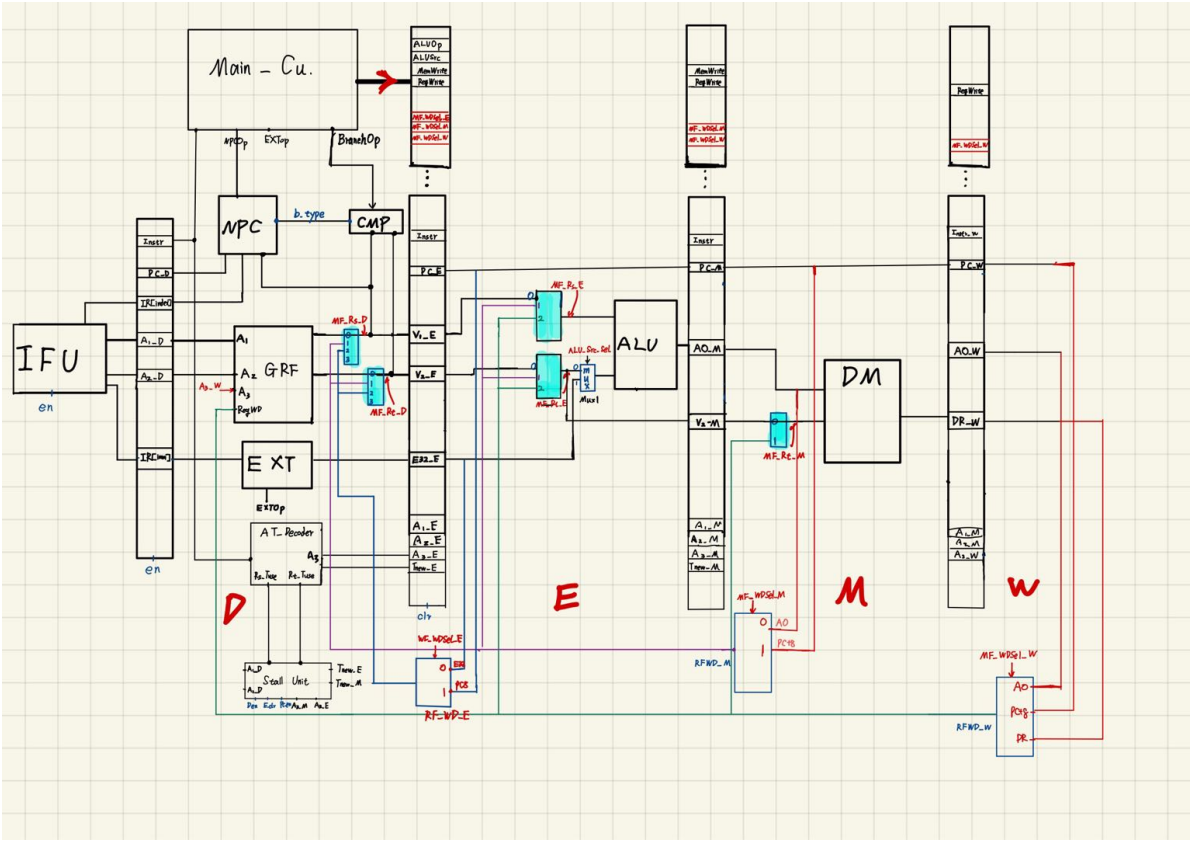
计算机组成P5实验报告

一、设计文档

（1）流水线CPU框架结构



本流水线CPU支持的指令集为{ add, sub, ori, lw, sw, beq, lui, jal, jr, nop }。

基于课堂上老师的PPT设计框架与《数字设计和计算机体系结构》中流水线CPU的框架设计，本人设计了如上五级流水线CPU框架，具体特征为：

1.采用集中式译码的方式

2.采用指令驱动型的控制信号产生方式

3.分离冒险处理模块与数据通路模块：冒险处理模块可以分为：AT编码器与阻塞模块

4.分阶段先整合转发数据，再往流水线前级进行转发

（2）数据通路设计

从设计难度的角度来思考，流水线CPU的数据通路是与单周期CPU较为接近的，故本人先从数据通路开始设计。流水线CPU与单周期CPU相比，增加了CMP模块以及四个流水线寄存器。CMP模块设计用于判断branch类型的指令，目的是为了提前判断branch的结果，提高流水线CPU的吞吐量。

在流水线寄存器方面，本实验设计的为5级流水线寄存器，相当于将原本单周期CPU分成5个独立的模块，模块的输入来自于流水线寄存器，输出前往下一级的流水线寄存器，从而实现不同级流水线处理不同指令的功能。

