计算机组成原理　P6 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-lite3。采用模块化和层次化设计。
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

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
WE	I	写使能信号 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..BE (E 级的存储数据扩展模块)

```
module BE(
    input [1:0] A,
    input [31:0] OldData,
    input [2:0] OP_LOAD,
    output [31:0] DMout
);
```

A	I	被加载的内存的地址的末两位
OldData	I	被加载内存的地址存储的整字数据
OP_LOAD	I	Load 指令类型的控制信号
DM_OUT	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
    );
```

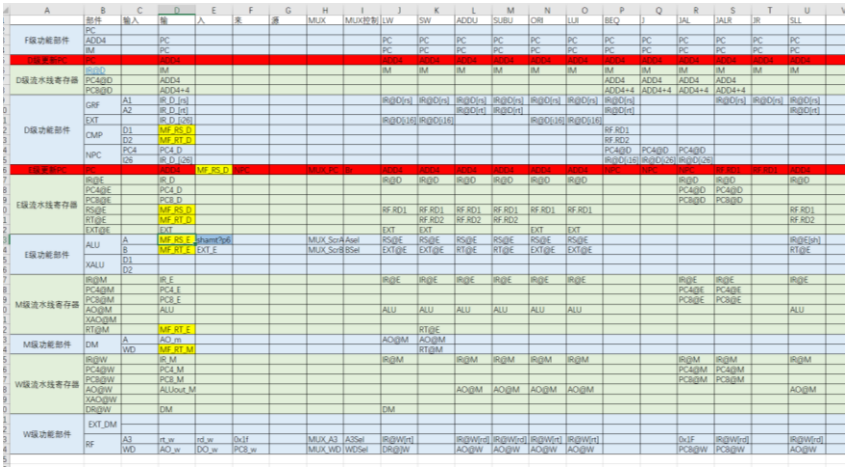
12.MULT_DIV （乘除模块）

```
module MULT_DIV(
    input clk,
    input reset,

    input [31:0] RS,
    input [31:0] RT,

    input [3:0] type,

    output MDstall,
    output [31:0] MD_out
);
```

信号名	方向	功能描述
RS[31:0]	I	32 位输入数据 1
RT[31:0]	I	32 位输入数据 2
TYPE[3:0]	I	乘除单元指令控制信号控制信号
Clk	I	时钟信号
reset	I	复位信号
MDstall	O	标志乘除模块正在启动（start）或者正在工作延迟中（busy）
MD_OUT	O	乘除模块的输出

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,
    input E_MDstall,
    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 ;
```

相较 P5 新增对乘除模块暂停的判断

```
    // as data giver(mf) : Tnew
    // as data user (md,mt) : Tuse
    // consider inner problems :  MUL_DIV is busy but we are going to use this module.
```

① mf　为 GRF 提供数据，作为数据的提供者，判断其 Tnew（在 E 阶段 Tnew 为 1，在 M 阶段 Tnew 为 0）

② md、mt　是数据的使用者，判断其 Tuse，均在 E 级使用数据，故在 D 级 Tuse 均为 1；md 的 rs 和 rt 均需要使用数据、而 mt 只有 rs 需要使用数据

③ 还需考虑专属于 MultDiv 模块的冲突。当 E 级的乘除模块处于启动 start 或者工作 busy 状态、而 D 级指令又是乘除类指令时，若不在 D 级将指令暂停，则乘除模块内部会造成冲突（主要是 hi、lo 两寄存器的冲突），故对该种情况造成的暂停进行特判：

```
    wire stall_MD = E_MDstall && ( D_md | D_mf | D_mt) ;
  assign stall = stall_E_rs | stall_E_rt | stall_M_rs |
stall_M_rt | stall_MD;
```

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

在 P5 的基础上，需要额外测试乘除模块的冲突；

Tuse Tnew 分析时，md 类指令可与 cal_R 等效，mt 类指令可与 cal_I 等效；

然后再测试当 E 级的乘除模块处于启动 start 或者工作 busy 状态、而 D 级指令又是乘除类指令这种冲突

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）

（1）P5 测试转发功能

```
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)

（2）测试 p6 新增指令行为

Addi addiu

