



同濟大學
TONGJI UNIVERSITY

计算机系统结构课程实验 总结报告

实验题目：动态流水线设计与性能定量分析

学号：1951443

姓名：罗劲桐

指导教师：陆有军

日期：2021.12.19

一、实验环境部署与硬件配置说明

Artix-7 NEXYS 4 DDR FPGA 硬件测试环境

Vivado 2016.2 开发环境

ModelSim 模拟仿真软件

二、实验的总体结构

1、 顶层模块结构

顶层模块包括动态流水线 CPU 和 7 段数码管两部分。通过接受数据 `reset`、`cpu_stall` 控制 CPU 复位和流水暂停。通过 `switch_pc_d` 选择 CPU 输出的数据当前 PC 值或#16 寄存器（对应 D[i]位置），并使用 7 段数码管显示出来。

2、 动态流水线的总体结构

此次实验实现的 CPU 为 54 条 MIPS 动态流水线 CPU，支持内中断（`break`、`teq`、`syscall` 等指令）和外中断（按键中断暂停 CPU 运行），支持设置专用路径实现动态数据前推到 id 段，支持推后处理的 id 段流水暂停等待，实现算数操作、乘除法运算、存储器数据传送、内中断、中断返回、跳转、条件跳转指令，并且可侦测读写冲突、溢出错误、乘除法暂停。

动态流水线分为五段流水，分别为 IF、ID、EX、ME、WB 段。所有流水段均在时钟上升沿执行，写流水寄存器；时钟下降沿执行流水，指令向下传递一个流水段，写指令寄存器。

CPU 顶部模块 `top_parts` 包含各分段子模块 `if_inst`、`id_inst`、`ex_inst`、`me_inst`、`wb_inst`，流水控制器 `flow_control_inst`，以下详细介绍各子模块。

3、 动态流水线的分段结构

1) IF 段

IF 段包含子模块指令存储器 `imem`，。PC 输入选择 `mux_Pc`。

IF 段执行取指令操作：上升沿写 PC 寄存器，写入下一个周期要执行的指令地址，下降沿从 `imem` 取指令至 IR 寄存器。

2) ID 段

ID 段包含子模块控制器 `controller_id`，流水寄存器输入选择 `mux_ALUa`、`mux_ALUb`，扩展选择 `ext_id`，寄存器模块 `cpu_ref`，CPO 模块 `cp0_inst`。

ID 段执行译码操作：主要包括从寄存器、CPO 寄存器，HI、LO 取值，中断执行和中断返回，转移指令成功的判断和地址的生成。上升沿写 `ALUa`、`ALUb` 算数流水寄存器，`Rt` 寄存器。下降沿将本段 IR 寄存器写入下一段的 IR 寄存器。组合逻辑生成寄存器取值地址，转移指令成功的判断和转移地址等。（转移地址提前到 ID 段生成，

因此不需要预测和排空流水线，提高了 CPU 吞吐率)。

另外，forward_ALUa、forward_ALUb、forward_Rt 三项输入实现了设置专用路径的定向技术实现动态数据前推到 id 段，加速了流水运行速率。

3) EX 段

EX 段包含子模块控制器 controller_ex，乘除法器 calculator_inst，ALU 算术单元 alu_inst。

EX 段执行算数执行操作：主要从 ALUa、ALUb 取值并执行算数操作，写入下一段流水寄存器。上升沿算术单元输出写 Z 寄存器，Rt 寄存器内容写到下一段 Rt 寄存器。组合逻辑输出算术逻辑输出。

由于乘除法是下降沿执行，流水状态寄存器是上升沿修改的，为防止与流水状态冲突，HI、LO 流水寄存器写入向后延迟半周期，上升沿修改。

4) ME 段

ME 段包含子模块控制器 controller_me，存储器 dmem_inst，数据扩展 ext_me。

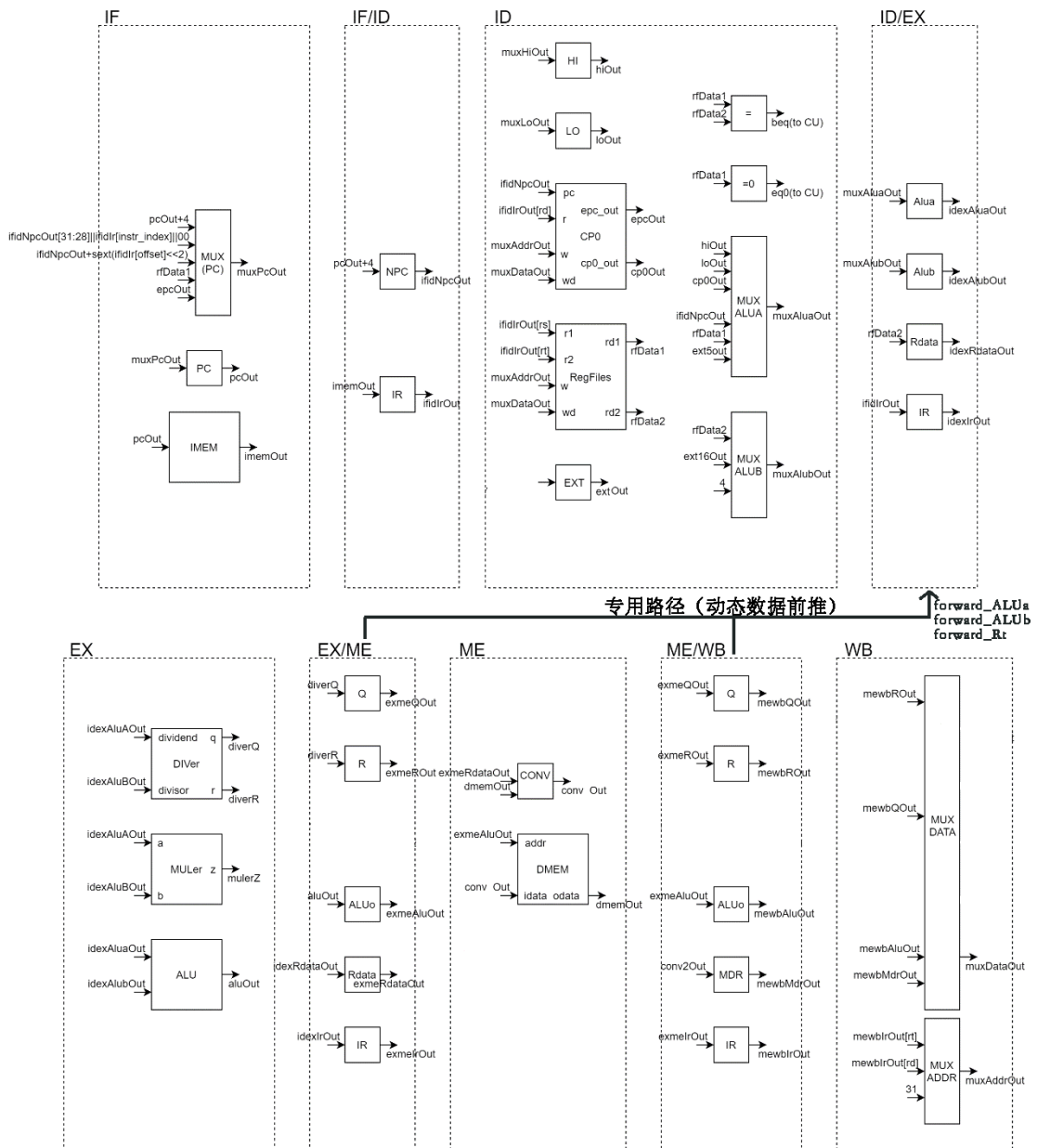
ME 段执行存储交互操作：上升沿使用 Rt 寄存器写到内存指定单元，或内存读出数据写道 MEM 流水寄存器，HI、LO、Z 寄存器内容写到下一段流水寄存器。组合逻辑生成数据有符号、无符号扩展结果。

5) WB 段

WB 段包含子模块控制器 controller_wb，寄存器写入地址选择 mux_Rdc，寄存器写入选择 mux_Rd。

WB 段执行寄存器写回操作：主要将内存读出的数值、计算的数值写回到寄存器、CP0、HI、LO。上升沿将写回地址 mux_Rdc_out，写回数值 mux_Rd_out 写到 ID 段中的寄存器、CP0、HI、LO 中。组合逻辑生成写回的地址和数值。

4、 动态流水线的总体结构图



5、流水控制器的总体结构

包含时序逻辑当前状态寄存器 **state**，时钟上升沿更新。状态有 **FLOW_NORMAL** 所有流水正常执行、**FLOW_OVERFLOW** 流水遇加减法溢出、**FLOW_MULDIV** 流水遇乘除法、**FLOW_VIOLATE** 流水遇读写冲突。

包含组合逻辑下一状态 **nextstate**，**cond[0:4]** 5 段流水线流水状态控制。这些状态控制每一段流水继续进行，停止或排空流水段。状态有 **PARTS_COND_FLOW** 流水段正常执行、**PARTS_COND_STALL** 流水段停止、**PARTS_COND_ZERO** 流水段排空。

具体情况如下：

- 正常运行时

1. `cond[0]<='PARTS_COND_FLOW; //IF`
2. `cond[1]<='PARTS_COND_FLOW; //ID`
3. `cond[2]<='PARTS_COND_FLOW; //EX`

```

4.      cond[3]<= `PARTS_COND_FLOW; //ME
5.      cond[4]<= `PARTS_COND_FLOW; //WB
6.      nextstate<= `FLOW_NORMAL;

```

- 发生乘除法器暂停时，暂停 32 周期等待乘除法器运行结束

```

1.      cond[0]<= `PARTS_COND_STALL; //IF
2.      cond[1]<= `PARTS_COND_STALL; //ID
3.      cond[2]<= `PARTS_COND_STALL; //EX
4.      cond[3]<= `PARTS_COND_STALL; //ME
5.      cond[4]<= `PARTS_COND_STALL; //WB
6.      nextstate<= `FLOW_MULDIV;

```

- 发生溢出时，排空一个流水段

```

1.      cond[0]<= `PARTS_COND_FLOW; //IF
2.      cond[1]<= `PARTS_COND_FLOW; //ID
3.      cond[2]<= `PARTS_COND_ZERO; //EX
4.      cond[3]<= `PARTS_COND_FLOW; //ME
5.      cond[4]<= `PARTS_COND_FLOW; //WB
6.      nextstate<= `FLOW_NORMAL;

```

- 发生读写冲突时，ID 前流水段先暂停：

```

1.      cond[0]<= `PARTS_COND_STALL; //IF
2.      cond[1]<= `PARTS_COND_ZERO; //ID
3.      cond[2]<= `PARTS_COND_FLOW; //EX
4.      cond[3]<= `PARTS_COND_FLOW; //ME
5.      cond[4]<= `PARTS_COND_FLOW; //WB
6.      nextstate<= `FLOW_NORMAL;

```

- 其他组合冲突的情况不再赘述，请参看 flow_control.v

另外，流水控制器包含组合逻辑的冲突判别和定向技术的专用路径选择。violation1、violation2、violation3 分别判断 EX 段、ME 段、RB 段的流水指令写入内容是否与 ID 段读取相关。violation 判断当前数据相关是否需要流水暂停（否则可以通过专用路径动态前推解决）。其中，专用通道的源流水寄存器由 EX、ME、RB 之间两个流水寄存区组成，包括 ex_HI、ex_LO、ex_Z、me_HI、me_LO、me_Z、me_MEM。目标流水寄存器为 ID、EX 间流水寄存区，包括 id_ALUa、id_ALUb、id_Rt。

6、 七段数码管的总体结构

将当前 PC 值或#16 寄存器（对应 D[i]位置）通过 switch_pc_d 选择并输入到七段数码管的输入数据中。在七段数码管时钟上升沿时，保存输入到内部寄存器 i_data_store，并切换 o_sel_r 数码管选择。组合逻辑输出数码管 7 段选择 o_seg_r。

三、总体架构部件的解释说明

1、 动态流水线总体结构部件的解释说明

1) 顶部模块

```

module top_parts(
    input clk,                //CPU 时钟信号
    input reset,              //CPU 复位信号，高电平有效
    output [31:0] top_pc,     //PC 寄存器地址输出
    output [31:0] top_ir,     //指令寄存器输出

    //for program out
    output [31:0] reg_16,     //16 寄存器输出
    input cpu_stall           //CPU 外中断暂停信号
);

```

2) IF 流水段模块

```

module instruction_fetch(
    input clk,                //CPU 时钟信号
    input reset,              //CPU 复位信号，高电平有效

    input [31:0] connect,     //j, jal 指令转移地址拼接输出
    input [31:0] npc_ext,     //相对地址转移指令输出
    input [31:0] regfile_Rs,  //寄存器转移指令输出
    input [31:0] cp0_EPC,     //CP0 返回地址
    input [31:0] cp0_intr_addr, //CP0 中断地址

    output [31:0] oNPC,       //NPC 地址
    output reg [31:0] rPC,    //当前指令地址 PC
    output reg [31:0] rIR,    //流水寄存器指令 IR

    //control signal
    input [1:0] cond,         //流水执行状态
    input [2:0] mux_pc_sel    //PC 寄存器输入选择
);

```

3) ID 流水段模块

```

module instruction_decode(
    input clk,                //CPU 时钟信号
    input reset,              //CPU 复位信号，高电平有效

    input [31:0] if_IR,       //流水寄存器 IR 输入
    input [31:0] if_NPC,      //流水寄存器 NPC 输入
    input [4:0] regfile_Rdc,   //寄存器（或 CP0）写入地址
    input [31:0] regfile_Rd,   //寄存器（或 CP0、HI、LO）写入数值
    input [31:0] Rd_out_for_LO, //LO 寄存器写入（仅当乘除法指令同时写入 HI、LO 时有效）

```

```

output reg [31:0] rALUa, //流水寄存器 ALUa
output reg [31:0] rALUb, //流水寄存器 ALUb
output reg [31:0] rRt, //流水寄存器 Rt
output reg [31:0] rIR, //流水寄存器 IR
output [31:0] cp0_EPC, //CP0 返回地址
output [31:0] cp0_intr_addr, //CP0 中断地址
output [31:0] ext_out, //数据扩展输出(对应相对地址转移
指令)

output [31:0] connect, //j, jal 指令转移地址拼接输出
output [31:0] regfile_Rs, //寄存器 Rs 输出
output [31:0] regfile_Rt, //寄存器 Rt 输出

//control signal
input [1:0] cond, //流水执行状态
output [2:0] mux_pc_sel, //PC 寄存器输入选择
input hi_w, //HI 写入使能
input lo_w, //LO 写入使能
input regfile_w, //寄存器写入使能
input cp0_w, //CP0 写入使能
output [6:0] flow_raddr1, //流水控制条件, 读地址
output [6:0] flow_raddr2, //流水控制条件, 读地址

//forward
input [31:0] forward_ALUa, //动态前推 ALUa 专用通道
input [31:0] forward_ALUb, //动态前推 ALUb 专用通道
input [31:0] forward_Rt, //动态前推 Rt 专用通道
input forward_ALUa_w, //ALUa 专用通道使能
input forward_ALUb_w, //ALUb 专用通道使能
input forward_Rt_w, //Rt 专用通道使能

//for program out
output [31:0] reg_16 // #16 寄存器输出
);