各级流水线流水的数据主要有以下几类：

1）主控制器产生的控制信号（基本同P3、P4，故不展开介绍）。

2）后续阶段所需要的数据：这里需要特别注意连线，流水线寄存器保存的值必须是经过转发更新后的值。

3）转发MUX的选择信号：本设计为了后续增添指令的便捷性，在E、M以及W阶段用选择信号选择转发时需要选择的数据。

1.F_IFU模块

输入输出端口设计：

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将PC寄存器清零。1：复位；0：无效。
en	I	使能信号，用于阻塞
NPC[31:0]	I	32位输入信号，下一条指令的地址
PC[31:0]	O	32位输出信号，当前指令的地址
Instr[31:0]	O	32位输出信号，当前指令的机器码

2.D_NPC模块

信号名	方向	描述
PC_F [31:0]	I	32位输入信号，F阶段指令的地址
PC_D[31:0]	I	32位输入信号，D阶段指令的地址
NPCOp[2:0]	I	下一个PC的选择信号
BranchSignal	I	1位输入信号，表示CMP模块比较结果
offset [31:0]	I	32位输入信号，代表地址偏移量
Instr_index [25:0]	I	26位输入信号，代表j/jal指令的26位地址索引
RegData [31:0]	I	32位输入信号，代表jr指令目标寄存器的值
NPC [31:0]	O	32位输出信号，下一个PC的地址

3.D_CMP模块

输入输出端口设计：

信号名	方向	作用
[31:0] MF_Rs_D	input	32位输入信号，代表branch类型指令需要的Rs数据
[31:0] MF_Rt_D	input	32位输入信号，代表branch类型指令需要的Rt数据
[2:0] BranchOp	input	3位输入信号，选择Branch类型中不同的指令
BranchSignal	output	1位输入信号，代表跳转条件是否成立

需要说明的是，由于在D阶段存在冒险的可能，故CMP模块的两个读入数据不能直接来源于GRF模块，而是需要接受经过转发后的数据，如表格中的MF_Rs_D与MF_Rt_D。

4.D_EXT模块

信号名	方向	描述
Imm [15:0]	I	16位输入信号，表示待进行拓展的立即数
EXTOp	I	EXT选择信号，1：进行符号拓展至32位；0：进行零拓展至32位
EXTOut [31:0]	O	32位输出信号，表示拓展后的立即数

5.D_GRF模块

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将32个寄存器的值全部清零
RegWrite	I	写使能信号，高电平时向A3寄存器写入数据WD3；低电平时无效
A1 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD1端口
A2 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD2端口
A3 [4:0]	I	5位输入信号，选择32个寄存器中的一个作为数据写入的目标寄存器
GRF_WD [31:0]	I	32位输入数据，当WE3有效时向GRF中写入数据
RF_RD1 [31:0]	O	32位输出信号，输出A1中选择的寄存器中的数据
RF_RD2 [31:0]	O	32位输出信号，输出A2中选择的寄存器中的数据

6.E_ALU模块

信号名	方向	描述
Src_A [31:0]	I	32位输入信号，待ALU处理的数据之一
Src_B [31:0]	I	32位输入信号，待ALU处理的数据之一
ALUOp [3:0]	I	3位输入选择信号，选择ALU需要进行的操作
ALUOut[31:0]	O	32位输出信号，ALU运算的输出结果

7.M_DM模块

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号；高电平时将RAM复位，低电平时无效
MemWrite	I	写使能信号，高电平时将WD中数据写入DM中，低电平时无效
DM_Addr [31:0]	I	32位输入信号，选择RAM中某个地址单元进行读写操作
DM_WD [31:0]	I	32位输入信号，待输入至DM中的数据
DM_RD [31:0]	O	32位输出信号，从DM中读出的数据

8.D_REG寄存器

```
module D_REG(
input clk,
input reset,
input en,
input [31:0] Instr_F,
input [31:0] PC_F,


output reg [31:0] Instr_D,
output reg [31:0] PC_D,
output reg [25:0] Instr_index_D,
output reg [15:0] Imm_D
output reg [4:0] A1_D,
output reg [4:0] A2_D,
);
```