```
ori $2,$zero,0x1

addiu $3,$2,3
```

```
addiu $4,$2,-3


ori $2,$zero,0x1
addi $3,$2,3
addi $4,$2,-3
```

add addu sub subu

```
lui $7 ,0xffff
ori $7 ,$7,0xffff
ori $8 ,$zero,0x2
add $9,$7,$8

ori $10 ,$0,0xffff
ori $11 ,$0,0x2
addu $12,$10,$11

ori $13 ,$0,0x2
ori $14 ,$0,0xff
sub $15,$13,$14

ori $16 ,$0,0x33
ori $17 ,$0,0x22
subu $18,$16,$17
```

and andi slt slti sltu sltiu

```
ori $7 ,$zero ,0x1111
ori $8 ,$zero ,0x2223
li $9,0x12131111

and $10, $7,$8
andi $11, $7,34
andi $12 ,$8,563
```

```
ori $13 ,$zero ,2

ori $14 ,$zero ,1

ori $15 ,$zero ,2

ori $16 ,$zero ,3


slt $17,$13,$14       # >

slt $18,$13,$15     # =

slt $19,$13,$16    # <


sltu $17,$13,$14       # >

sltu $18,$13,$15     # =

sltu $19,$13,$16    # <


slti $17,$14 ,34    # >

slti $18,$15 ,2     # =

slti $19,$16 ,4    # <


sltiu $17,$14 ,34    # >

sltiu $18,$15 ,2     # =

sltiu $19,$16 ,4    # <
```

BRANCH

```
ori $1 ,$zero ,5

ori $2 ,$zero ,5

ori $3 ,$zero ,6


beq $1 ,$2 ,cut1

lui $10 ,0xffff

cut1:

lui $11 ,0xffff

beq $1 ,$3 ,cut2

lui $12 ,0xffff

cut2:
```

```
lui $13 ,0xffff

#####################################################

ori $1 ,$zero ,5

ori $2 ,$zero ,5

ori $3 ,$zero ,6

bne $1 ,$3 ,cut3

lui $10 ,0xffff

cut3:

lui $11 ,0xffff

bne $1 ,$2 ,cut4

lui $12 ,0xffff

cut4:

lui $13 ,0xffff

#################################################################

ori $1 ,$zero ,5

ori $2 ,$zero ,0

ori $3 ,$zero ,-3

blez $1     ,cut5

lui $10 ,0xffff

cut5:

lui $11 ,0xffff

blez $2     ,cut6

lui $12 ,0xffff

cut6:

lui $13 ,0xffff

blez $3     ,cut7

lui $14 ,0xffff

cut7:
```

```
lui $15 ,0xffff

################################################################################33333

ori $1 ,$zero ,5
ori $2 ,$zero ,0
ori $3 ,$zero ,-3

bgtz $1     ,cut8
lui $10 ,0xffff
cut8:
lui $11 ,0xffff
bgtz $2     ,cut9
lui $12 ,0xffff
cut9:
lui $13 ,0xffff
bgtz $3     ,cut10
lui $14 ,0xffff
cut10:
lui $15 ,0xffff

################################################################################33333
ori $1 ,$zero ,5
ori $2 ,$zero ,0
ori $3 ,$zero ,-3

bltz $1     ,cut11
lui $10 ,0xffff
cut11:
lui $11 ,0xffff
bltz $2     ,cut12
lui $12 ,0xffff
cut12:
```

```
lui $13 ,0xffff

bltz $3     ,cut13

lui $14 ,0xffff

cut13:

lui $15 ,0xffff


#################################################################################33333

ori $1 ,$zero ,5

ori $2 ,$zero ,0

ori $3 ,$zero ,-3


bgez $1     ,cut14

lui $10 ,0xffff

cut14:

lui $11 ,0xffff

bgez $2     ,cut15

lui $12 ,0xffff

cut15:

lui $13 ,0xffff

bgez $3     ,cut16

lui $14 ,0xffff

cut16:

lui $15 ,0xffff
```

逻辑运算

```
and $10, $7,$8

andi $11, $10,34

andi $12 ,$11,563


or $10, $7,$8

ori $11, $10,45

ori $12 ,$11,11
```

```
xor $10, $7,$8

xori $11, $10,43

xori $12 ,$11,595


nor $10, $7,$8

nor $11, $10, $10

nor $12 ,$11, $11
```

load 、 store

```
.text

ori $8, $zero, 0

lui $9,0x1234

ori $9, $9, 0x56b8


sw $9, 0($8)

sh $9, 4($8)

sh $9, 2($8)

sb $9, 0($8)

sb $9, 1($8)

sb $9, 2($8)