```

4) EX 流水段模块

```

module execute(
    input clk, //CPU 时钟信号
    input reset, //CPU 复位信号, 高电平有效

    input [31:0] alua, //算数部件输入
    input [31:0] alub, //算数部件输入
    input [31:0] id_Rt, //流水寄存器 Rt 输入
    input [31:0] id_IR, //流水寄存器 IR 输入

```

```

output reg [31:0] rHI, //流水寄存器 HI 乘除法输出
output reg [31:0] rLO, //流水寄存器 LO 乘除法输出
output reg [31:0] rZ, //流水寄存器 Z ALU 输出
output reg [31:0] rRt, //流水寄存器 Rt
output reg [31:0] rIR, //流水寄存器 IR

//control signal
input [1:0] cond, //流水执行状态
output mult_div_stall, //流水控制条件，乘除法暂停
output cal_finish, //流水控制条件，乘除法结束
output overflow_stall, //流水控制条件，加减法溢出
output [6:0] flow_waddr, //流水控制条件，写地址

//for program in
input cpu_stall //CPU 外中断暂停信号
);

```

5) ME 流水段模块

```

module memory_access(
    input clk, //CPU 时钟信号
    input reset, //CPU 复位信号，高电平有效

    input [31:0] ex_HI, //流水寄存器 HI 输入
    input [31:0] ex_LO, //流水寄存器 LO 输入
    input [31:0] ex_Z, //流水寄存器 Z 输入
    input [31:0] ex_Rt, //流水寄存器 Rt 输入
    input [31:0] ex_IR, //流水寄存器 IR 输入

    output reg [31:0] rHI, //流水寄存器 HI
    output reg [31:0] rLO, //流水寄存器 LO
    output reg [31:0] rZ, //流水寄存器 Z
    output reg [31:0] rMEM, //流水寄存器 MEM，存储器输出
    output reg [31:0] rIR, //流水寄存器 IR

    //control signal
    input [1:0] cond, //流水执行状态
    output [6:0] flow_waddr, //流水控制条件，写地址
    output dmem_r, //读内存信号，用于专用路径判断
    output use_mul //mul 乘法信号，用于专用路径判断
);

```

6) WB 流水段模块

```

module write_back(
    input clk, //CPU 时钟信号

```



```

input reset,                //CPU 复位信号，高电平有效

input [31:0] me_HI,         //流水寄存器 HI 输入
input [31:0] me_LO,         //流水寄存器 LO 输入
input [31:0] me_Z,          //流水寄存器 Z 输入
input [31:0] me_MEM,        //流水寄存器 MEM 输入
input [31:0] me_IR,         //流水寄存器 IR 输入

output [4:0] mux_Rdc_out,    //寄存器（或 CP0）写入地址
output [31:0] mux_Rd_out,    //寄存器（或 CP0、HI、LO）写入数
值
output [31:0] Rd_out_for_LO, //LO 寄存器写入（仅当乘除法指令
同时写入 HI、LO 时有效）

//control signal
input [1:0] cond,           //流水执行状态
output hi_w,                //HI 写入使能
output lo_w,                //LO 写入使能
output cp0_w,               //CP0 写入使能
output regfile_w,           //寄存器写入使能
output [6:0] flow_waddr,    //流水控制条件，写地址
output dmem_r,              //读内存信号，用于专用路径判断
output use_mul               //mul 乘法信号，用于专用路径判断
);

```

2、 动态流水线控制器模块的解释说明

1) 流水控制模块

```

module flow_control(
input clk,
input reset,
input [6:0] raddr1,        //流水控制条件，读地址
input [6:0] raddr2,        //流水控制条件，读地址
input [6:0] waddr1,        //流水控制条件，写地址
input [6:0] waddr2,        //流水控制条件，写地址
input [6:0] waddr3,        //流水控制条件，写地址
input mult_div_stall,       //流水控制条件，乘除法暂停
input mult_div_over,        //流水控制条件，乘除法结束
input overflow_stall,       //流水控制条件，加减法溢出

/*-----flow_control-----*/
output [1:0] cond0,         //流水执行状态
output [1:0] cond1,         //流水执行状态
output [1:0] cond2,         //流水执行状态

```

```

output [1:0] cond3,          //流水执行状态
output [1:0] cond4          //流水执行状态

/*-----flow_data-----*/
output reg [31:0] id_ALUa,    //动态前推 ALUa 专用通道
output reg [31:0] id_ALUb,    //动态前推 ALUb 专用通道
output reg [31:0] id_Rt,      //动态前推 Rt 专用通道
output id_ALUa_w,            //ALUa 专用通道使能
output id_ALUb_w,            //ALUb 专用通道使能
output id_Rt_w,              //Rt 专用通道使能
input [31:0] ex_HI,          //专用通道输入 HI
input [31:0] ex_LO,          //专用通道输入 LO
input [31:0] ex_Z,           //专用通道输入 Z
input ex_from_mem,           //ME 段内指令将读取内存，用于专
用路径判断
input ex_mul,                //ME 段内指令是 mul 乘法，用于专
用路径判断
input [31:0] me_HI,          //专用通道输入 HI
input [31:0] me_LO,          //专用通道输入 LO
input [31:0] me_Z,           //专用通道输入 Z
input [31:0] me_MEM,         //专用通道输入内存读出数据
input me_from_mem,           //RB 段内指令将读取内存，用于专
用路径判断
input me_mul,                //RB 段内指令是 mul 乘法，用于专
用路径判断

//for program in
input cpu_stall              //CPU 外中断暂停信号
);

```

2) 组合控制逻辑模块

```

module controller(
input [31:0] inst, //组合逻辑输入指令
input judge_beq, // BEQ BNE 跳转指令满足
input judge_bgez, // BGEZ 跳转指令满足

/*-----flow_control-----*/
output reg [6:0] raddr1,output reg [6:0] raddr2,output reg [6:0]
waddr,output reg dmem_r,          //流水控制条
件，读、写地址，读内存

/*-----control-----*/
//ID
output reg [2:0] mux_pc_sel,      //PC 寄存器输入选择

```

```

        output reg [2:0] mux_ALUa_sel,output reg [1:0] mux_ALUb_sel, //
流水寄存器输入选择
        output reg [2:0] ext_sel,          //数据扩展输出选择
        output reg cp0_exception,output reg cp0_eret,output reg [4:0]
cp0_cause,          //CP0 指令输入，中断原因输入
        //EX
        output reg [3:0] alu_sel,output reg [1:0] cal_sel, //算数操作选择
        output reg cal_ena/*also for stall*/,output reg use_overflow, //算
数操作控制（乘除法器开始，使用溢出判断）
        //ME
        output reg dmem_w,output reg [1:0] dmem_width,output reg [2:0]
mem_sel,          //存储器写入使能、长度、扩展
方式
        //WB
        output reg [1:0] mux_Rdc_sel,output reg [1:0] mux_Rd_sel, //寄存
器写会地址、数据选择
        output reg hi_w,output reg lo_w,output reg regfile_w,output reg
cp0_w          //写回使能
    );

```

3、 动态流水线子模块部件的解释说明

1) CPU 寄存器堆

```

module regfile(
    input clk,          //时钟信号，上升沿写入
    input rst,          //复位信号
    input we,           //写入使能，高电平有效
    input [31:0] raddr1, //32 位 Rsc
    input [31:0] raddr2, //32 位 Rtc
    input [31:0] rdata1, //32 位 Rs
    input [31:0] rdata2, //32 位 Rt
    input [31:0] waddr,  //32 位 Rdc
    input [31:0] wdata   //32 位 Rd

    //for program out
    output [31:0] reg_16//#16 寄存器输出
);

```

2) ALU 算数逻辑单元

```

module alu(
    input [31:0] a,      //32 位输入，操作数 1
    input [31:0] b,      //32 位输入，操作数 2
    input [3:0] aluc,     //4 位输入，控制 alu 的输出结果
    output reg [31:0] r,  //32 位输出，a，b 经过 aluc 指定的操作生成

```

```

output reg zero,    //0 标志位，不使用
output reg carry,   //进位标志位，不使用
output reg negative, //负数标志位，不使用
output reg overflow //溢出标志位
);

```

3) 数据选择器

```

module mux_len32_sel8(    //32 位 8 选 1 数据选择器
    input [2:0] sel,
    input [31:0] iData_1,
    input [31:0] iData_2,
    input [31:0] iData_3,
    input [31:0] iData_4,
    input [31:0] iData_5,
    input [31:0] iData_6,
    input [31:0] iData_7,
    input [31:0] iData_8,
    output reg [31:0] oData
);

module mux_len32_sel4(    //32 位 4 选 1 数据选择器
    input [1:0] sel,
    input [31:0] iData_1,
    input [31:0] iData_2,
    input [31:0] iData_3,
    input [31:0] iData_4,
    output reg [31:0] oData
);

module mux_len32_sel2(    //32 位 2 选 1 数据选择器
    input sel,
    input [31:0] iData_1,
    input [31:0] iData_2,
    output reg [31:0] oData
);

module mux_len5_sel4(    //5 位 4 选 1 数据选择器
    input [1:0] sel,
    input [4:0] iData_1,
    input [4:0] iData_2,
    input [4:0] iData_3,
    input [4:0] iData_4,
    output reg [4:0] oData
);

```

4) 外部存储器

```

module ram(

```

```

input clk,                //时钟信号, posedge write-active
input ena,                //片选信号 active-high
input wena,               //写有效 high:write low:read
output reg finish,        //存储结束信号 active-high
input [1:0] width,        //数据宽度 0:word,1:hword,2:byte
input [31:0] addr,        //32 位地址
input [31:0] data_in,     //32 位数据输入, 宽度不足低位有效
output reg [31:0] data_out //32 位数据输出, 宽度不足低位有效
);

```

5) CP0 协处理器

```

module CP0(
    input clk,                //时钟信号, posedge write-active
    input rst,                //复位信号, 高电平有效
    input mfc0,               //cpu 指令 mfc0,high-active
    input mtc0,               //cpu 指令 mtc0,high-active
    input [31:0] pc,          //32 位 pc 地址
    input [4:0] Rd,           //选定 CP0 reg
    input [31:0] wdata,        //data from GP reg to place CP0 reg
    input exception,          //对应指令 syscall,break,teq,high-active
    input eret,               //指令 eret,high-active
    input [4:0] cause,        //中断原因
    input intr,
    output [31:0] rdata,       //data from CP0 reg for GP reg
    output [31:0] status,      //中断禁止位置,low-active
    output reg timer_int,      //中断允许
    output [31:0] exc_addr     //32 位中断时 pc 位置
);

```

6) 乘除法器

```

module calculator(
    input clk,                //时钟信号, posedge write-active
    input [31:0] a,           //32 位输入 multiplicand/dividend
    input [31:0] b,           //32 位输入 multiplier/divisor
    input [1:0] calc,         //计算控制指令
    input reset,              //复位信号, 高电平有效
    input ena,                //开始计算 high-active
    output reg [31:0] oLO,     //输出到 LO
    output reg [31:0] oHI,     //输出到 HI
    output sum_finish,        //计算结束标志

    //for program in
    input cpu_stall            //CPU 外中断暂停信号
);

```

7) 有符号除法器

```
module DIV(  
    input [31:0] dividend, //32 位无符号被除数  
    input [31:0] divisor, //32 位无符号除数  
    input start, //读入数据开始执行，高电平有效  
    input clock, //时钟信号，上升沿有效  
    input reset, //复位信号，active-high  
    output [31:0] q, //32 位商输出  
    output [31:0] r, //32 位余数输出  
    output reg busy, //除法器正在执行指示位，高电平有效  
    output reg finish, //计算结束信号，保持一周期高电平  
  
    //for program in  
    input cpu_stall //CPU 外中断暂停信号  
);
```

8) 无符号除法器

```
module DIVU(  
    input [31:0] dividend, //32 位无符号被除数  
    input [31:0] divisor, //32 位无符号除数  
    input start, //读入数据开始执行，高电平有效  
    input clock, //时钟信号，上升沿有效  
    input reset, //复位信号，active-high  
    output [31:0] q, //32 位商输出  
    output [31:0] r, //32 位余数输出  
    output reg busy, //除法器正在执行指示位，高电平有效  
    output reg finish, //计算结束信号，保持一周期高电平  
  
    //for program in  
    input cpu_stall //CPU 外中断暂停信号  
);
```

9) 有符号乘法器

```
module MULT(  
    input clk, //输入时钟信号，posedge write-active  
    input reset, //乘法器复位信号，active high  
    input start, //读入数据开始执行，高电平有效  
    input [31:0] a, //32 位被乘数 multiplicand  
    input [31:0] b, //32 位乘数 multiplier  
    output reg [63:0] z, //64 位无符号乘法结果输出  
    output reg busy, //乘法器正在执行指示位，高电平有效  
    output reg finish, //计算结束信号，保持一周期高电平
```

```

//for program in
input cpu_stall           //CPU 外中断暂停信号
);

```

10) 无符号乘法器

```

module MULTU(
    input clk,           //输入时钟信号，posedge write-active
    input reset,         //乘法器复位信号，active high
    input start,         //读入数据开始执行，高电平有效
    input [31:0] a,      //32 位被乘数 multiplicand
    input [31:0] b,      //32 位乘数 multiplier
    output reg [63:0] z,  //64 位无符号乘法结果输出
    output reg busy,     //乘法器正在执行指示位，高电平有效
    output reg finish,   //计算结束信号，保持一周期高电平

    //for program in
    input cpu_stall      //CPU 外中断暂停信号
);

```

4、 七段数码管部件的解释说明

1) 下板总顶部模块，连接 CPU 和七段数码管

```

module seg7_top(
    input clk_in,         //开发板 100MHZ 时钟信号
    input reset,          //总复位信号
    output [7:0] o_seg,   //7 段数码管输出
    output [7:0] o_sel,   //7 段数码管输出

    input cpu_stall,      //CPU 外中断暂停信号
    input switch_pc_d     //七段数码管输出选择
);

```

2) 7 段数码管模块

```

module seg7x16(
    input clk,           //输入时钟信号
    input reset,         //复位信号,high-active
    input cs,            //片选,high-active
    input [31:0] i_data,  //32 位输入显示信号
    output [7:0] o_seg,   //7 段数码管输出
    output [7:0] o_sel    //7 段数码管输出
);