9.E_REG寄存器

```
module E_REG(

input clk,
input reset,
```

```
input clr,
//come from MainCU
input [3:0] ALUOp,
input ALUSrc_Sel,
input MemWrite,
input RegWrite,
input [1:0] Elevel_Sel,
input [1:0] Mlevel_Sel,
input [1:0] Wlevel_Sel,

 //come from D_stage
input [31:0] Instr_D,
input [31:0] PC_D,
input [31:0] EXTOut,
input [4:0] A1_D,
input [4:0] A2_D,
 //come from AT-Decoder
input [4:0] A3,
input [2:0] Tnew,

 //come from forward-Mux
input [31:0] MF_Rs_D,
input [31:0] MF_Rt_D,

//control signal
output reg [3:0] ALUOp_E,
output reg ALUSrc_Sel_E,
output reg MemWrite_E,
output reg RegWrite_E,
output reg [1:0] Elevel_Sel_E,    //E级选择数据转发
output reg [1:0] Mlevel_Sel_E,    //M级选择数据转发
output reg [1:0] Wlevel_Sel_E,    //W级选择数据转发

output reg [31:0] Instr_E,
output reg [31:0] PC_E,
output reg [31:0] V1_E,
output reg [31:0] V2_E,
output reg [31:0] E32_E,

output reg [4:0] A1_E,
output reg [4:0] A2_E,
output reg [4:0] A3_E,
output reg [2:0] Tnew_E
);
```

10.M_REG寄存器

```
module M_REG(
input clk,
input reset,
input MemWrite_E,
input RegWrite_E,
input [1:0] Mlevel_Sel_E,
```

```
input [1:0] Wlevel_Sel_E,

input [31:0] Instr_E,
input [31:0] PC_E,
input [31:0] ALUOut,
input [31:0] V2_E,

input [4:0] A1_E,
input [4:0] A2_E,
input [4:0] A3_E,
input [2:0] Tnew_E,

output reg MemWrite_M,
output reg RegWrite_M,
output reg [1:0] Mlevel_Sel_M,  //M阶段转发数据选择信号
output reg [1:0] Wlevel_Sel_M,

output reg [31:0] Instr_M,
output reg [31:0] PC_M,
output reg [31:0] AO_M,     //写回寄存器的WriteData
output reg [31:0] V2_M,     //输入到DM中的WriteData

output reg [4:0] A1_M,
output reg [4:0] A2_M,
output reg [4:0] A3_M,
output reg [2:0] Tnew_M
);
```

11.W_REG寄存器

```
module W_REG(
input clk,
input reset,

input RegWrite_M,
input [1:0] Wlevel_Sel_M,

input [31:0] PC_M,
input [31:0] Instr_M,
input [31:0] AO_M,
input [31:0] DM_RD,

input [4:0] A1_M,
input [4:0] A2_M,
input [4:0] A3_M,

output reg RegWrite_W,
output reg [31:0] Instr_W,
output reg [1:0] Wlevel_Sel_W,      //W阶段转发数据选择信号

output reg [31:0] PC_W,
output reg [31:0] AO_W,
output reg [31:0] DR_W,
```

```
output reg [4:0] A1_W,
output reg [4:0] A2_W,
output reg [4:0] A3_W
);
```

(3) 控制单元设计

本设计控制单元基于教程网站的AT法进行设计，主要分为三个部分：AT编码器、阻塞单元以及主控制器，首先分析不同类型的指令的Tuse与Tnew值并作出有以下表格：

Tuse:指令进入 D 级后，其后的某个功能部件再经过多少时钟周期就必须要使用寄存器值。

Tnew:位于 E 级及其后各级的指令，再经过多少周期就能够产生要写入寄存器的结果。

当Tuse < Tnew且满足寄存器产生冲突且寄存器不为0号寄存器时，则需要触发阻塞机制，其余采取暴力转发即可。

Tuse(Rs)	time	Tuse(Rt)	time
cal_r(Rs)	1	cal_r(Rt)	1
cal_i(Rs)	1	store(Rt)	2
load(Rs)	1	branch(Rt)	0
store(Rs)	1		
branch(Rs)	0		
jumpreg(Rs)	0		

```
assign Rs_Tuse=  (jumpreg_D|branch_D)?3'd0:
                 (cal_r_D|cal_i_D|load_D|store_D)?3'd1:3'd5;

assign Rt_Tuse= (branch_D)?3'd0:
                (cal_r_D)?3'd1:
                (store_D)?3'd2:3'd5;
```

Tnew	time
jumplink	0
lui	0
cal_r	1
cal_i	1
load	2

```
assign Tnew=    (jumplink_D|lui_D)?3'd0:
                (cal_r_D|cal_i_D)?3'd1:
                (load_D)?3'd2:3'd0;
```