sb $9, 3($8)


lb $11, 0($8)

lbu $12, 1($8)

lb $13,2($8)

lb $14,3($8)

lh $15, 0($8)

lhu $16, 2($8)

add $17,$16,$16
```

移位

```
ori $8, $zero, 1

sll $8, $8, 2
```

```
ori $8, $zero, 0x1000

srl $8, $8, 2


li $8, 0xf0000000

sra $8, $8, 2


li $9 ,2

ori $8, $zero, 1

sllv $8, $8,$9


ori $8, $zero, 0x1000

srlv $8, $8, $9


li $8, 0xf0000000

srav $8, $8, $9
```

乘除指令

```
ori $1, $0, 4

ori $2, $0, 5

ori $3, $0, 6

ori $4, $0, 12

sw $3, 0($0)

sw $2, 4($0)

sw $2, 8($0)

sw $2, 12($0)

sw $1, 16($0)

sw $4, 20($0)

sw $2, 24($0)

sw $4, 28($0)

sw $3, 32($0)

sw $3, 36($0)

sw $2, 40($0)

sw $2, 44($0)
```

```
sw $2, 48($0)
sw $4, 52($0)
sw $1, 56($0)
sw $3, 60($0)
sw $2, 64($0)
sw $1, 68($0)
sw $2, 72($0)
sw $2, 76($0)
sw $4, 80($0)
sw $1, 84($0)
sw $3, 88($0)
sw $3, 92($0)
sw $1, 96($0)
sw $2, 100($0)
sw $2, 104($0)
sw $2, 108($0)
sw $1, 112($0)
sw $1, 116($0)
sw $1, 120($0)
sw $4, 124($0)
lbu $1, 0($2)
beq $1, $2, TAG1
lbu $4, 0($1)
lw $3, 0($1)
TAG1:
lui $2, 13
sb $3, 0($3)
lui $2, 7
and $3, $2, $3
TAG2:
mtlo $3
addi $2, $3, 11
mult $2, $2
```

```
lui $3, 6

TAG3:

mtlo $3

lui $1, 14

sll $0, $0, 0

slti $4, $1, 15

TAG4:

mtlo $4

lbu $1, 0($4)

sltiu $4, $4, 7

addiu $4, $4, 12

TAG5:

subu $2, $4, $4

lui $2, 3

sll $0, $0, 0

div $2, $2

TAG6:

mflo $1

bgtz $4, TAG7

sb $4, 0($4)

beq $1, $1, TAG7

TAG7:

lb $1, 0($1)

bne $1, $1, TAG8

sh $1, 0($1)

bgtz $1, TAG8

TAG8:

mthi $1

mflo $4

lbu $4, 0($4)

mtlo $4

TAG9:

beq $4, $4, TAG10
```

```
mthi $4

mthi $4

lw $3, 0($4)

TAG10:

xor $4, $3, $3

bne $4, $3, TAG11

mtlo $4

mfhi $4

TAG11:

sltu $2, $4, $4

mflo $1

sb $4, 0($2)

mflo $2

TAG12:

bgtz $2, TAG13

ori $2, $2, 13

slti $2, $2, 10

bgez $2, TAG13

TAG13:

sh $2, 0($2)

mthi $2

bltz $2, TAG14

mtlo $2

TAG14:

lw $4, 0($2)

bltz $4, TAG15

mfhi $2

sltiu $4, $4, 10

TAG15:

mflo $2

sb $2, 0($2)

srl $4, $2, 0

mtlo $2
```

```
TAG16:
lui $4, 7
ori $2, $4, 15
bgez $4, TAG17
mthi $2
TAG17:
ori $4, $2, 6
sllv $4, $4, $2
bne $4, $2, TAG18
lui $1, 2
TAG18:
sll $0, $0, 0
mflo $3
lui $1, 12
mthi $1
TAG19:
mfhi $2
mfhi $1
multu $1, $1
mfhi $1
TAG20:
sw $1, -144($1)
lui $4, 8
sll $0, $0, 0
lhu $1, -144($1)
TAG21:
slt $2, $1, $1
mult $2, $2
lui $4, 3
bgez $2, TAG22
TAG22:
mfhi $2
xori $4, $2, 5
```