```

四、实验仿真过程

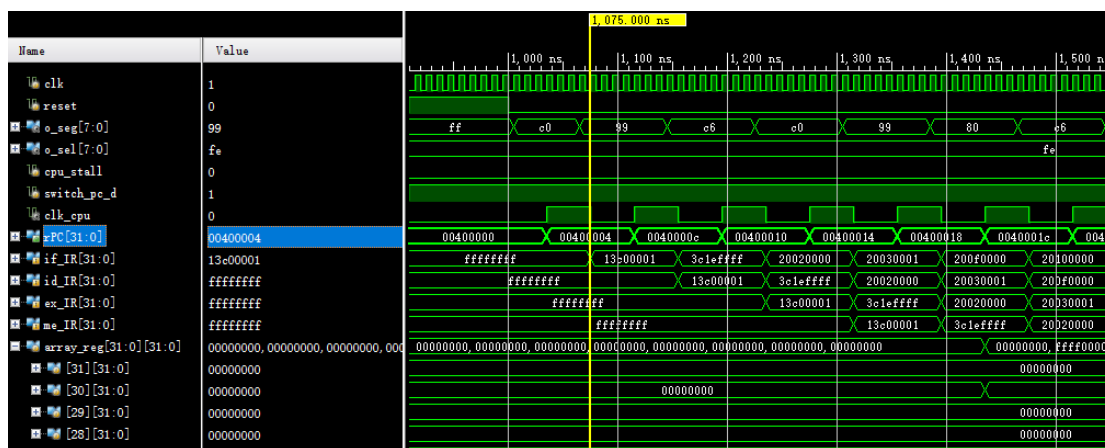
1、测试样例说明

仿真测试的程序即下面的实验验算模型，输出 D[59]为 9FD0B7EA（#16 寄存器的值）。

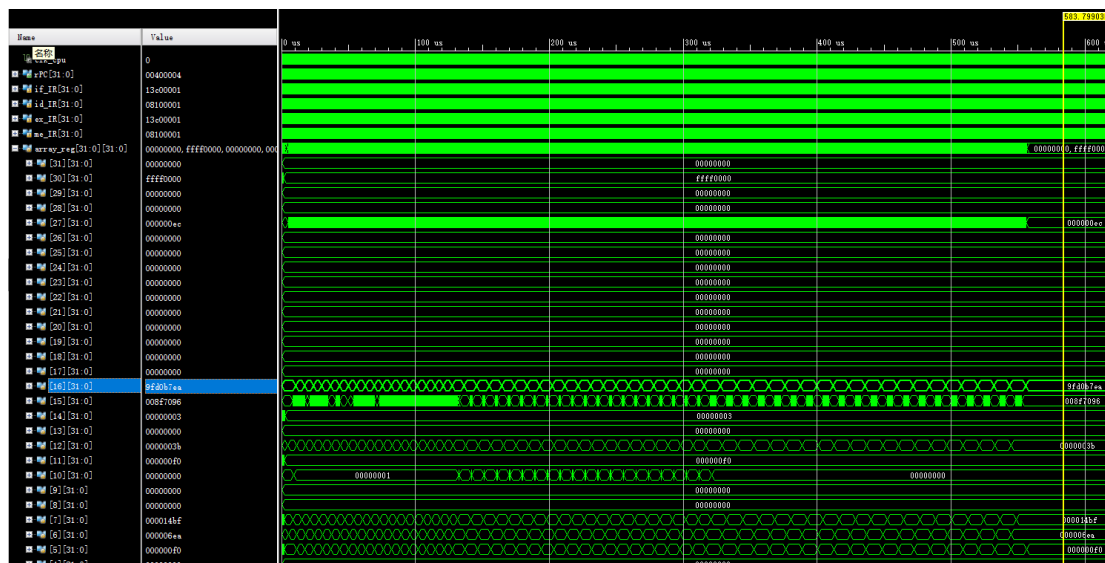
前仿真过程采用波形图输出和寄存器输出到文件的方式查看结果。后仿真无法访问寄存器，因此只采用波形图形式。

2、 动态流水线的前仿真过程

1) 仿真结果

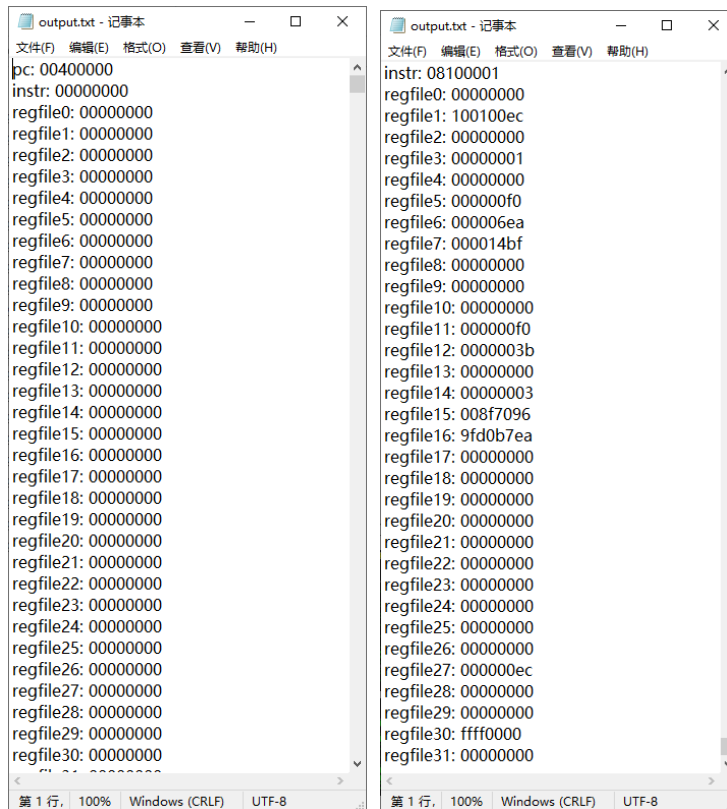


通过 if IR、id IR、ex IR、me IR 可观察各流水段流动和停止情况。

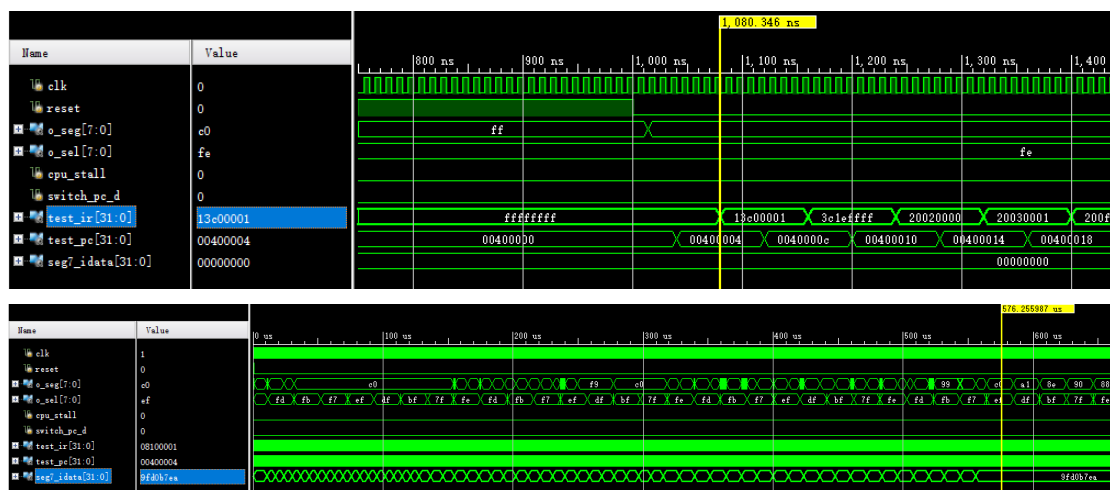


最终结果 array reg[16]= 9FD0B7EA。结果正确。

2) 寄存器结果输出



3、 动态流水线的后仿真过程(分频 CLK_PERIOD=3)



最终结果 9FD0B7EA。结果正确。

五、 实验仿真的波形图及某时刻寄存器值的物理意义

1、 动态流水线的波形图及某时刻寄存器值的物理意义

程序中用到的所有寄存器分为四部分。初始值寄存器#2、#3，分别寄存 A[0]、B[0] 的初值。结果寄存器，#6、#7、#15、#16 分别存储 A[i]、B[i]、C[i]、D[i]在不同


```

int a[m],b[m],c[m],d[m];
a[0]=0;
b[0]=1;
a[i]=a[i-1]+i;
b[i]=b[i-1]+3i;
c[i]=
{
    a[i],          0≤i≤19
    a[i]+b[i],    20≤i≤39
    a[i]*b[i],    40≤i≤59
}
d[i]=
{
    b[i],          0≤i≤19
    a[i]*c[i],    20≤i≤39
    c[i]*b[i],    40≤i≤59
}

```

2) 算法汇编程序

数据段在内存中为 A、B、C、D 四部分各预留 240 字节的空间，保存运算中的中间变量。最终运算结束时，调用内中断 **break** 转入中断处理子程序，最终显示的值为 D[59]的值。

```

1. .data
2. A:.space 240
3. B:.space 240
4. C:.space 240
5. D:.space 240
6. .text
7. sll $0,$0,0
8. exc:      #exception
9. beq $30,$0,main
10. j exc
11.
12. main:
13. lui $30,0xffff    #enable exception
14. addi $2,$0,0      #a[i]
15. addi $3,$0,1      #b[i]
16. addi $15,$0,0     #c[i]
17. addi $16,$0,0     #d[i]
18. addi $5,$0,4      #counter
19. addi $6,$0,0      #a[i-1]
20. addi $7,$0,1      #b[i-1]
21. addi $10,$0,0     #flag for i<20 || i<40
22. addi $11,$0,240   #sum counts
23. addi $14,$0,3
24. #addi $30,$0,0
25.
26. # 把 0 1 0 0 ($2,...,$16) 分别存入 A B C D
27. lui $27,0x0000
28. addu $27,$27,$0
29. sw $2,A($27)
30. lui $27,0x0000

```

```

31. addu $27,$27,$0
32. sw $3,B($27)
33. lui $27,0x0000
34. addu $27,$27,$0
35. sw $2,C($27)
36. lui $27,0x0000
37. addu $27,$27,$0
38. sw $3,D($27)
39.
40. # 循环
41. loop:
42. ## $5(4) 除以 4 (=i) 存入 $12
43. srl $12,$5,2
44. # $6 加 i. 自此, $6 就是 a[i] 而不是 a[i-1] 了
45. add $6,$6,$12
46. # 把 a[i] 的内容存入 A[i] 中
47. lui $27,0x0000
48. addu $27,$27,$5
49. sw $6,A($27)
50. # $14 (3) 乘以 $5/4 (i) ( = 3i )
51. mul $15,$14,$12
52. # 把 $7 (b[i-1]) 的内容加上 3i, 存入 B[i]. 自此, $7 就是 b[i] 而不是 b[i-1]
53. add $7,$7,$15
54. lui $27,0x0000
55. addu $27,$27,$5
56. sw $7,B($27)
57. # $5 是否小于 80 (i 是否小于 20)? 记入 $10
58. slti $10,$5,80
59. # 若不是, 跳转
60. bne $10,1,c1
61.
62. # (0<=i<=19)
63. # 把 $6 的内容存入 C[i] 中 (c[i] = a[i])
64. lui $27,0x0000
65. addu $27,$27,$5
66. sw $6,C($27)
67. # 把 $7 的内容存入 D[i] 中 (d[i] = b[i])
68. lui $27,0x0000
69. addu $27,$27,$5
70. sw $7,D($27)
71. addi $15,$6,0 # $15 $16 分别赋值为 c[i] d[i]
72. addi $16,$7,0
73. j endc
74. c1: # (20<=i<=39)
75. # i 是否小于 40 ? 若不是, 跳转到 c2
76. slti $10,$5,160
77. addi $27,$0,1
78. bne $10,$27,c2
79. # C[i] = a[i] + b[i]
80. add $15,$6,$7
81. lui $27,0x0000
82. addu $27,$27,$5
83. sw $15,C($27)
84. # D[i] = a[i] * b[i]
85. mul $16,$15,$6
86. lui $27,0x0000
87. addu $27,$27,$5
88. sw $16,D($27)

```

```

89. j endc
90. c2: # (i>=40)
91. # C[i] = a[i] * b[i]
92. mul $15,$6,$7
93. lui $27,0x0000
94. addu $27,$27,$5
95. sw $15,C($27)
96. # D[i] = c[i] * b[i]
97. mul $16,$15,$7
98. lui $27,0x0000
99. addu $27,$27,$5
100. sw $16,D($27)
101.
102. endc:
103. addi $5,$5,4 # i = i + 1
104. bne $5,$11,loop # i = 60 不跳转
105. break
106. # 最后可通过验证 $16 的正确性来验证正确性

```

3) 验证程序结果

MIPS 程序结束位置（Mars 会在内中断出停止，实际中内中断转子程序后在子程序停止）：

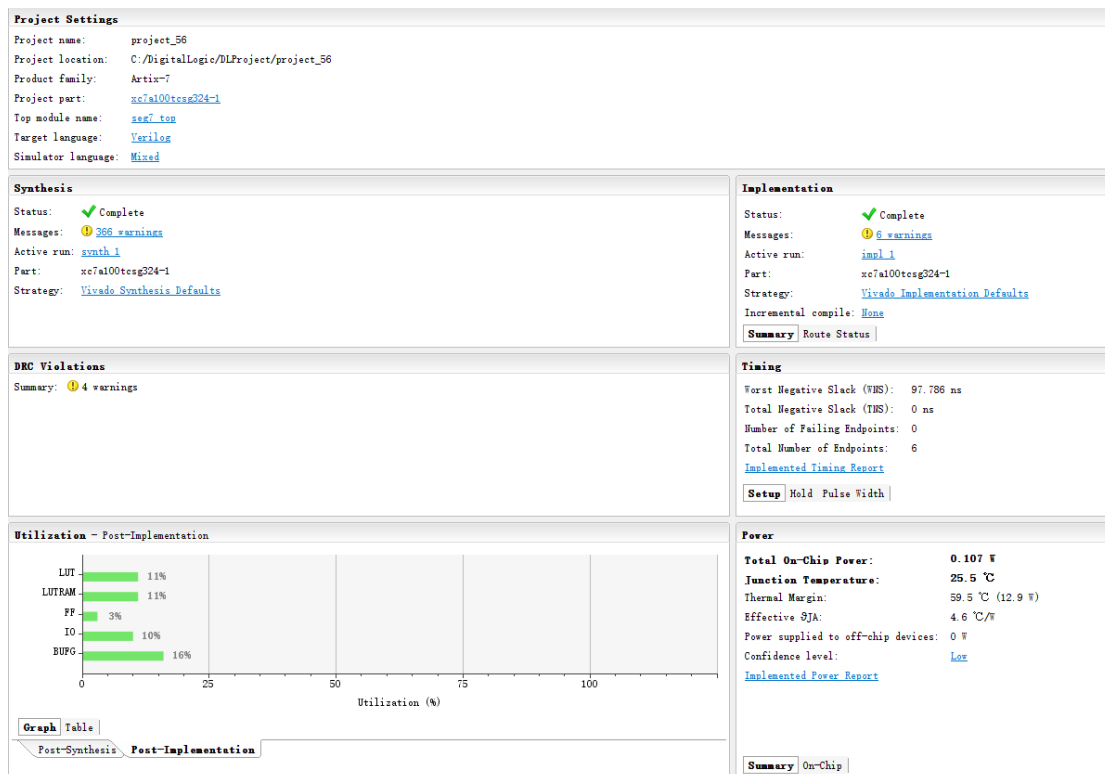
0x00400170	0x20ab0004	addi \$5, \$5, 0x00000004	103: addi \$5, \$5, 4 # i = i + 1
0x00400174	0x14abffc4	bne \$5, \$11, 0xffffffc4	104: bne \$5, \$11, loop # i = 60 消腐烦杞
0x00400178	0x0000000d	break	105: break