基于表格，具体设计如下：

1.阻塞单元

数据端口：

端口信号	方向	功能
[31:0] Instr_D	Input	32位输入信号，代表D级的指令
[2:0] Tnew_E	I	3位输入信号，代表E阶段指令的Tnew
[2:0] Tnew_M	I	3位输入信号，代表M阶段指令的Tnew
[4:0] A1_D	I	5位输入信号，代表D阶段指令的A1
[4:0] A2_D	I	5位输入信号，代表D阶段指令的A2
[4:0] A3_E	I	5位输入信号，代表E阶段指令的A3
[4:0] A3_M	I	5位输入信号，代表M阶段指令的A3
[2:0] Tnew	Output	3位输出信号，代表当前指令能产生新的写入值需要的周期数
[4:0] A3	Output	5位输出信号，代表当前指令的写入寄存器。若不需要写入寄存器，则A3的值为0
D_REG_en	O	1位输出信号，D级寄存器的使能信号
E_ERG_clr	O	1位输出信号，E级寄存器的清零信号
IFU_en	O	1位输出信号，IFU的使能信号

2.主控制器

端口信号	方向	功能
[5:0] Op	I	6位输入信号，代表D阶段指令的Op字段
[5:0] Funct	I	6位输入信号，代表D阶段指令的Funct字段
MemWrite	O	1位输出信号，DM写使能信号
RegWrite	O	1位输出信号，GRF写使能信号
[3:0] ALUOp	O	4位输出信号，ALU操作选择

端口信号	方向	功能
[1:0] EXTOp	O	2位输出信号，EXT操作选择
[2:0] NPCOp	O	3位输出信号，NPC操作选择
[2:0] BranchOp	O	3位输出信号，代表Branch类型指令的具体比较操作
ALUSrc_Sel	O	1位输出信号，ALU的源操作数选择Rs或者立即数
[1:0] Elevel_Sel	O	2位输出信号，E阶段选择数据进行转发
[1:0] Mlevel_Sel	O	2位输出信号，M阶段选择数据进行转发
[1:0] Wlevel_Sel	O	2位输出信号，W阶段选择数据进行转发

(4)转发机制设置

本设计并未选择构建额外的转发模块进行数据转发的处理，而是选择在数据通路内完成转发机制。具体设计分为转发输出与转发接收阶段：

在转发输出阶段，根据流水下寄存器的选择信号选择最终需要转发出去的数据；

在转发接收阶段，判断1.寄存器编号是否为0 2.寄存器编号是否相同（依据优先级判断），选择是否接收数据。

```
//----Forward----//
//--E_Level--//
wire [31:0] RF_WD_E= (Elevel_Sel_E==2'b00)? E32_E:
                     (Elevel_Sel_E==2'b01)? PC_E+8:32'hffffffff;
//--M_Level--//
wire [31:0] RF_WD_M=(Mlevel_Sel_M==2'b00)?AO_M:
                    (Mlevel_Sel_M==2'b01)?PC_M+8:32'hffffffff;

//--M_Level--//
wire [31:0] RF_WD_W= (Wlevel_Sel_W==2'b00)?AO_W:
                     (Wlevel_Sel_W==2'b01)?PC_W+8:
                     (Wlevel_Sel_W==2'b10)?DR_W:32'hffffffff;

//-----Receive----//
//--D_Level--//
wire [31:0] MF_Rs_D=(A1_D==5'd0)?32'd0:
                            (A1_D==A3_E)?RF_WD_E:
                            (A1_D==A3_M)?RF_WD_M:
                            RF_RD1;

wire [31:0] MF_Rt_D=(A2_D==5'd0)?32'd0:
                            (A2_D==A3_E)?RF_WD_E:
                            (A2_D==A3_M)?RF_WD_M:
                            RF_RD2;

//--E_Level--//
wire [31:0] MF_Rs_E=(A1_E==5'd0)?32'd0:
                            (A1_E==A3_M)?RF_WD_M:
```

```
                            (A1_E==A3_W)?RF_WD_W:
                            V1_E;

wire [31:0] MF_Rt_E=(A2_E==5'd0)?32'd0:
                            (A2_E==A3_M)?RF_WD_M:
                            (A2_E==A3_W)?RF_WD_W:
                            V2_E;

//--M_Level--//
wire [31:0] MF_Rt_M=(A2_M==5'd0)?32'd0:
                            (A2_M==A3_W)?RF_WD_W:V2_M;
```

二、测试数据

（1）转发机制测试

一、D级Rs

1.E级jal

```
ori $t0,$t0,10
jal label
add $t0,$ra,$0
label:
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
165@00003008: $ 8 <= 0000300c
```

