1. 为什么所有的供给者都是存储了上一级传来的各种数据的**流水级寄存器**，而不是由 ALU 或者 DM 等部件来提供数据？

答：转发信号作为接收者的输入信号，从流水线寄存器传来可以保证所有部件的输入信号只在时钟上升沿更新，这样数据安全性更高更稳定；同时这样也能保证转发数据的来源同样是经过转发了的数据。

AT 法处理流水线数据冒险

1. “转发（旁路）机制的构造”中的 Thinking 1-4；

Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？

Answer 1： 会出现 Tuse<Tnew 的问题；上一级中的原始数据可能包含组合逻辑部分，会在非 clk 上升沿更新，使流水线数据安全性降低。

Thinking 2：我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

Answer 2：GRF 在一个时钟周期里是读写两用的；上升沿写入下降沿读出（半周期写入半周期读出）。

Thinking 3：为什么 0 号寄存器需要特殊处理？

Answer 3：若当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址均为 0，则当前位点的读取值一定为 0，但是转发来源的写入值不一定为 0，这样会造成错误。

Thinking 4：什么是“最新产生的数据”？

Answer 4: 在当前指令之前、离当前指令最近的指令产生的数据，也就是后续最近的流水线寄存器中的有效数据。

2. 在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了**用且仅用** A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

答：因为我实现转发机制时，转发值还并没有真正的写入寄存器，转发使用的正是还在流水线中流水的数据。我在实现转发时，对转发数据的使用的的地址依照指令功能的需要、做出了 rs、rt 的区分，也就是对 aRead 的选择已经做出了限制。当 aRead 和转发提供者的 aWrite 相等时，就已经满足了 AT 法转发的条件。

在线测试相关说明

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？

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略**，比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

暂停↓

			Tnew			
IF/ID阶段 Tuse分析			ID/EX			EX/MEM
指令类型	寄存器	Tuse	cal_r	cal_i	LOAD	LOAD
			rd-1	rt-1	rt-2	rt-1
BRANCH	rs/rt	0	S	S	S	S
LOAD	rs	1			S	
STORE	rs	1			S	
	rt	2				
jr	rs	0	S	S	S	S
jalr	rs	0	S	S	S	S
cal_r	rs/rt	1			S	
cal_i	rs	1			S	

转发↓

						ID/EX		EX/MEM				MEM/WB				
						jal	jalr	cal_r	cal_i	jal	jalr	cal_r	cal_i	load	jal	jalr
流水线级别	寄存器	涉及指令	MUX	控制信号	无转发值	31	rd	rd	rt	31	rd	rd	rt	rt	31	rd
D	rs	cal_r/cal_i/load/branch/store/j_reg	MF_RS_D	F_RS_D_Sel	D_RS	PC8_E	PC8_E	ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
D	rt	cal_r/branch/store	MF_RT_D	F_RT_D_Sel	D_RT	PC8_E	PC8_E	ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
E	rs	cal_r/cal_i/load/store	MF_RS_E	F_RS_E_Sel	E_RS			ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
E	rt	cal_r/store	MF_RT_E	F_RT_E_Sel	E_RT			ALUout	ALUout	PC8_M	PC8_M	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
M	rt	store	MF_RT_M	F_RT_M_Sel	M_RT							MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
			优先级——	D	0	3	3	1	1	4	4	2	2	2	5	5
				E	0			1	1	3	3	2	2	2	4	4
				M	0							1	1	1	2	2

上述两表是 AT 法分析指令数据冲突在 MIPS—2 的指令集下的全部情况，对每一种情况分别构造测试程序分析即可。（见前文的测试程序部分）