MIPS 程序输出：

\$t5	13	0x00000000
\$t6	14	0x00000003
\$t7	15	0x008f7096
\$s0	16	0x9fd0b7ea
\$s1	17	0x00000000

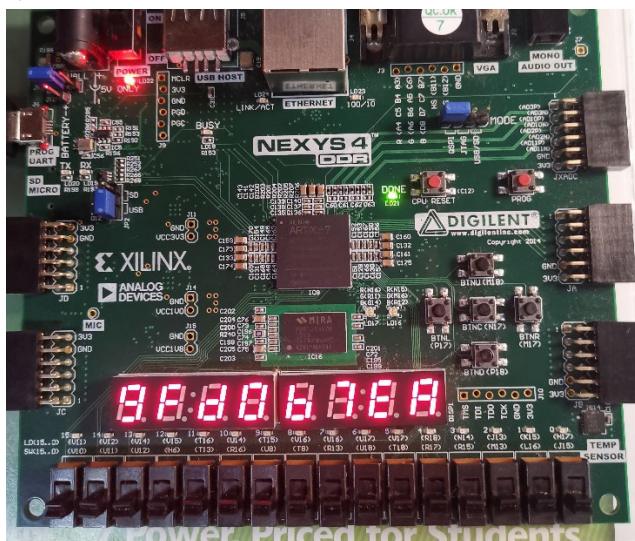
七、实验验算程序下板测试过程与实现

1) 综合、实现、下板的运行结果

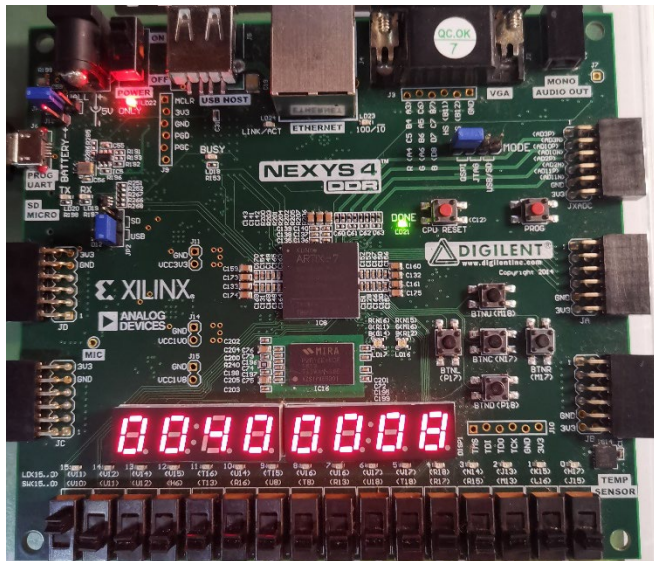


2) 下板结果

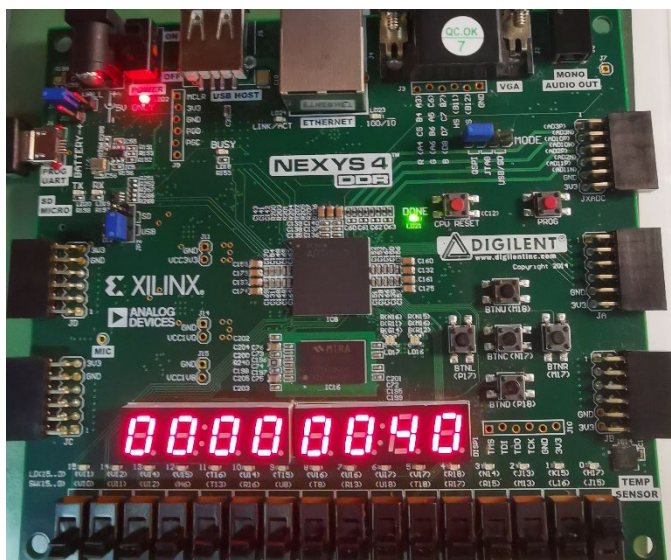
最终结果 9FD0B7EA，结果正确：



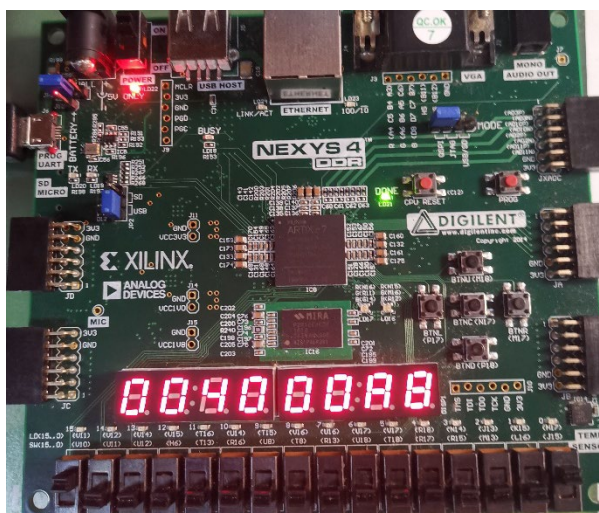
使用最左侧的开关（V10），显示最终的 PC 值，结果正确：



在执行过程中，使用最右侧的开关（J15），外中断暂停 CPU，显示当前的 D[i]值：



使用最左侧的开关（V10），当前的 PC 值：（PC=004000A8 时，EX 段正在计算 mul 指令，占用 32 周期，容易在暂停时刚好执行到这个位置）



八、流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析）

1、 动态流水线的性能指标定性分析

（注：为了明显地显示外中断效果，顶部模块设置分频 CLK_PERIOD=20。
对于本项性能分析，采用经验证可行的最低分频 CLK_PERIOD=3）

吞吐率 $TP=n/T_m=2022/278.936\mu s=7.2490$ (MIPS)

加速比（考虑到单周期乘除法，单周期的周期长度设为流水线周期的 32 倍） $S=T_0/T_m=2022*32*40ns/278.936\mu s=9.2787$

效率 $E=n$ 个任务占用的时空区 / m 个功能段的总的时空区
 $=2022*5*40ns/278.936\mu s*5=0.2900$

2、 冲突分析

动态流水线只会产生先写后读冲突。其中 ID 段取数值，WB 段写回数值，因此要保证 ID 段取值时不会与 EX、ME、WB 段指令产生先写后读冲突。这里使用 raddr 和 waddr 组合逻辑来比较指令是否冲突，定义如下：

```
1. //judge read/write violation
2. `define VIOLATION_REGFILE_HEAD 2'b00
3. `define VIOLATION_CP0REG_HEAD 2'b01
4. `define VIOLATION_HI 7'b100_0000
5. `define VIOLATION_LO 7'b100_0001
6. `define VIOLATION_HILO 7'b100_0010
7. `define VIOLATION_NON 7'b110_0000
```

使用一个 7 位数据表示使用到的寄存器，对于使用通用寄存器和 CP0 寄存器的情况，前两位设置为 2'b00、2'b01，并在后五位填入对应的地址，使用 HI、LO 和不使用的情况对应后四项定义。这里 violation 只涉及 EX 段冲突和 ME 段读内存后写寄存器冲突，因为其他的冲突都可用设置专用路径的方式直接动态数据前推解决，无需暂停。判别逻辑如下。

```
1. wire
  violation1=(waddr1!=`VIOLATION_NON)&&(raddr1==waddr1 || raddr2==waddr1 || (waddr1==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_HI || raddr2==`VIOLATION_LO)));
2. wire
  violation2=(waddr2!=`VIOLATION_NON)&&(raddr1==waddr2 || raddr2==waddr2 || (waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_HI || raddr2==`VIOLATION_LO)));
3. wire
  violation3=(waddr3!=`VIOLATION_NON)&&(raddr1==waddr3 || raddr2==waddr3 || (waddr3==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_HI || raddr2==`VIOLATION_LO)));
4. wire violation=violation1|(violation2&ex_from_mem);
```

violation 发生时，暂停前两段流水段，并使后三段顺序流动一个周期，然后再判别冲突，控制信号如下。

```
1. cond[0]<=`PARTS_COND_STALL; //IF
```



```

2.         cond[1]<=`PARTS_COND_ZERO; //ID
3.         cond[2]<=`PARTS_COND_FLOW; //EX
4.         cond[3]<=`PARTS_COND_FLOW; //ME
5.         cond[4]<=`PARTS_COND_FLOW; //WB
6.         nextstate<=`FLOW_NORMAL;

```

3、 设置专用路径分析

针对前文 violation1、violation2、violation3 中产生冲突而不使用流水暂停的情况，使用设置专用路径的方法解决。id_ALUa_w、id_ALUb_w、id_Rt_w 分别表示对应 ID 段流水寄存器是否需要使用专用路径的值，id_ALUa、id_ALUb、id_Rt 表示这些专用路径。如下为判别地情况：

```

1.     assign
id_ALUa_w=(!violation)&&(raddr1!=`VIOLATION_NON)&&(raddr1==waddr2||raddr1==waddr3
|| (waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO))|| (waddr3==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO)));
2.     assign
id_ALUb_w=(!violation)&&(raddr2!=`VIOLATION_NON)&&(raddr2==waddr2||raddr2==waddr3
);
3.     assign id_Rt_w=id_ALUb_w;

```

对于专用路径中传递的值，由以下几项判断：waddr 决定传送使用的数据在 Rt、HI、LO 中的选择；me_from_mem，选择使用 me_MEM 的值；由于读内存指令在 ME 段后才读到值，因此 EX、ME 段间的流水寄存器不能用于读到的内存数据前推，因此使用 wire violation=violation1|(violation2&ex_from_mem)，流水暂停。判定逻辑如下：

```

1.     always @(*) begin
2.         if(!violation) begin
3.             if(id_ALUa_w) begin //use alua
4.
if(raddr1==waddr2|| (waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO))) begin //use ex
5.                 case (waddr2)
6.                     `VIOLATION_HI: id_ALUa<=ex_HI;
7.                     `VIOLATION_LO: id_ALUa<=ex_LO;
8.                     `VIOLATION_HILO: begin
9.                         if (raddr1==`VIOLATION_HI)
10.                            id_ALUa<=ex_HI;
11.                        else
12.                            id_ALUa<=ex_LO;
13.                    end
14.                    default: begin
15.                        case (waddr2[6:5])
16.                            `VIOLATION_REGFILE_HEAD: id_ALUa<=ex_Z;
17.                            `VIOLATION_CP0REG_HEAD: id_ALUa<=ex_Z;
18.                            default: begin end
19.                        endcase
20.                    end
21.                endcase
22.            end
23.        else begin //use me
24.            if (me_from_mem) begin
25.                id_ALUa<=me_MEM;
26.            end
27.            case (waddr2)

```

```

28.         `VIOLATION_HI: id_ALUa<=me_HI;
29.         `VIOLATION_LO: id_ALUa<=me_LO;
30.         `VIOLATION_HILO: begin
31.             if (raddr1==`VIOLATION_HI)
32.                 id_ALUa<=me_HI;
33.             else
34.                 id_ALUa<=me_LO;
35.             end
36.         default: begin
37.             case (waddr2[6:5])
38.                 `VIOLATION_REGFILE_HEAD: id_ALUa<=me_Z;
39.                 `VIOLATION_CP0REG_HEAD: id_ALUa<=me_Z;
40.                 default: begin end
41.             endcase
42.         end
43.     endcase
44. end
45. end
46.
47.     if (id_ALUb_w) begin //use alub
48.         if (raddr2==waddr2) //use ex
49.             id_ALUb<=ex_mul?ex_LO:ex_Z;
50.         else begin //use me
51.             if (me_from_mem)
52.                 id_ALUb<=me_MEM;
53.             else
54.                 id_ALUb<=me_mul?me_LO:me_Z;
55.             end
56.         end
57.
58.         if (id_Rt_w) begin //use Rt
59.             if (raddr2==waddr2) //use ex
60.                 id_Rt<=ex_mul?ex_LO:ex_Z;
61.             else begin //use me
62.                 if (me_from_mem)
63.                     id_Rt<=me_MEM;
64.                 else
65.                     id_Rt<=me_mul?me_LO:me_Z;
66.                 end
67.             end
68.         end
69.     end

```

九、总结与体会

在本次 CPU 设计实验中，设计并实现了顺序执行的 54 条 MIPS 动态流水线 CPU，支持内中断（break、teq、syscall 等指令）和外中断（按键中断暂停 CPU 运行），支持设置专用路径实现动态数据前推到 id 段，支持推后处理的 id 段流水暂停等待，实现算数操作、乘除法运算、存储器数据传送、内中断、中断返回、跳转、条件跳转指令，并且可侦测读写冲突、溢出错误、乘除法暂停。

相比静态流水线，动态流水线的流水控制模块设计了新的功能专用路径和外中断暂停。由于新功能实现的要求，同时需要修改了乘除法器的时序逻辑、不同流水段的输入输出以满足专用路径的控制，防止产生错误。七段数码管的信号输入输出以适配流水线 CPU 的时序逻辑。

此外，因为测试程序的设计同时要考虑到显示，每个寄存器意义唯一，仅使用一个中间临时变量寄存器，如果将这些寄存器的使用分离开减少读写冲突，CPU 加速比和吞吐率会更高。此时实验中乘除法指令较多，每次计算乘除法程序必须暂停等待 32 周期，其余指令不执行，这影响到了 CPU 的吞吐率和效率。

在本次 CPU 设计实验中，我学习到动态流水线 CPU 的流水处理方式。与静态流水线 CPU 不同，动态流水线由于专用路径和外中断的设立，对时序严谨性提出了更高要求。例如乘除法器时钟沿与流水控制模块时钟沿的冲突，此时就不能像逻辑单元 ALU 一样直接在上升沿写流水寄存器，对于时序模块乘除法器 calculator，写寄存器时机需要重新分配到下降沿，才不与乘除法器和流水控制冲突。专用路径和外中断的设立要求设计 CPU 是对时序逻辑模块的时钟沿更合理的分配，往往出现上下时钟沿冲突的情况，这与之前设计的 CPU 有很大的不同。