2.E级lui

```
ori $t0,$t0,10
lui $t1,0x1111
add $t0,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 11110000
165@00003008: $ 8 <= 11110000
```

3.M级jal

```
ori $t0,$t0,10
jal label
nop
add $t0,$ra,$0
label:
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
```

4.M级lui

```
ori $t0,$t0,10
lui $t1,0x1111
nop
add $t0,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 11110000
175@0000300c: $ 8 <= 11110000
```

5.M级cal_r

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
175@0000300c: $10 <= 00000014
```

6.M级cal_i

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
ori $t2,$t1,10
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
175@0000300c: $10 <= 0000001e
```

7.W级jal

```
ori $t0,$t0,10
jal label
nop
nop
add $t0,$ra,$0
label:
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
```

8.W级lui

```
ori $t0,$t0,10
lui $t1,0x1111
nop
nop
add $t0,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 11110000
185@00003010: $ 8 <= 11110000
```

9.W级cal_r

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
185@00003010: $10 <= 00000014
```

10.W级cal_i

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
nop
ori $t2,$t1,10
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
185@00003010: $10 <= 0000001e
```

11.W级load

```
ori $t1,$t1,10
ori $t0,$t0,0
lw $t1,($t0)
nop
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 9 <= 0000000a
155@00003004: $ 8 <= 00000000
165@00003008: $ 9 <= 00000000
195@00003014: $10 <= 00000000
```

二、D级Rt

1.E级jal

```
ori $t0,$t0,10
jal label
add $t0,$0,$ra
label:
```

```
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
165@00003008: $ 8 <= 0000300c
```

2.E级lui

```
ori $t0,$t0,10
lui $t1,0x1111
add $t0,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 11110000
165@00003008: $ 8 <= 11110000
```

3.M级jal

```
ori $t0,$t0,10
jal label
nop
add $t0,$0,$ra
label:
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
```

4.M级lui

```
ori $t0,$t0,10
lui $t1,0x1111
nop
add $t0,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 11110000
175@0000300c: $ 8 <= 11110000
```

5.M级cal_r

```
ori $t0,$t0,10
add $t1,$t1,$t0
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 0000000a
175@0000300c: $10 <= 0000000a
```

6.M级cal_i

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
175@0000300c: $10 <= 00000014
```

7.W级jal

```
ori $t0,$t0,10
jal label
nop
nop
add $t0,$0,$ra
label:
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $31 <= 0000300c
```

8.W级lui

```
ori $t0,$t0,10
lui $t1,0x1111
nop
nop
add $t0,$0,$t1
```

9.W级cal_r

```
ori $t0,$t0,10
add $t1,$t1,$t0
nop
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 0000000a
185@00003010: $10 <= 0000000a
```

10.W级cal_i

```
ori $t0,$t0,10
ori $t1,$t1,20
nop
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 0000000a
155@00003004: $ 9 <= 00000014
185@00003010: $10 <= 00000014
```

11.W级load

```
ori $t1,$t1,10
ori $t0,$t0,0
lw $t1,($t0)
nop
nop
add $t2,$0,$t1
```

```
145@00003000: $ 9 <= 0000000a
155@00003004: $ 8 <= 00000000
165@00003008: $ 9 <= 00000000
195@00003014: $10 <= 00000000
```

三、E级Rs

1.M级jal

```
jal label
add $t0,$ra,$0
label:
```

```
测试结果：
145@00003000: $31 <= 00003008
155@00003004: $ 8 <= 00003008
```

2.M级lui

```
ori $t0,$0,1
lui $t1,100
add $t2,$t1,$0
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00640000
165@00003008: $10 <= 00640000
```

3.M级cal_r

```
ori $t0,$0,1
add $t1,$0,$t0
add $t2,$t1,$0
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000001
165@00003008: $10 <= 00000001
```

4.M级cal_i

```
ori $t0,$0,1
ori $t1,$0,100
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000064
165@00003008: $10 <= 00000064
```

5.W级jal

```
jal label
nop
label:
add $t0,$ra,$0
```

```
测试结果：
145@00003000: $31 <= 00003008
165@00003008: $ 8 <= 00003008
```

6.W级lui

```
ori $t0,$0,1
lui $t1,100
nop
add $t2,$t1,$0
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00640000
175@0000300c: $10 <= 00640000
```

7.W级cal_r

```
ori $t0,$0,1
add $t1,$0,$t0
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000001
175@0000300c: $10 <= 00000001
```

8.W级cal_i