```
beq $4, $2, TAG23

mult $4, $4

TAG23:

lui $2, 6

div $2, $4

mfhi $2

subu $3, $2, $2

TAG24:

beq $3, $3, TAG25

slti $4, $3, 13

mthi $4

bne $3, $3, TAG25

TAG25:

sll $1, $4, 1

mfhi $4

mfhi $1

blez $1, TAG26

TAG26:

sb $1, 0($1)

bne $1, $1, TAG27

lb $4, 0($1)

lb $3, 0($4)

TAG27:

lui $3, 15

mfhi $2

mthi $3

beq $3, $3, TAG28

TAG28:

mtlo $2

lbu $4, 0($2)

ori $2, $2, 15

blez $4, TAG29

TAG29:
```

```
sltu $2, $2, $2

lw $2, 0($2)

mthi $2

lb $2, -400($2)

TAG30:

lui $4, 9

lw $2, 112($2)

xor $3, $2, $2

mflo $4

TAG31:

mtlo $4

bgez $4, TAG32

lui $2, 2

multu $4, $4

TAG32:

mfhi $4

sw $4, -400($4)

mtlo $4

slti $2, $2, 3

TAG33:

mfhi $4

lui $3, 14

and $1, $3, $2

lhu $4, 0($1)

TAG34:

sll $0, $0, 0

lh $3, -400($4)

lw $1, -400($4)

sra $1, $1, 2

TAG35:

lh $1, 0($1)

lbu $2, 0($1)

bltz $2, TAG36
```

```
mtlo $1

TAG36:

lui $1, 0

sb $1, 0($2)

lui $1, 3

mtlo $1

TAG37:

blez $1, TAG38

mtlo $1

sll $0, $0, 0

sll $0, $0, 0

TAG38:

beq $4, $4, TAG39

lh $4, -400($4)

mfhi $1

divu $1, $4

TAG39:

bne $1, $1, TAG40

sll $0, $0, 0

sll $0, $0, 0

sll $0, $0, 0

TAG40:

xori $1, $1, 8

bne $1, $1, TAG41

mtlo $1

blez $1, TAG41

TAG41:

lui $4, 8

blez $1, TAG42

mfhi $3

blez $1, TAG42

TAG42:

sh $3, -400($3)
```

```
sw $3, -400($3)

srl $3, $3, 9

beq $3, $3, TAG43

TAG43:

lui $3, 15

mthi $3

sll $0, $0, 0

mtlo $3

TAG44:

sll $0, $0, 0

mflo $1

multu $3, $1

mtlo $1

TAG45:

mfhi $2

sb $1, -225($2)

lui $4, 6

lui $1, 2

TAG46:

multu $1, $1

bne $1, $1, TAG47

lui $1, 9

mflo $2

TAG47:

bne $2, $2, TAG48

sh $2, 0($2)

lui $3, 7

mthi $2

TAG48:

addu $3, $3, $3

sra $1, $3, 6

bgtz $3, TAG49

sltiu $4, $3, 13
```

```
TAG49:

mflo $4

bgez $4, TAG50

lui $2, 6

bgez $4, TAG50

TAG50:

mflo $3

lui $1, 2

lui $4, 6

mtlo $4

TAG51:

bne $4, $4, TAG52

mflo $1

mfhi $2

sra $4, $2, 0

TAG52:

sb $4, 0($4)

sh $4, 0($4)

srav $2, $4, $4

nor $2, $4, $4

TAG53:

divu $2, $2

mtlo $2

andi $4, $2, 1

lui $1, 13

TAG54:

sll $0, $0, 0

multu $1, $1

divu $1, $1

lui $1, 8

TAG55:

blez $1, TAG56

slti $4, $1, 5
```

```
sh $4, 0($4)

bgtz $4, TAG56

TAG56:

lui $3, 4

mult $3, $3

sll $0, $0, 0

mflo $4

TAG57:

lb $3, 0($4)

slti $2, $3, 11

bltz $2, TAG58

srl $4, $3, 13

TAG58:

sh $4, 0($4)

mflo $4

mfhi $1

mtlo $4

TAG59:

mthi $1

mthi $1

andi $4, $1, 7

lui $4, 1

TAG60:

sll $0, $0, 0

sll $0, $0, 0

sh $1, 0($1)

sll $0, $0, 0

TAG61:

xori $2, $1, 0

srlv $2, $1, $2

mtlo $1

slti $3, $2, 14

TAG62:
```

```
blez $3, TAG63

mflo $4

mfhi $2

div $2, $3

TAG63:

bne $2, $2, TAG64

subu $3, $2, $2

mult $2, $3

divu $2, $2

TAG64:

sh $3, 0($3)

lbu $2, 0($3)

sw $3, 0($3)

bgez $2, TAG65

TAG65:

ori $3, $2, 14

sb $3, 0($3)

xor $1, $3, $3

lui $1, 15

TAG66:

lui $3, 0

bgez $1, TAG67

sll $0, $0, 0

sltu $3, $3, $1

TAG67:

lui $2, 5

mtlo $3

lw $2, 0($3)

mthi $2

TAG68:

lui $1, 10

bne $1, $2, TAG69

sltiu $3, $1, 3
```

```
mtlo $2

TAG69:

beq $3, $3, TAG70

multu $3, $3

mfhi $4

sb $3, 0($4)

TAG70:

sw $4, 0($4)

mflo $4

beq $4, $4, TAG71

xor $4, $4, $4

TAG71:

nor $3, $4, $4

subu $1, $3, $4

mfhi $4

addu $4, $3, $4

TAG72:

addu $4, $4, $4

beq $4, $4, TAG73

divu $4, $4

or $1, $4, $4

TAG73:

blez $1, TAG74

sw $1, 1($1)

mtlo $1

srav $4, $1, $1

TAG74:

sh $4, 2($4)

mfhi $1

sltu $2, $4, $4

lh $4, 0($2)
```

（3）测试 因为增加 p6 乘除指令带来的暂停

见（2）乘除指令部分

（4）综合测试：使用讨论区强测数据（见附件）

六．思考题

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

答：实际的 CPU 实现乘除法也是通过加法实现的，实现乘法相较 ALU 里的简单计算耗时更久，如果并入 ALU 会降低整个 CPU 的运行效率。

HI、LO 寄存器可以使有多周期时延的乘除法过程和无多周期时延的 ALU 计算过程相互独立，再通过 stall 暂停机制的巧妙运转，可以提高 CPU 的运算效率。

- 参照你对延迟槽的理解，试解释“乘除槽”。

答：类似延迟槽，当乘除法进行的时候，会有一个 start 信号，在下一个周期会产生 busy 信号，但是这个时间内并不影响其他指令的执行，而且不止一条其他指令，而延迟槽只是一条指令的继续执行。只要 md 类型指令后边跟的不是乘除法型指令，就不需要暂停了，乘除槽使得其他跟在后面的指令的执行不受影响。

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

答：“abcd”是字符串常量，连续存储。按照字访问，想取出一个字符，需要先取字，再取字符。而字节访问就比较简单，可以直接取一个字符。此时按字节访问内存相对于按字访问内存性能上更有优势。

- 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。

此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答

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，再加上乘除法指令带来的新的暂停情况，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）

- 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？
- 1. 指令 func、opcode 段采用 parameter 常数替代，减少错误
- 2. 控制信号取值用宏表述，方便在多文件中复用
- 3. 对指令进行分类，并将指令分类结果作为 CU 的输出，在需要使用时可以进行分布式译码

```
//TYPE
assign BRANCH = beq | bne | blez | bgtz | bltz | bgez;
assign STORE =  sw | sh | sb ;
assign LOAD =   lw | lh | lb | lhu | lbu ;

assign CAL_R  = ( addu | subu | add | sub ) | ( and_ | or_ | xor_ | nor_) | ( sll | sra | srl) | (sllv|srav|srlv) | (slt|sltu);
assign CAL_I  =  lui | ( addiu | addi ) | (ori | andi| xori ) | ( slti|sltiu );

assign j_26  = j | jal ;
assign j_reg = jr |jalr ;
assign j_link = jal | jalr ;

// assign shift_s = sll | sra | srl ;
// assign shift_v = sllv | srav |srlv ;
assign md = mult | multu | div | divu ;
assign mt = mtlo | mthi ;
assign mf = mflo | mfhi ;
```

好处：在用 AT 法分析 CPU 的暂停和阻塞的时候，可以合并指令同类项，减少错误并且降低了指令的复杂度，增加了指令的可扩展性。

例如：jal，jalr 同属于 j_link，意思是跳转并链接 PC+8 的指令，这样在译码 WDSel 信号的时候就可以将两个指令合并。

```
/// W
assign WDSel = (LOAD )? `WD_load
             :(j_link )? `WD_addr_jalX
             :( mf ) ? `WD_MuxDiv
             : `WD_ALU  ;
```