十、附件（所有程序）

1、 提交的所有文件

设计源码文件 17 个，分别为：alu.v、calculator.v、controller.v、cp0.v、cpu_regfile.v、cpu_top.v、DIV.v、DIVU.v、extend.v、flow_control.v、MULT.v、MULTU.v、mux.v、parts.v、ram.v、seg7x16.v、seg7_top.v
仿真测试文件：cpu_simulation_tb.v、cpu_synthesis_implementation_tb.v
宏定义文件：define.vh
IP 核文件 3 个，分别为：dist_mem_gen_v8_0.v、imem.mif、imem.v
XDC 管脚约束文件：icf.xdc
BIT 下板文件：seg7_top.bit
验算程序 MIPS 汇编代码：test.asm
验算程序 COE 汇编结果：test.coe

2、 动态流水线的设计程序

1) define.vh

```
1. //-----Instruction-----
2.
3. `define PARTS_COND_FLOW 2'b00
4. `define PARTS_COND_STALL 2'b01
5. `define PARTS_COND_ZERO 2'b10
6. `define PC_ADDR_INIT 32'h00400000
7. `define IR_NON 32'hffffffff
8. `define IR_NON_31_26 6'b111111
9.
10. //-----Operation-----
11.
12. `define CAL_MULTU 2'b00
13. `define CAL_MULT 2'b01
14. `define CAL_DIVU 2'b10
15. `define CAL_DIV 2'b11
16.
```

```

17. //-----ALU-----
18.
19. `define ALU_ADDU 4'b0000
20. `define ALU_ADD 4'b0010
21. `define ALU_SUBU 4'b0001
22. `define ALU_SUB 4'b0011
23. `define ALU_AND 4'b0100
24. `define ALU_OR 4'b0101
25. `define ALU_XOR 4'b0110
26. `define ALU_NOR 4'b0111
27. `define ALU_LUI 4'b1000
28. `define ALU_SLT 4'b1011
29. `define ALU_SLTU 4'b1010
30. `define ALU_SRA 4'b1100
31. `define ALU_SLL 4'b1110
32. `define ALU_SLA 4'b1111
33. `define ALU_SRL 4'b1101
34. `define ALU_CLZ 4'b1001
35.
36. //-----CP0-----
37.
38. `define EXC_SYSCALL 4'b1000
39. `define EXC_BREAK 4'b1001
40. `define EXC_TEQ 4'b1101
41.
42. //-----Extend-----
43.
44. `define EXT5_Z 3'b000
45. `define EXT16_SL2_S 3'b001
46. `define EXT16_Z 3'b010
47. `define EXT16_S 3'b011
48. `define EXT8_Z 3'b100
49. `define EXT8_S 3'b101
50. `define EXT32_NON 3'b110
51.
52. //-----Mux-----
53.
54. `define MUX_PC_NPCEXT 3'b000
55. `define MUX_PC_RS 3'b001
56. `define MUX_PC_INTR_ADDR 3'b010
57. `define MUX_PC_EPC 3'b011
58. `define MUX_PC_CONNECT 3'b100
59. `define MUX_PC_NPC 3'b101
60.
61. `define MUX_ALUa_HI 3'b000
62. `define MUX_ALUa_LO 3'b001
63. `define MUX_ALUa_NPC 3'b010
64. `define MUX_ALUa_RS 3'b011
65. `define MUX_ALUa_EXT 3'b100
66. `define MUX_ALUa_CP0 3'b101
67. `define MUX_ALUa_IMM0 3'b110
68.
69. `define MUX_ALUb_IMM0 2'b00
70. `define MUX_ALUb_RT 2'b01
71. `define MUX_ALUb_EXT 2'b10
72.
73. `define MUX_RDC_IR_15_11 2'b00
74. `define MUX_RDC_IR_20_16 2'b01

```

```

75. `define MUX_RDC_IMM31 2'b10
76.
77. `define MUX_RD_Z 2'b00
78. `define MUX_RD_MEM 2'b01
79. `define MUX_RD_HI 2'b10
80. `define MUX_RD_LO 2'b11
81.
82. //-----RAM-----
83.
84. `define RAM_WIDTH_32 2'b00
85. `define RAM_WIDTH_16 2'b01
86. `define RAM_WIDTH_8 2'b10
87.
88. //-----CtrlUnit-----
89.
90. `define FLOW_NORMAL 2'b00
91. `define FLOW_OVERFLOW 2'b01
92. `define FLOW_MULDIV 2'b10
93. `define FLOW_VIOLATE 2'b11
94.
95. //-----Others-----
96.
97. //judge read/write violation
98. `define VIOLATION_REGFILE_HEAD 2'b00
99. `define VIOLATION_CP0REG_HEAD 2'b01
100. `define VIOLATION_HI 7'b100_0000
101. `define VIOLATION_LO 7'b100_0001
102. `define VIOLATION_HILO 7'b100_0010
103. `define VIOLATION_NON 7'b110_0000

```

2) alu.v

```

1. `include "define.vh"
2.
3. module alu(
4.     input [31:0] a,
5.     input [31:0] b,
6.     input [3:0] aluc,
7.     output reg [31:0] r,
8.     output reg zero,
9.     output reg carry,
10.    output reg negative,
11.    output reg overflow
12. );
13.    reg [1:0] carry2;
14.    reg [31:0] tmp;
15.    always@(*)
16.    begin
17.        case(aluc)
18.            `ALU_ADDU:    //unsigned +
19.            begin
20.                {carry,r[31:0]}={1'b0,a}+{1'b0,b};
21.                zero=(0==r)?1:0;
22.                negative=r[31];
23.            end
24.            `ALU_ADD:    //signed +
25.            begin
26.                {carry2[0],r}={1'b0,a}+{1'b0,b};
27.                zero=(0==r)?1:0;

```

```

28.         negative=r[31];
29.         overflow=(r[31]+a[31]+b[31])^carry2[0];
30.     end
31. `ALU_SUBU:    //unsigned -
32. begin
33.     {carry,r[31:0]}={1'b0,a}-{1'b0,b};
34.     zero=(0==r)?1:0;
35.     negative=r[31];
36. end
37. `ALU_SUB:    //signed -
38. begin
39.     {carry2[0],r}={1'b0,a}-{1'b0,b};
40.     zero=(0==r)?1:0;
41.     negative=r[31];
42.     overflow=(r[31]+a[31]+b[31])^carry2[0];
43. end
44.
45. `ALU_AND:    //and
46. begin
47.     r=a&b;
48.     zero=(0==r)?1:0;
49.     negative=r[31];
50. end
51. `ALU_OR:    //or
52. begin
53.     r=a|b;
54.     zero=(0==r)?1:0;
55.     negative=r[31];
56. end
57. `ALU_XOR:    //xor
58. begin
59.     r=a^b;
60.     zero=(0==r)?1:0;
61.     negative=r[31];
62. end
63. `ALU_NOR:    //nor
64. begin
65.     r=~(a|b);
66.     zero=(0==r)?1:0;
67.     negative=r[31];
68. end
69.
70. `ALU_LUI:    //lui
71. begin
72.     r={b[15:0],16'b0};
73.     zero=(0==r)?1:0;
74.     negative=r[31];
75. end
76. `ALU_SLT:    //signed <
77. begin
78.     {carry2[0],tmp}={a[31],a}-{b[31],b};
79.     zero=(a==b)?1:0;
80.     negative=carry2[0];
81.     r={31'b0,negative};
82. end
83. `ALU_SLTU:    //unsigned <
84. begin
85.     carry=(a<b)?1:0;

```

```

86.         zero=(a==b)?1:0;
87.         negative=0;
88.         r={31'b0,carry};
89.     end
90.
91. `ALU_SRA:    //shiftR arithmetic
92. begin
93.     if(a[4:0]>0)
94.     begin
95.         carry=b[a[4:0]-1];
96.         tmp={32{b[31]}};
97.         r={tmp,b}>>a[4:0];
98.     end
99.     else
100.    begin
101.        carry=b[0];
102.        r=b;    //fix
103.    end
104.    zero=(0==r)?1:0;
105.    negative=r[31];
106. end
107. `ALU_SLL,`ALU_SLA:    //shiftL
108. begin
109.     if(a[4:0]>0)
110.         carry=b[32-a[4:0]];
111.     else
112.     begin
113.         carry=b[31];
114.         r=b;    //fix
115.     end
116.     r=b<<a[4:0];
117.     zero=(0==r)?1:0;
118.     negative=r[31];
119. end
120. `ALU_SRL:    //shiftR logic
121. begin
122.     if(a[4:0]>0)
123.         carry=b[a[4:0]-1];
124.     else
125.     begin
126.         carry=b[0];
127.         r=b;    //fix
128.     end
129.     r=b>>a[4:0];
130.     zero=(0==r)?1:0;
131.     negative=r[31];
132. end
133. `ALU_CLZ:    //count leading zero
134. begin
135.     if(a==32'b0) begin
136.         r=32'd32;
137.     end
138.     else begin
139.         r[31:5]=27'b0;
140.         tmp[31:0]=a;
141.         if(tmp[31:16]==16'b0) begin
142.             r[4]=1;
143.         end

```

```

144.         else begin
145.             r[4]=0;
146.             tmp[15:0]=tmp[31:16];
147.         end
148.         if(tmp[15:8]==8'b0) begin
149.             r[3]=1;
150.         end
151.         else begin
152.             r[3]=0;
153.             tmp[7:0]=tmp[15:8];
154.         end
155.         if(tmp[7:4]==4'b0) begin
156.             r[2]=1;
157.         end
158.         else begin
159.             r[2]=0;
160.             tmp[3:0]=tmp[7:4];
161.         end
162.         if(tmp[3:2]==2'b0) begin
163.             r[1]=1;
164.         end
165.         else begin
166.             r[1]=0;
167.             tmp[1:0]=tmp[3:2];
168.         end
169.         r[0]=(tmp[1]==1'b0)?1:0;
170.     end
171. end
172. default:begin end
173. endcase
174. end
175. endmodule

```

3) calculator.v

```

1. `include "define.vh"
2.
3. module calculator(
4.     input clk,
5.     input [31:0] a, //multiplicand/dividend
6.     input [31:0] b, //multiplier/divisor
7.     input [1:0] calc,
8.     input reset,    //high-active,at the beginning of test
9.     input ena,      //high-active
10.    output reg [31:0] oLO,
11.    output reg [31:0] oHI,
12.    output reg sum_finish,
13.
14.    //for program in
15.    input cpu_stall
16. );
17.
18. wire cal_clk=~clk;
19. reg start,busy;
20. wire
    busyDivu,busyDiv,busyMult,busyMultu,finishDivu,finishDiv,finishMult,finishMultu;
21. wire [63:0] oMult,oMultu,oDiv,oDivu;
22.
23. reg [1:0] inner_calc;

```



```

24.     always @(negedge clk) begin
25.         inner_calc<=calc;
26.     end
27.
28.     always @(*) begin
29.         case (inner_calc)
30.             `CAL_MULTU: begin
31.                 oLO<=oMultu[31:0];
32.                 oHI<=oMultu[63:32];
33.                 sum_finish<=finishMultu;
34.             end
35.             `CAL_MULT: begin
36.                 oLO<=oMult[31:0];
37.                 oHI<=oMult[63:32];
38.                 sum_finish<=finishMult;
39.             end
40.             `CAL_DIVU: begin
41.                 oLO<=oDivu[31:0];
42.                 oHI<=oDivu[63:32];
43.                 sum_finish<=finishDivu;
44.             end
45.             `CAL_DIV: begin
46.                 oLO<=oDiv[31:0];
47.                 oHI<=oDiv[63:32];
48.                 sum_finish<=finishDiv;
49.             end
50.             default: begin end
51.         endcase
52.     end
53.
54.     always @(*) begin
55.         case (calc)
56.             `CAL_MULTU: begin
57.                 busy<=busyMultu;
58.             end
59.             `CAL_MULT: begin
60.                 busy<=busyMult;
61.             end
62.             `CAL_DIVU: begin
63.                 busy<=busyDivu;
64.             end
65.             `CAL_DIV: begin
66.                 busy<=busyDiv;
67.             end
68.             default: begin end
69.         endcase
70.     end
71.
72.     always @(posedge ena or posedge busy or posedge reset) begin
73.         if (reset) begin
74.             start<=0;
75.         end else begin
76.             if (busy)
77.                 start<=0;
78.             else if(ena)
79.                 start<=1;
80.         end
81.     end

```

```

82.
83.     (* KEEP = "{TRUE|FALSE|SOFT}" *) wire
      start_mult,start_multu,start_div,start_divu;
84.     assign start_mult=start&&(calc==`CAL_MULT);
85.     assign start_multu=start&&(calc==`CAL_MULTU);
86.     assign start_div=start&&(calc==`CAL_DIV);
87.     assign start_divu=start&&(calc==`CAL_DIVU);
88.     MULT MULT_inst(
89.         .clk(cal_clk),
90.         .reset(reset),    //active high
91.         .start(start_mult),
92.         .a(a), //multiplicand
93.         .b(b), //multiplier
94.         .z(oMult),
95.         .busy(busyMult),
96.         .finish(finishMult),
97.         .cpu_stall(cpu_stall)
98.     );
99.     MULTU MULTU_inst(
100.         .clk(cal_clk),
101.         .reset(reset),    //active high
102.         .start(start_multu),
103.         .a(a), //multiplicand
104.         .b(b), //multiplier
105.         .z(oMultu),
106.         .busy(busyMultu),
107.         .finish(finishMultu),
108.         .cpu_stall(cpu_stall)
109.     );
110.     DIV DIV_inst(
111.         .dividend(a),
112.         .divisor(b),
113.         .start(start_div),
114.         .clock(cal_clk),
115.         .reset(reset),    //active-high
116.         .q(oDiv[31:0]),
117.         .r(oDiv[63:32]),
118.         .busy(busyDiv),
119.         .finish(finishDiv),
120.         .cpu_stall(cpu_stall)
121.     );
122.     DIVU DIVU_inst(
123.         .dividend(a),
124.         .divisor(b),
125.         .start(start_divu),
126.         .clock(cal_clk),
127.         .reset(reset),    //active-high
128.         .q(oDivu[31:0]),
129.         .r(oDivu[63:32]),
130.         .busy(busyDivu),
131.         .finish(finishDivu),
132.         .cpu_stall(cpu_stall)
133.     );
134. endmodule

```

4) controller.v

```

1. `include "define.vh"
2.