```
ori $t0,$0,1
ori $t1,$0,100
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000064
175@0000300c: $10 <= 00000064
```

9.W级load

```
ori $t1,$t1,10
ori $t0,$t0,0
lw $t1,0($t0)
nop
add $t2,$t1,$0
```

```
测试结果：
145@00003000: $ 9 <= 0000000a
155@00003004: $ 8 <= 00000000
165@00003008: $ 9 <= 00000000
185@00003010: $10 <= 00000000
```

四、E级Rt

1.M级jal

```
jal label
add $t0,$0,$ra
label:
```

```
测试结果：
145@00003000: $31 <= 00003008
155@00003004: $ 8 <= 00003008
```

2.M级lui

```
ori $t0,$0,1
lui $t1,100
add $t2,$0,$t1
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00640000
165@00003008: $10 <= 00640000
```

3.M级cal_r

```
ori $t0,$0,1
add $t1,$0,$t0
add $t2,$0,$t1
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000001
165@00003008: $10 <= 00000001
```

4.M级cal_i

```
ori $t0,$0,1
ori $t1,$0,100
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000064
165@00003008: $10 <= 00000064
```

5.W级jal

```
jal label
nop
label:
add $t0,$0,$ra
```

```
测试结果：
145@00003000: $31 <= 00003008
165@00003008: $ 8 <= 00003008
```

6.W级lui

```
ori $t0,$0,1
lui $t1,100
nop
add $t2,$0,$t1
nop
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00640000
175@0000300c: $10 <= 00640000
```

7.W级cal_r

```
ori $t0,$0,1
add $t1,$0,$t0
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000001
175@0000300c: $10 <= 00000001
```

8.W级cal_i

```
ori $t0,$0,1
ori $t1,$0,100
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000064
175@0000300c: $10 <= 00000064
```

9.W级load

```
ori $t1,$t1,10
ori $t0,$t0,0
lw $t1,0($t0)
nop
add $t2,$0,$t1
```

```
测试结果：
145@00003000: $ 9 <= 0000000a
155@00003004: $ 8 <= 00000000
165@00003008: $ 9 <= 00000000
185@00003010: $10 <= 00000000
```

五、M级Rt

1.W级jal

```
jal label
sw $ra,0($0)
label:
```

```
测试结果：
145@00003000: $31 <= 00003008
145@00003004: *00000000 <= 00003008
```

2.W级lui

```
lui $t0,0x1111
sw $t0,0($0)
```

```
测试结果：
145@00003000: $ 8 <= 11110000
145@00003004: *00000000 <= 11110000
```

3.W级cal_r

```
ori $t0,$t0,100
add $t1,$t0,$0
sw $t1,0($0)
```

```
测试结果：
145@00003000: $ 8 <= 00000064
155@00003004: $ 9 <= 00000064
155@00003008: *00000000 <= 00000064
```

4.W级cal_i

```
ori $t0,$t0,100
ori $t1,$t0,10
sw $t1,0($0)
```

```
测试结果：
145@00003000: $ 8 <= 00000064
155@00003004: $ 9 <= 0000006e
155@00003008: *00000000 <= 0000006e
```

5.W级load

```
ori $t0,$t0,100
lw $t0,($0)
sw $t0,($0)
```

```
测试结果：
145@00003000: $ 8 <= 00000064
155@00003004: $ 8 <= 00000000
155@00003008: *00000000 <= 00000000
```

（2）阻塞机制测试

一、branch

1.E级 cal_r

```
ori $t0,$t0,1
add $t1,$t0,$0
beq $t0,$t1,label
nop
label:
add $s0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000001
195@00003010: $16 <= 00000000
```

2.E级 cal_i

```
ori $t0,$t0,1
ori $t1,$t0,10
beq $t0,$t1,label
nop
label:
add $s0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 0000000b
195@00003010: $16 <= 00000000
```

3.E级 load

```
ori $t0,$t0,1
lw  $t1,($0)
beq $t0,$t1,label
nop
label:
add $s0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000000
205@00003010: $16 <= 00000000
```

4.M级 load

```
ori $t0,$t0,1
lw  $t1,($0)
nop
beq $t0,$t1,label
nop
label:
add $s0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00000001
155@00003004: $ 9 <= 00000000
205@00003014: $16 <= 00000000
```

二、cal_r

1.E级 load

```
lw $t0,($0)
add $t1,$t0,$t0
```

```
测试结果：
145@00003000: $ 8 <= 00000000
165@00003004: $ 9 <= 00000000
```

三、cal_i

1.E级 load