```

```

3. module controller(
4.     input [31:0] inst,
5.     input judge_beq, //judge BEQ BNE condition to jump
6.     input judge_bgez, //judge BGEZ condition to jump
7.
8.     /*-----flow_control-----*/
9.     output reg [6:0] raddr1,output reg [6:0] raddr2,output reg [6:0] waddr,output reg
dmem_r,
10.
11.     /*-----control-----*/
12.     //ID
13.     output reg [2:0] mux_pc_sel,
14.     output reg [2:0] mux_ALUa_sel,output reg [1:0] mux_ALUb_sel,output reg [2:0]
ext_sel,
15.     output reg cp0_exception,output reg cp0_eret,output reg [4:0] cp0_cause,
16.     //EX
17.     output reg [3:0] alu_sel,output reg [1:0] cal_sel,output reg cal_ena/*also for
stall*/,output reg use_overflow,
18.     //ME
19.     output reg dmem_w,output reg [1:0] dmem_width,output reg [2:0] mem_sel,
20.     //WB
21.     output reg [1:0] mux_Rdc_sel,output reg [1:0] mux_Rd_sel,
22.     output reg hi_w,output reg lo_w,output reg regfile_w,output reg cp0_w
23. );
24.
25. always @(*) begin
26.     case (inst[31:26])
27.         `IR_NON_31_26: begin //do nothing
28.             //FLOW
29.
raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
30.             //IF
31.             mux_pc_sel<=`MUX_PC_NPC;
32.             //ID
33.
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
34.             cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
35.             //EX
36.
alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
37.             dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
38.             //WB
39.             mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
40.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
41.         end
42.         6'b000000: begin
43.             case (inst[5:0])
44.                 6'b100000: begin //add
45.                     //FLOW
46.
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]];raddr2<={`VIOLATION_REGFILE_HEAD,in
st[20:16]];waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]];dmem_r<=1'b0;
47.                     //IF
48.                     mux_pc_sel<=`MUX_PC_NPC;
49.                     //ID
50.
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
51.                     cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;

```

```

52.                //EX
53.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
54.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
55.                //WB
56.                mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
57.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
58.            end
59.            6'b100001: begin    //addu
60.                //FLOW
61.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
62.                //IF
63.                mux_pc_sel<=`MUX_PC_NPC;
64.                //ID
65.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
66.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
67.                //EX
68.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
69.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
70.                //WB
71.                mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
72.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
73.            end
74.            6'b100010: begin    //sub
75.                //FLOW
76.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
77.                //IF
78.                mux_pc_sel<=`MUX_PC_NPC;
79.                //ID
80.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
81.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
82.                //EX
83.
    alu_sel<=`ALU_SUB;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
84.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
85.                //WB
86.                mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
87.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
88.            end
89.            6'b100011: begin    //subu
90.                //FLOW
91.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
92.                //IF
93.                mux_pc_sel<=`MUX_PC_NPC;
94.                //ID
95.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
96.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
97.                //EX

```

```

98.     alu_sel<=`ALU_SUBU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
99.         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
100.            //WB
101.            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
102.            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
103.        end
104.        6'b100100: begin    //and
105.            //FLOW
106.
107.            raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
108.            st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
109.            //IF
110.            mux_pc_sel<=`MUX_PC_NPC;
111.            //ID
112.            mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
113.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
114.            //EX
115.            alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
116.            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
117.            //WB
118.            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
119.            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
120.        end
121.        6'b100101: begin    //or
122.            //FLOW
123.
124.            raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
125.            st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
126.            //IF
127.            mux_pc_sel<=`MUX_PC_NPC;
128.            //ID
129.            mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
130.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
131.            //EX
132.            alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
133.            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
134.            //WB
135.            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
136.            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
137.        end
138.        6'b100110: begin    //xor
139.            //FLOW
140.
141.            raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
142.            st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
143.            //IF
144.            mux_pc_sel<=`MUX_PC_NPC;
145.            //ID
146.            mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
147.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
148.            //EX

```

```

143.    alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
144.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
145.        //WB
146.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
147.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
148.    end
149.    6'b100111: begin    //nor
150.        //FLOW
151.
152.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
153.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
154.        //IF
155.        mux_pc_sel<=`MUX_PC_NPC;
156.        //ID
157.
158.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
159.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
160.        //EX
161.
162.        alu_sel<=`ALU_NOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
163.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
164.        //WB
165.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
166.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
167.    end
168.    6'b101010: begin    //slt
169.        //FLOW
170.
171.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
172.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
173.        //IF
174.        mux_pc_sel<=`MUX_PC_NPC;
175.        //ID
176.
177.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
178.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
179.        //EX
180.
181.        alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
182.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
183.        //WB
184.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
185.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
186.    end
187.    6'b101011: begin    //sltu
188.        //FLOW
189.
190.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
191.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
192.        //IF
193.        mux_pc_sel<=`MUX_PC_NPC;
194.        //ID
195.
196.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
197.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
198.        //EX

```

```

188.    alu_sel<=`ALU_SLTU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
189.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
190.        //WB
191.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
192.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
193.    end
194.        6'b000000: begin    //sll
195.            //FLOW
196.
197.            raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
LATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
198.            //IF
199.            mux_pc_sel<=`MUX_PC_NPC;
200.            //ID
201.            mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
202.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
203.            //EX
204.            alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
205.            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
206.            //WB
207.            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
208.            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
209.        end
210.        6'b000010: begin    //sr1
211.            //FLOW
212.
213.            raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
LATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
214.            //IF
215.            mux_pc_sel<=`MUX_PC_NPC;
216.            //ID
217.            mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
218.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
219.            //EX
220.            alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
221.            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
222.            //WB
223.            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
224.            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
225.        end
226.        6'b000011: begin    //sra
227.            //FLOW
228.
229.            raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
LATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
230.            //IF
231.            mux_pc_sel<=`MUX_PC_NPC;
232.            //ID
233.            mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
234.            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
235.            //EX

```

```

233.    alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
234.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
235.        //WB
236.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
237.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
238.    end
239.    6'b000100: begin    //sllv
240.        //FLOW
241.
242.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
243.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
244.        //IF
245.        mux_pc_sel<=`MUX_PC_NPC;
246.        //ID
247.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
248.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
249.        //EX
250.        alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
251.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
252.        //WB
253.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
254.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
255.    end
256.    6'b000110: begin    //srlv
257.        //FLOW
258.
259.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
260.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
261.        //IF
262.        mux_pc_sel<=`MUX_PC_NPC;
263.        //ID
264.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
265.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
266.        //EX
267.        alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
268.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
269.        //WB
270.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
271.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
272.    end
273.    6'b000111: begin    //srav
274.        //FLOW
275.
276.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
277.        st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
278.        //IF
279.        mux_pc_sel<=`MUX_PC_NPC;
280.        //ID
281.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
282.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
283.        //EX

```



```

278.    alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
279.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
280.        //WB
281.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
282.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
283.    end
284.    6'b001000: begin    //jr
285.        //FLOW
286.
287.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
288.        ATION_NON;dmem_r<=1'b0;
289.        //IF
290.        mux_pc_sel<=`MUX_PC_RS;
291.        //ID
292.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
293.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
294.        //EX
295.        alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
296.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
297.        //WB
298.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
299.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
300.    end
301.    6'b011010: begin    //div
302.        //FLOW
303.
304.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
305.        st[20:16]};waddr<=`VIOLATION_HILO;dmem_r<=1'b0;
306.        //IF
307.        mux_pc_sel<=`MUX_PC_NPC;
308.        //ID
309.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
310.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
311.        //EX
312.        alu_sel<=`ALU_SLL;cal_sel<=`CAL_DIV;cal_ena<=1'b1;use_overflow<=1'b0;
313.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
314.        //WB
315.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
316.        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
317.    end
318.    6'b011011: begin    //divu
319.        //FLOW
320.
321.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
322.        st[20:16]};waddr<=`VIOLATION_HILO;dmem_r<=1'b0;
323.        //IF
324.        mux_pc_sel<=`MUX_PC_NPC;
325.        //ID
326.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
327.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
328.        //EX

```

```

323.    alu_sel<='ALU_SLL;cal_sel<='CAL_DIVU;cal_ena<=1'b1;use_overflow<=1'b0;
324.        dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
325.        //WB
326.        mux_Rdc_sel<='MUX_RDC_IR_15_11;mux_Rd_sel<='MUX_RD_HI;
327.        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
328.    end
329.    6'b011001: begin    //multu
330.        //FLOW
331.
332.        raddr1<={'VIOLATION_REGFILE_HEAD,instr[25:21]};raddr2<={'VIOLATION_REGFILE_HEAD,in
333.        str[20:16]};waddr<='VIOLATION_HILO;dmem_r<=1'b0;
334.        //IF
335.        mux_pc_sel<='MUX_PC_NPC;
336.        //ID
337.        mux_ALUa_sel<='MUX_ALUa_RS;mux_ALUb_sel<='MUX_ALUb_RT;ext_sel<='EXT5_Z;
338.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
339.        //EX
340.        alu_sel<='ALU_SLL;cal_sel<='CAL_MULTU;cal_ena<=1'b1;use_overflow<=1'b0;
341.        dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
342.        //WB
343.        mux_Rdc_sel<='MUX_RDC_IR_15_11;mux_Rd_sel<='MUX_RD_HI;
344.        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
345.    end
346.    6'b001001: begin    //jarl
347.        //FLOW
348.
349.        raddr1<={'VIOLATION_REGFILE_HEAD,instr[25:21]};raddr2<='VIOLATION_NON;waddr<={'VIO
350.        LATION_REGFILE_HEAD,instr[15:11]};dmem_r<=1'b0;
351.        //IF
352.        mux_pc_sel<='MUX_PC_RS;
353.        //ID
354.        mux_ALUa_sel<='MUX_ALUa_NPC;mux_ALUb_sel<='MUX_ALUb_IMM0;ext_sel<='EXT5_Z;
355.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
356.        //EX
357.        alu_sel<='ALU_ADDU;cal_sel<='CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
358.        dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
359.        //WB
360.        mux_Rdc_sel<='MUX_RDC_IR_15_11;mux_Rd_sel<='MUX_RD_Z;
361.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
362.    end
363.    6'b001101: begin    //break
364.        //FLOW
365.
366.        raddr1<={'VIOLATION_CP0REG_HEAD,5'd12};raddr2<='VIOLATION_NON;waddr<='VIOLATION_N
367.        ON;dmem_r<=1'b0;
368.        //IF
369.        mux_pc_sel<='MUX_PC_INTR_ADDR;
370.        //ID
371.        mux_ALUa_sel<='MUX_ALUa_RS;mux_ALUb_sel<='MUX_ALUb_RT;ext_sel<='EXT5_Z;
372.        cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
373.        //EX

```

```

368.      alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
369.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
370.          //WB
371.          mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
372.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
373.      end
374.      6'b001100: begin    //syscall
375.          //FLOW
376.
377.          raddr1<={`VIOLATION_CP0REG_HEAD,5'd12};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_N
378.          ON;dmem_r<=1'b0;
379.          //IF
380.          mux_pc_sel<=`MUX_PC_INTR_ADDR;
381.          //ID
382.          mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
383.          cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<=`EXC_SYSCALL;
384.          //EX
385.
386.      alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
387.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
388.          //WB
389.          mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
390.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
391.      end
392.      6'b010000: begin    //mfhi
393.          //FLOW
394.
395.      raddr1<=`VIOLATION_HI;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst
396.      [15:11]};dmem_r<=1'b0;
397.      //IF
398.      mux_pc_sel<=`MUX_PC_NPC;
399.      //ID
400.      mux_ALUa_sel<=`MUX_ALUa_HI;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
401.      cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
402.      //EX
403.
404.      alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
405.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
406.          //WB
407.          mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
408.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
409.      end
410.      6'b010010: begin    //mflo
411.          //FLOW
412.
413.      raddr1<=`VIOLATION_LO;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst
414.      [15:11]};dmem_r<=1'b0;
415.      //IF
416.      mux_pc_sel<=`MUX_PC_NPC;
417.      //ID
418.      mux_ALUa_sel<=`MUX_ALUa_LO;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
419.      cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
420.      //EX

```

```

413.    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
414.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
415.        //WB
416.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
417.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
418.    end
419.    6'b010001: begin    //mthi
420.        //FLOW
421.
422.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
423.        ATION_HI;dmem_r<=1'b0;
424.        //IF
425.        mux_pc_sel<=`MUX_PC_NPC;
426.        //ID
427.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
428.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
429.        //EX
430.        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
431.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
432.        //WB
433.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
434.        hi_w<=1'b1;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
435.    end
436.    6'b010011: begin    //mtlo
437.        //FLOW
438.
439.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
440.        ATION_LO;dmem_r<=1'b0;
441.        //IF
442.        mux_pc_sel<=`MUX_PC_NPC;
443.        //ID
444.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
445.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
446.        //EX
447.        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
448.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
449.        //WB
450.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
451.        hi_w<=1'b0;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
452.    end
453.    6'b110100: begin    //teq
454.        //FLOW
455.
456.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
457.        st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
458.        //IF
459.        mux_pc_sel<=`MUX_PC_INTR_ADDR;
460.        //ID
461.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
462.        cp0_exception<=judge_beq;cp0_eret<=1'b0;cp0_cause<=`EXC_TEQ;
463.        //EX

```

```

458.     alu_sel<='ALU_ADDU;cal_sel<='CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
459.         dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
460.         //WB
461.         mux_Rdc_sel<='MUX_RDC_IMM31;mux_Rd_sel<='MUX_RD_Z;
462.         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
463.     end
464.     default: begin //do nothing
465.         //FLOW
466.
467.         raddr1<='VIOLATION_NON;raddr2<='VIOLATION_NON;waddr<='VIOLATION_NON;dmem_r<=1'b0;
468.         //IF
469.         mux_pc_sel<='MUX_PC_NPC;
470.         //ID
471.         mux_ALUa_sel<='MUX_ALUa_RS;mux_ALUb_sel<='MUX_ALUb_RT;ext_sel<='EXT5_Z;
472.         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
473.         //EX
474.         alu_sel<='ALU_ADDU;cal_sel<='CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
475.         dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
476.         //WB
477.         mux_Rdc_sel<='MUX_RDC_IMM31;mux_Rd_sel<='MUX_RD_Z;
478.         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
479.     end
480. endcase
481. 6'b001000: begin //addi
482.     //FLOW
483.
484.     raddr1<={'VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<='VIOLATION_NON;waddr<={'VIO
485.     LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
486.     //IF
487.     mux_pc_sel<='MUX_PC_NPC;
488.     //ID
489.     mux_ALUa_sel<='MUX_ALUa_RS;mux_ALUb_sel<='MUX_ALUb_EXT;ext_sel<='EXT16_S;
490.     cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
491.     //EX
492.     alu_sel<='ALU_ADD;cal_sel<='CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
493.     dmem_w<=1'b0;dmem_width<='RAM_WIDTH_16;mem_sel<='EXT8_Z;
494.     //WB
495.     mux_Rdc_sel<='MUX_RDC_IR_20_16;mux_Rd_sel<='MUX_RD_Z;
496.     hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
497.     end
498.     6'b001001: begin //addiu
499.         //FLOW
500.
501.         raddr1<={'VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<='VIOLATION_NON;waddr<={'VIO
502.         LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
503.         //IF
504.         mux_pc_sel<='MUX_PC_NPC;
505.         //ID
506.         mux_ALUa_sel<='MUX_ALUa_RS;mux_ALUb_sel<='MUX_ALUb_EXT;ext_sel<='EXT16_S;
507.         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<='EXC_BREAK;
508.         //EX

```

```

505.      alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
506.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
507.          //WB
508.          mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
509.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
510.      end
511.      6'b001100: begin    //andi
512.          //FLOW
513.
          raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
          LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
514.          //IF
515.          mux_pc_sel<=`MUX_PC_NPC;
516.          //ID
517.
          mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
518.          cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
519.          //EX
520.
          alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
521.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
522.          //WB
523.          mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
524.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
525.      end
526.      6'b001101: begin    //ori
527.          //FLOW
528.
          raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
          LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
529.          //IF
530.          mux_pc_sel<=`MUX_PC_NPC;
531.          //ID
532.
          mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
533.          cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
534.          //EX
535.
          alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
536.          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
537.          //WB
538.          mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
539.          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
540.      end
541.      6'b001110: begin    //xori
542.          //FLOW
543.
          raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
          LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
544.          //IF
545.          mux_pc_sel<=`MUX_PC_NPC;
546.          //ID
547.
          mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
548.          cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
549.          //EX

```

```

550.    alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
551.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
552.        //WB
553.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
554.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
555.    end
556.    6'b001111: begin    //lui
557.        //FLOW
558.
559.        raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,in
560.        t[20:16]};dmem_r<=1'b0;
561.        //IF
562.        mux_pc_sel<=`MUX_PC_NPC;
563.        //ID
564.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
565.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
566.        //EX
567.        alu_sel<=`ALU_LUI;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
568.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
569.        //WB
570.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
571.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
572.    end
573.    6'b100011: begin    //lw
574.        //FLOW
575.
576.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
577.        LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
578.        //IF
579.        mux_pc_sel<=`MUX_PC_NPC;
580.        //ID
581.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
582.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
583.        //EX
584.        alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
585.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
586.        //WB
587.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
588.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
589.    end
590.    6'b101011: begin    //sw
591.        //FLOW
592.
593.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
594.        st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
595.        //IF
596.        mux_pc_sel<=`MUX_PC_NPC;
597.        //ID
598.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
599.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
600.        //EX

```

```

595.    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
596.        dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
597.        //WB
598.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
599.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
600.    end
601.    6'b000100: begin    //beq
602.        //FLOW
603.
        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
        st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
604.        //IF
605.        mux_pc_sel<=judge_beq?`MUX_PC_NPCEXT:`MUX_PC_NPC;
606.        //ID
607.
        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
608.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
609.        //EX
610.
        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
611.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
612.        //WB
613.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
614.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
615.    end
616.    6'b000101: begin    //bne
617.        //FLOW
618.
        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
        st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
619.        //IF
620.        mux_pc_sel<=judge_beq?`MUX_PC_NPC:`MUX_PC_NPCEXT;
621.        //ID
622.
        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
623.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
624.        //EX
625.
        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
626.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
627.        //WB
628.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
629.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
630.    end
631.    6'b001010: begin    //slti
632.        //FLOW
633.
        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
        LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
634.        //IF
635.        mux_pc_sel<=`MUX_PC_NPC;
636.        //ID
637.
        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
638.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
639.        //EX

```



```

640.    alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
641.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
642.        //WB
643.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
644.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
645.    end
646.    6'b001011: begin    //sltiu
647.        //FLOW
648.
        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
        LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
649.        //IF
650.        mux_pc_sel<=`MUX_PC_NPC;
651.        //ID
652.
        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
653.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
654.        //EX
655.
        alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
656.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
657.        //WB
658.        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
659.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
660.    end
661.    6'b000010: begin    //j
662.        //FLOW
663.
        raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
664.        //IF
665.        mux_pc_sel<=`MUX_PC_CONNECT;
666.        //ID
667.
        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
668.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
669.        //EX
670.
        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
671.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
672.        //WB
673.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
674.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
675.    end
676.    6'b000011: begin    //jal
677.        //FLOW
678.
        raddr1<={`VIOLATION_REGFILE_HEAD,5'd31};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_
        NON;dmem_r<=1'b0;
679.        //IF
680.        mux_pc_sel<=`MUX_PC_CONNECT;
681.        //ID
682.
        mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
683.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
684.        //EX
685.
        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;

```

```

686.             dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
687.             //WB
688.             mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
689.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
690.         end
691.         6'b000001: begin    //bgez
692.             //FLOW
693.
694.             raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
695.             ATION_NON;dmem_r<=1'b0;
696.             //IF
697.             mux_pc_sel<=judge_bgez?`MUX_PC_NPCEXT:`MUX_PC_NPC;
698.             //ID
699.             mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT16_SL2_S;
700.             cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
701.             //EX
702.             alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
703.             dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
704.             //WB
705.             mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
706.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
707.         end
708.         6'b100100: begin    //lbu
709.             //FLOW
710.
711.             raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
712.             LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
713.             //IF
714.             mux_pc_sel<=`MUX_PC_NPC;
715.             //ID
716.             mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
717.             cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
718.             //EX
719.             alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
720.             dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_Z;
721.             //WB
722.             mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
723.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
724.         end
725.         6'b100101: begin    //lhu
726.             //FLOW
727.
728.             raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
729.             LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
730.             //IF
731.             mux_pc_sel<=`MUX_PC_NPC;
732.             //ID
733.             mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
734.             cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
735.             //EX
736.             alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
737.             dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_Z;

```

```

732.                //WB
733.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
734.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
735.            end
736.            6'b100000: begin    //lb
737.                //FLOW
738.
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
                LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
739.                //IF
740.                mux_pc_sel<=`MUX_PC_NPC;
741.                //ID
742.
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
743.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
744.                //EX
745.
                alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
746.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
747.                //WB
748.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
749.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
750.            end
751.            6'b100001: begin    //lh
752.                //FLOW
753.
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
                LATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
754.                //IF
755.                mux_pc_sel<=`MUX_PC_NPC;
756.                //ID
757.
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
758.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
759.                //EX
760.
                alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
761.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
762.                //WB
763.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
764.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
765.            end
766.            6'b101000: begin    //sb
767.                //FLOW
768.
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
                st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
769.                //IF
770.                mux_pc_sel<=`MUX_PC_NPC;
771.                //ID
772.
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
773.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
774.                //EX
775.
                alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
776.                dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
777.                //WB

```

```

778.             mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
779.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
780.         end
781.         6'b101001: begin    //sh
782.             //FLOW
783.
784.             raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
785.             st[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
786.             //IF
787.             mux_pc_sel<=`MUX_PC_NPC;
788.             //ID
789.             mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
790.             cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
791.             //EX
792.             alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
793.             dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
794.             //WB
795.             mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
796.             hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
797.         end
798.         6'b010000: begin
799.             case (inst[25:21])
800.                 5'b10000: begin    //eret
801.                     //FLOW
802.
803.                     raddr1<={`VIOLATION_CP0REG_HEAD,5'd14};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_N
804.                     ON;dmem_r<=1'b0;
805.                     //IF
806.                     mux_pc_sel<=`MUX_PC_EPC;
807.                     //ID
808.                     mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
809.                     cp0_exception<=1'b0;cp0_eret<=1'b1;cp0_cause<=`EXC_BREAK;
810.                     //EX
811.                     alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
812.                     dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
813.                     //WB
814.                     mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
815.                     hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
816.                 end
817.                 5'b00000: begin    //mfc0
818.                     //FLOW
819.
820.                     raddr1<={`VIOLATION_CP0REG_HEAD,inst[15:11]};raddr2<=`VIOLATION_NON;waddr<={`VIOL
821.                     ATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
822.                     //IF
823.                     mux_pc_sel<=`MUX_PC_NPC;
824.                     //ID
825.                     mux_ALUa_sel<=`MUX_ALUa_CP0;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
826.                     cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
827.                     //EX
828.                     alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
829.                     dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;

```

```

824.                //WB
825.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
826.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
827.            end
828.            5'b00100: begin    //mtc0
829.                //FLOW
830.
831.                raddr1<={`VIOLATION_REGFILE_HEAD,inst[20:16]};raddr2<=`VIOLATION_NON;waddr<={`VIO
LATION_CP0REG_HEAD,inst[15:11]};dmem_r<=1'b0;
831.                //IF
832.                mux_pc_sel<=`MUX_PC_NPC;
833.                //ID
834.
835.                mux_ALUa_sel<=`MUX_ALUa_IMM0;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
835.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
836.                //EX
837.
838.                alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
838.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
839.                //WB
840.                mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
841.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b1;
842.            end
843.            default: begin    //do nothing
844.                //FLOW
845.
846.                raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
846.                //IF
847.                mux_pc_sel<=`MUX_PC_NPC;
848.                //ID
849.
850.                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
850.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
851.                //EX
852.
853.                alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
853.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
854.                //WB
855.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
856.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
857.            end
858.        endcase
859.    end
860.    6'b011100: begin
861.        case (inst[5:0])
862.            6'b000010: begin    //mul
863.                //FLOW
864.
865.                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
865.                //IF
866.                mux_pc_sel<=`MUX_PC_NPC;
867.                //ID
868.
869.                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
869.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
870.                //EX

```

```

871.    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b1;use_overflow<=1'b0;
872.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
873.        //WB
874.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_L0;
875.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
876.    end
877.    6'b100000: begin    //clz
878.        //FLOW
879.
880.        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
LATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
881.        //IF
882.        mux_pc_sel<=`MUX_PC_NPC;
883.        //ID
884.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
885.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
886.        //EX
887.        alu_sel<=`ALU_CLZ;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
888.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
889.        //WB
890.        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
891.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
892.    end
893.    default: begin    //do nothing
894.        //FLOW
895.
896.        raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
897.        //IF
898.        mux_pc_sel<=`MUX_PC_NPC;
899.        //ID
900.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
901.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
902.        //EX
903.        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
904.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
905.        //WB
906.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
907.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
908.    end
909.    endcase
910.    end
911.    default: begin    //do nothing
912.        //FLOW
913.
914.        raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
915.        //IF
916.        mux_pc_sel<=`MUX_PC_NPC;
917.        //ID
918.        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
919.        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
920.        //EX

```

```

918.    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
919.        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
920.        //WB
921.        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
922.        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
923.        end
924.    endcase
925. end
926. endmodule

```

5) cp0.v

```

1. `include "define.vh"
2.
3. module CP0(
4.     input clk,
5.     input rst,
6.     input mtc0, //cpu instruction mtc0,high-active
7.     input [31:0] pc,
8.
9.     input [4:0] waddr, //specifies CP0 reg to write
10.    input [31:0] wdata, //data from GP reg to place CP0 reg
11.    input exception, //instruction syscall,break,teq,high-active
12.    input eret, //instruction eret,high-active
13.    input [4:0] cause,
14.
15.    input [4:0] raddr, //specifies CP0 reg to read
16.    output [31:0] rdata, //data from CP0 reg for GP reg
17.    output [31:0] status, //mask,low-active
18.    output [31:0] exc_addr, //address for PC at the beginning of an exception
19.    output [31:0] intr_addr //exception check
20. );
21.
22. reg [31:0] cp0_reg[31:0]; //regfiles
23. wire [31:0] status_left,status_right;
24. assign status_left=cp0_reg[12]<<5;
25. assign status_right=cp0_reg[12]>>5;
26. assign status=cp0_reg[12];
27. assign exc_addr=cp0_reg[14];
28. assign rdata=cp0_reg[raddr];
29.
30. wire
    timer_int=exception&&((cause[3:0]==`EXC_SYSCALL&&status[1]&&status[0])||(cause[3:
0]==`EXC_BREAK&&status[2]&&status[0])||(cause[3:0]==`EXC_TEQ&&status[3]&&status[0
]));
31. assign intr_addr=timer_int?32'h00400004:pc;
32.
33. always@(posedge clk,posedge rst) begin
34.     if(rst) begin
35.         cp0_reg[0]<=32'b0;
36.         cp0_reg[1]<=32'b0;
37.         cp0_reg[2]<=32'b0;
38.         cp0_reg[3]<=32'b0;
39.         cp0_reg[4]<=32'b0;
40.         cp0_reg[5]<=32'b0;
41.         cp0_reg[6]<=32'b0;
42.         cp0_reg[7]<=32'b0;
43.         cp0_reg[8]<=32'b0;

```

```

44.         cp0_reg[9]<=32'b0;
45.         cp0_reg[10]<=32'b0;
46.         cp0_reg[11]<=32'b0;
47.         cp0_reg[12]<=32'h0000_000f;           //reg_status
48.         cp0_reg[13]<=32'h0;           //reg_cause
49.         cp0_reg[14]<=32'h0;           //reg_epc
50.         cp0_reg[15]<=32'b0;
51.         cp0_reg[16]<=32'b0;
52.         cp0_reg[17]<=32'b0;
53.         cp0_reg[18]<=32'b0;
54.         cp0_reg[19]<=32'b0;
55.         cp0_reg[20]<=32'b0;
56.         cp0_reg[21]<=32'b0;
57.         cp0_reg[22]<=32'b0;
58.         cp0_reg[23]<=32'b0;
59.         cp0_reg[24]<=32'b0;
60.         cp0_reg[25]<=32'b0;
61.         cp0_reg[26]<=32'b0;
62.         cp0_reg[27]<=32'b0;
63.         cp0_reg[28]<=32'b0;
64.         cp0_reg[29]<=32'b0;
65.         cp0_reg[30]<=32'b0;
66.         cp0_reg[31]<=32'b0;
67.     end
68.     else if(eret) begin
69.         cp0_reg[12]<=status_right;
70.     end
71.     else if(exception) begin
72.         case(cause[3:0])
73.             `EXC_SYSCALL: if(status[1]&status[0]) begin //syscall
74.                 cp0_reg[12]<=status_left;
75.                 cp0_reg[13][6:2]=cause;
76.                 cp0_reg[14]<=pc-32'h4;
77.             end
78.             `EXC_BREAK: if(status[2]&status[0]) begin //break
79.                 cp0_reg[12]<=status_left;
80.                 cp0_reg[13][6:2]=cause;
81.                 cp0_reg[14]<=pc-32'h4;
82.             end
83.             `EXC_TEQ: if(status[3]&status[0]) begin //teq
84.                 cp0_reg[12]<=status_left;
85.                 cp0_reg[13][6:2]=cause;
86.                 cp0_reg[14]<=pc-32'h4;
87.             end
88.             default: begin end
89.         endcase
90.     end
91.     else if(mtc0) begin
92.         cp0_reg[waddr]<=wdata;
93.     end
94.     else begin end
95. end
96. endmodule

```

6) cpu_regfile.v

```

1. module regfile(
2.     input clk, //posedge write-active
3.     input rst, //active-high asynchronous