```
lw $t0,($0)
ori $t0,$t0,100
```

```
测试结果：
145@00003000: $ 8 <= 00000000
165@00003004: $ 8 <= 00000064
```

四、load

1.E级 load

```
lw $t0,($0)
lw $t1,($t0)
```

```
测试结果：
145@00003000: $ 8 <= 00000000
165@00003004: $ 9 <= 00000000
```

五、store

1.E级 load

```
lw $t0,($0)
sw $t1,($t0)
```

```
测试结果：
145@00003000: $ 8 <= 00000000
155@00003004: *00000000 <= 00000000
```

六、jumpreg

1.E级 cal_r

```
ori $t0,$t0,0x300c
add $t1,$t0,$0
jr $t1
nop
add $t0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 0000300c
155@00003004: $ 9 <= 0000300c
205@00003010: $ 8 <= 00000000
```

2.E级 cal_i

```
ori $t0,$t0,0x3010
ori $t1,$t0,0
jr $t1
nop
add $t0,$0,$0
```

```
输出结果：
145@00003000: $ 8 <= 00003010
155@00003004: $ 9 <= 00003010
195@00003010: $ 8 <= 00000000
```

3.E级 load

```
ori $t0,$t0,0x3010
sw $t0,($0)
lw $t1,($0)
jr $t1
add $t0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00003010
145@00003004: *00000000 <= 00003010
165@00003008: $ 9 <= 00003010
205@00003010: $ 8 <= 00000000
215@00003010: $ 8 <= 00000000
```

4.M级 load

```
ori $t0,$t0,0x3014
sw $t0,($0)
lw $t1,($0)
nop
jr $t1
add $t0,$0,$0
```

```
测试结果：
145@00003000: $ 8 <= 00003014
145@00003004: *00000000 <= 00003014
165@00003008: $ 9 <= 00003014
205@00003014: $ 8 <= 00000000
215@00003014: $ 8 <= 00000000
```

（3）测试分析

本测试采用“枚举两条相关的指令单位”的方法，对本流水线CPU进行了转发、阻塞机制的测试，可以初步验证本流水线CPU的准确性。但是由于仅枚举了两条指令，可能会漏掉一些转发或暂停的错误，故对本流水线CPU的测试仍可进一步完善。

三、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。

原因：由于流水线CPU中存在数据冒险，beq指令需要从寄存器堆中读出Rs和Rt的值进行判断，而Rs和Rt的值会受到数据冒险的影响。尽管对beq的判断提前了，但是当数据冒险存在且无法通过转发解决时（如lw与beq组合），将不得不阻塞若干个时钟周期以产生正确的比较数值，这时候提前分支判断是没办法提高运行效率的。

例子：

`ori $t0,$t0,1`

`ori $t1,$t1,1`

`ori $t2,$t2,2`

`sw $t2,($0)`

`lw $t1,($0)`

`beq $t0,$t1,branch`

`branch:`

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?

因为针对jal等指令，由于延迟槽的存在，其下一条指令是必然会执行的（即PC+4的指令必然被执行）。为了在跳转返回时不用重复执行延迟槽的指令，我们需要设计返回地址为PC+8。

3、我们要求大家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?

因为流水线寄存器的设计初衷是将指令的执行的每个阶段分离开来，这样时钟周期就仅取决于需要最长执行时间的阶段。在每个周期的开始，该阶段所需要的数据都是在寄存器内已经准备好的，并且在周期结束之前输入到下一级寄存器值中。但是如果转发数据来源于功能部件，比如将ALU的输出端口与D级的转发接收MUX相连接，那么D阶段转发接收MUX要输出正确的数据就必须要等ALU执行结果产生，两个不同阶段的执行时间会发生变化导致时钟周期混乱。

4、我们为什么要使用 GPR 内部转发？该如何实现?

因为在流水线CPU中存在结构冒险的情况，即“寄存器文件需要在 D 级和 W 级同时被使用（读写）时并且读和写的寄存器为同一个寄存器时”。主要解决方法有：

1）在前半时钟周期进行写入，后半个时钟周期进行读出，但是该方法在verilog中较难实现

2）采用GRF内部转发，当结构冒险存在时，直接将写入的数据连接至读出的端口，同时向寄存器内写入新值，代码如下：

```
assign RF_RD1= (A1==5'b0)? 32'b0:
               (A1==A3&&RegWrite)?GRF_WD:registers[A1];

assign RF_RD2= (A2==5'b0)? 32'b0:
               (A2==A3&&RegWrite)?GRF_WD:registers[A2];
```

5、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?

数据的需求者：数据冒险的主体为Rs和Rt的数据，所以数据的需求者即是在不同阶段需要Rs和Rt数据的模块，主要有以下五个位置：

1）D阶段

CMP模块的两个32位输入端口，用以比较branch类型指令

D阶段用于写入E寄存器的Rs与Rt值

2）E阶段

ALU模块的两个源操作数

3）M阶段

用于写入DM的数据

数据的供给者：数据的供给者来源于E，M，W阶段

1）E阶段：该阶段lui与jal指令的值已经计算完成，可以进行转发，故供给者为PC_E与E32_E。

2）M阶段：该阶段cal_r、cal_I等计算类指令均已经计算完成，供给者为PC_M与AO_M

3）W阶段：该阶段所有指令的值均已计算完成，供给者为AO_M，PC_W,与DR_W

转发数据通路：转发的数据通路主要有以下五条数据通路

1)E阶段--->D阶段

2)M阶段--->E阶段

3)M阶段--->D阶段

4)W阶段--->E阶段

5)W阶段--->M阶段

6、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。

参照课程网站的分类，P5中mips指令集主要可以分为以下若干类：

- 寄存器立即数计算：`addi, addiu, slti, sltiu, andi, ori, xori, sll, srl, sra`
- 寄存器寄存器计算：`add, addu, sub, subu, slt, sltu, and, or, nor, xor, sllv, srlv, srav`
- 根据寄存器分支：`beq, bne, bgez, bgtz, blez, bltz`
- 写内存：`sw, sh, sb`
- 读内存：`lw, lh, lhu, lb, lbu`
- 跳转并链接：`jal, jalr`
- 跳转寄存器：`jr, jalr`
- 加载高位：`lui`
- 空指令：`nop`

1)寄存器立即数计算与寄存器寄存器计算：计算类指令在实现时较为简单，主要是需要在ALU根据新的ALUOp信号进行课上测试要求的操作即可。

2)根据寄存器分支：该类型指令主要需要在CMP模块中对对应的指令进行条件判断并输出branchsignal来判断是否需要进行分支跳转。

3)写内存:根据不同写内存类型，需要由控制器生成DM_mode并输入至DM之中以执行不同的写内存操作。

4)读内存：根据不同读内存类型，需要由控制器生成DM_mode并输入至DM之中以执行不同的读内存操作。

5)跳转并链接:该类型的指令经常与条件判断一起进行考察，故需要在生成条件的流水线级的数据通路中新增信号，并对往寄存器中流水的控制信号进行修改。

6)跳转寄存器:基本同上。

7、简要描述你的译码器架构，并思考该架构的优势以及不足。

译码器架构:我的流水线CPU采用集中式译码的方式，在D阶段完成全部译码并将产生的信号通过流水线寄存器进行输送。

具体而言，我的译码器架构分为三个部分：

1）AT编码器：该模块用于将D阶段指令译码产生相应的Tuse、Tnew以及A3值，并将Tuse输入到阻塞模块中用于判断阻塞条件，将Tnew与A3输入回数据通路之中在寄存器中流水。

2）阻塞模块：该模块接收来自AT编码器的Tuse以及流水线寄存器中Tnew_E,Tnew_M,以及E阶段与M阶段的写入寄存器值，并且产生对应的阻塞信号，进行插入控制令的操作。

3）主控制器：该模块接收D阶段的指令并产生各类控制信号，并将控制信号输入至流水线寄存器中进行流水。

优势：本架构整体来看采用集中式译码，只需要在初始对指令进行一次译码，减少后续操作的复杂性。具体来看，译码器架构分为三个字模块，各模块功能清晰、明确。

不足：译码器与数据通路之间传输数据量较大，修改代码时较为复杂。

8.[P5 选做] 请详细描述你的测试方案及测试数据构造策略。

测试方案：本人采取的测试方案为自主构建测试数据以验证流水线CPU的正确性，具体而言为枚举两条相关的指令单位并在中间插入零到两个无关指令。

测试数据构造策略：

在构造测试数据的时候，首先根据转发的五个接收端分别构造数据冒险以验证CPU是否能处理转发，针对五个接收端与每个接收端可能接受的数据来源构建测试数据以验证是否能够转发正确数据。

然后根据Tuse与Tnew表格，设计测试数据验证在CPU需要产生阻塞时是否能够正常阻塞。具体的两条相关指令构建即为比表格中Tuse<Tnew的地方。