```



```

4.     input we,    //high:write low:read
5.     input [31:0] raddr1,
6.     input [31:0] raddr2,
7.     output [31:0] rdata1,
8.     output [31:0] rdata2,
9.     input [31:0] waddr,
10.    input [31:0] wdata,
11.
12.    //for program out
13.    output [31:0] reg_16
14.    );
15.
16.    (* KEEP = "{TRUE|FALSE|SOFT}" *) reg [31:0] array_reg[31:0]; //regfiles
17.
18.    always @(posedge clk or posedge rst) begin
19.        if(rst) begin
20.            array_reg[0]<=32'b0;
21.            array_reg[1]<=32'b0;
22.            array_reg[2]<=32'b0;
23.            array_reg[3]<=32'b0;
24.            array_reg[4]<=32'b0;
25.            array_reg[5]<=32'b0;
26.            array_reg[6]<=32'b0;
27.            array_reg[7]<=32'b0;
28.            array_reg[8]<=32'b0;
29.            array_reg[9]<=32'b0;
30.            array_reg[10]<=32'b0;
31.            array_reg[11]<=32'b0;
32.            array_reg[12]<=32'b0;
33.            array_reg[13]<=32'b0;
34.            array_reg[14]<=32'b0;
35.            array_reg[15]<=32'b0;
36.            array_reg[16]<=32'b0;
37.            array_reg[17]<=32'b0;
38.            array_reg[18]<=32'b0;
39.            array_reg[19]<=32'b0;
40.            array_reg[20]<=32'b0;
41.            array_reg[21]<=32'b0;
42.            array_reg[22]<=32'b0;
43.            array_reg[23]<=32'b0;
44.            array_reg[24]<=32'b0;
45.            array_reg[25]<=32'b0;
46.            array_reg[26]<=32'b0;
47.            array_reg[27]<=32'b0;
48.            array_reg[28]<=32'b0;
49.            array_reg[29]<=32'b0;
50.            array_reg[30]<=32'b0;
51.            array_reg[31]<=32'b0;
52.        end
53.        else if (we&&waddr) begin    //cannot modify $0
54.            array_reg[waddr]<=wdata;
55.        end
56.    end
57.
58.    assign rdata1=array_reg[raddr1];
59.    assign rdata2=array_reg[raddr2];
60.
61.    //for program out

```

```

62.     assign reg_16=array_reg[16];
63. endmodule

```

7) cpu_top.v

```

1. `include "define.vh"
2.
3. module top_parts(
4.     input clk, //posedge write-active
5.     input reset, //active-high asynchronous
6.     output [31:0] top_pc,
7.     output [31:0] top_ir,
8.
9.     //for program out
10.    output [31:0] reg_16,
11.    input cpu_stall
12. );
13.
14.    wire [1:0] cond0,cond1,cond2,cond3,cond4;
15.    wire [6:0] flow_raddr1,flow_raddr2,flow_waddr1,flow_waddr2,flow_waddr3;
16.    wire [31:0] forward_ALUa,forward_ALUb,forward_Rt;
17.    wire forward_ALUa_w,forward_ALUb_w,forward_Rt_w;
18.
19.    wire [31:0] connect,npc_ext,regfile_Rs,cp0_EPC,cp0_intr_addr;
20.    wire [31:0] if_NPC,if_IR;
21.    wire [2:0] mux_pc_sel;
22.    assign top_ir=if_IR;
23.
24.    instruction_fetch if_inst(
25.        .clk(clk), //posedge write-active
26.        .reset(reset), //active-high asynchronous
27.        .connect(connect),
28.        .npc_ext(npc_ext),
29.        .regfile_Rs(regfile_Rs),
30.        .cp0_EPC(cp0_EPC),
31.        .cp0_intr_addr(cp0_intr_addr),
32.        .oNPC(if_NPC),
33.        .rPC(top_pc),
34.        .rIR(if_IR),
35.        .cond(cond0),
36.        .mux_pc_sel(mux_pc_sel)
37.    );
38.
39.    wire hi_w,lo_w,cp0_w,regfile_w;
40.    wire [4:0] regfile_Rdc;wire [31:0] regfile_Rd,Rd_out_for_L0;
41.    wire [31:0] id_ALUa,id_ALUb,id_Rt,id_IR,ext_out;
42.    assign npc_ext = ext_out+if_NPC;
43.
44.    instruction_decode id_inst(
45.        .clk(clk),
46.        .reset(reset),
47.        .if_IR(if_IR),
48.        .if_NPC(if_NPC),
49.        .regfile_Rdc(regfile_Rdc), //also cp0
50.        .regfile_Rd(regfile_Rd), //also cp0
51.        .Rd_out_for_L0(Rd_out_for_L0),
52.        .rALUa(id_ALUa),
53.        .rALUb(id_ALUb),
54.        .rRt(id_Rt),

```

```

55.         .rIR(id_IR),
56.         .cp0_EPC(cp0_EPC),
57.         .cp0_intr_addr(cp0_intr_addr),
58.         .ext_out(ext_out),
59.         .connect(connect),
60.         .regfile_Rs(regfile_Rs),
61.         .regfile_Rt(),
62.         .cond(cond1),
63.         .mux_pc_sel(mux_pc_sel),
64.         .hi_w(hi_w),
65.         .lo_w(lo_w),
66.         .regfile_w(regfile_w),
67.         .cp0_w(cp0_w),
68.         .flow_raddr1(flow_raddr1),
69.         .flow_raddr2(flow_raddr2),
70.         .forward_ALUa(forward_ALUa),
71.         .forward_ALUb(forward_ALUb),
72.         .forward_Rt(forward_Rt),
73.         .forward_ALUa_w(forward_ALUa_w),
74.         .forward_ALUb_w(forward_ALUb_w),
75.         .forward_Rt_w(forward_Rt_w),
76.
77.         //for program out
78.         .reg_16(reg_16)
79.     );
80.
81.     wire [31:0] ex_HI,ex_LO,ex_Z,ex_Rt,ex_IR;
82.     wire mult_div_stall,cal_finish,overflow_stall;
83.
84.     execute ex_inst(
85.         .clk(clk),
86.         .reset(reset),
87.         .alua(id_ALUa),
88.         .alub(id_ALUb),
89.         .id_Rt(id_Rt),
90.         .id_IR(id_IR),
91.         .rHI(ex_HI),
92.         .rLO(ex_LO),
93.         .rZ(ex_Z),
94.         .rRt(ex_Rt),
95.         .rIR(ex_IR),
96.         .cond(cond2),
97.         .mult_div_stall(mult_div_stall),    //cause controller to stall 32 periods
98.         .cal_finish(cal_finish),
99.         .overflow_stall(overflow_stall),
100.        .flow_waddr(flow_waddr1),
101.
102.        //for program in
103.        .cpu_stall(cpu_stall)
104.    );
105.
106.    wire [31:0] me_HI,me_LO,me_Z,me_MEM,me_IR;
107.    wire ex_from_mem,me_from_mem;
108.    wire ex_mul,me_mul;
109.
110.    memory_access me_inst(
111.        .clk(clk),
112.        .reset(reset),

```

```

113.         .ex_HI(ex_HI),
114.         .ex_LO(ex_LO),
115.         .ex_Z(ex_Z),
116.         .ex_Rt(ex_Rt),
117.         .ex_IR(ex_IR),
118.         .rHI(me_HI),
119.         .rLO(me_LO),
120.         .rZ(me_Z),
121.         .rMEM(me_MEM),
122.         .rIR(me_IR),
123.         .cond(cond3),
124.         .flow_waddr(flow_waddr2),
125.         .dmem_r(ex_from_mem),
126.         .use_mul(ex_mul)
127.     );
128.
129.     write_back wb_inst(
130.         .clk(clk),
131.         .reset(reset),
132.         .me_HI(me_HI),
133.         .me_LO(me_LO),
134.         .me_Z(me_Z),
135.         .me_MEM(me_MEM),
136.         .me_IR(me_IR),
137.         .mux_Rdc_out(regfile_Rdc),
138.         .mux_Rd_out(regfile_Rd),
139.         .Rd_out_for_LO(Rd_out_for_LO),
140.         .cond(cond4),
141.         .hi_w(hi_w),
142.         .lo_w(lo_w),
143.         .cp0_w(cp0_w),
144.         .regfile_w(regfile_w),
145.         .flow_waddr(flow_waddr3),
146.         .dmem_r(me_from_mem),
147.         .use_mul(me_mul)
148.     );
149.
150.     flow_control flow_control_inst(
151.         .clk(clk), .reset(reset),
152.         .raddr1(flow_raddr1), .raddr2(flow_raddr2), .waddr1(flow_waddr1), .waddr2(
flow_waddr2), .waddr3(flow_waddr3),
153.         .mult_div_stall(mult_div_stall), .mult_div_over(cal_finish), .overflow_st
all(overflow_stall),
154.         /*-----flow_control-----*/
155.         .cond0(cond0), .cond1(cond1), .cond2(cond2), .cond3(cond3), .cond4(cond4),
156.         /*-----flow_data-----*/
157.         .id_ALUa(forward_ALUa), .id_ALUb(forward_ALUb), .id_Rt(forward_Rt), .id_AL
Ua_w(forward_ALUa_w), .id_ALUb_w(forward_ALUb_w), .id_Rt_w(forward_Rt_w),
158.         .ex_HI(ex_HI), .ex_LO(ex_LO), .ex_Z(ex_Z), .ex_from_mem(ex_from_mem), .ex_m
ul(ex_mul),
159.         .me_HI(me_HI), .me_LO(me_LO), .me_Z(me_Z), .me_MEM(me_MEM), .me_from_mem(me
_from_mem), .me_mul(me_mul),
160.         //for program in
161.         .cpu_stall(cpu_stall)
162.     );
163. endmodule

```

8) DIV.v

```

1. module DIV(
2.     input [31:0] dividend,
3.     input [31:0] divisor,
4.     input start,
5.     input clock,
6.     input reset,    //active-high
7.     output [31:0] q,
8.     output [31:0] r,
9.     output reg busy,
10.    output reg finish,
11.
12.    //for program in
13.    input cpu_stall
14.    );
15.
16.    reg sign_dnd,sign_vsr;
17.    wire [31:0] udividend,udivisor;
18.    wire [31:0] uq,ur;
19.    assign udividend=dividend[31]?(~dividend+1'b1):dividend;
20.    assign udivisor=divisor[31]?(~divisor+1'b1):divisor;
21.    assign q=(sign_dnd^sign_vsr)?(~uq+1'b1):uq;
22.    assign r=sign_dnd?(~ur+1'b1):ur;
23.
24.    reg [5:0] cnt;
25.    reg [32:0] inner_sr;
26.    reg [31:0] rmdr;    //remainder
27.    reg [31:0] qtnt;    //quotient
28.    reg sign;
29.
30.    wire [32:0] inner_complement_sr;
31.    assign inner_complement_sr=~inner_sr+1'b1;
32.    wire [32:0] add;
33.    assign add={rmdr,qtnt[31]}+(sign?inner_complement_sr:inner_sr);
34.
35.    assign ur=rmdr;
36.    assign uq=qtnt;
37.
38.    always@(posedge(clock) or posedge(reset)) begin
39.        if(reset==1) begin
40.            cnt<=0;
41.            busy<=0;
42.            rmdr<=0;
43.            qtnt<=0;
44.            inner_sr<=0;
45.            sign<=0;
46.
47.            sign_dnd<=0;
48.            sign_vsr<=0;
49.            finish<=0;
50.        end
51.        else begin
52.            if(start) begin
53.                rmdr<=0;
54.                qtnt<=udividend;
55.                inner_sr<={1'b0,udivisor};
56.                busy<=1;
57.                cnt<=1;

```

```

58.             sign<=1;
59.
60.             sign_dnd<=dividend[31];
61.             sign_vsr<=divisor[31];
62.             finish<=0;
63.         end
64.         else if(busy) begin
65.             if (!cpu_stall) begin
66.                 case(cnt)
67.
68. 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
31: begin
68.                     {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]};
69.                     sign<=~add[32];
70.                     cnt<=cnt+1;
71.                     finish<=0;
72.                 end
73.                 32: begin
74.                     {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]}+(add[32]?{inner_sr[31:0],32'b0}:64'b
0);
75.                     sign<=~add[32];
76.                     cnt<=cnt+1;
77.                     busy<=0;
78.                     finish<=1;
79.                 end
80.                 default: begin end
81.             endcase
82.         end
83.     end
84.     else
85.         finish<=0;
86.     end
87. end
88. endmodule

```

9) DIVU.v

```

1. module DIVU(
2.     input [31:0] dividend,
3.     input [31:0] divisor,
4.     input start,
5.     input clock,
6.     input reset,    //active-high
7.     output [31:0] q,
8.     output [31:0] r,
9.     output reg busy,
10.    output reg finish,
11.
12.    //for program in
13.    input cpu_stall
14. );
15.
16. reg [5:0] cnt;
17. reg [32:0] inner_sr;
18. reg [31:0] rmdr;    //remainder
19. reg [31:0] qtnt;    //quotient
20. reg sign;
21.

```

```

22. wire [32:0] inner_complement_sr;
23. assign inner_complement_sr=~inner_sr+1'b1;
24. wire [32:0] add;
25. assign add={rmdr,qtnt[31]}+(sign?inner_complement_sr:inner_sr);
26.
27. assign r=rmdr;
28. assign q=qtnt;
29.
30. always@(posedge(clock) or posedge(reset)) begin
31.     if(reset==1) begin
32.         cnt<=0;
33.         busy<=0;
34.         rmdr<=0;
35.         qtnt<=0;
36.         inner_sr<=0;
37.         sign<=0;
38.         finish<=0;
39.     end
40.     else begin
41.         if(start) begin
42.             rmdr<=0;
43.             qtnt<=dividend;
44.             inner_sr<={1'b0,divisor};
45.             busy<=1;
46.             cnt<=1;
47.             sign<=1;
48.             finish<=0;
49.         end
50.         else if(busy) begin
51.             if (!cpu_stall) begin
52.                 case(cnt)
53.                     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
54.                     31: begin
55.                         {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]};
56.                         sign<=~add[32];
57.                         cnt<=cnt+1;
58.                         finish<=0;
59.                     end
60.                     32: begin
61.                         {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]}+(add[32]?{inner_sr[31:0],32'b0}:64'b
62.                         0);
63.                         sign<=~add[32];
64.                         cnt<=cnt+1;
65.                         busy<=0;
66.                         finish<=1;
67.                     end
68.                     default: begin end
69.                 endcase
70.             end
71.             else
72.                 finish<=0;
73.         end
74.     end
endmodule

```

10) extend.v

```

1. `include "define.vh"
2.
3. module ext(
4.     input [2:0] ext_switch,
5.
6.     input [4:0] iData_len5,
7.     input [7:0] iData_len8,
8.     input [15:0] iData_len16,
9.     input [31:0] iData_len32,
10.    output reg [31:0] oData
11.);
12.
13. always @(*) begin
14.    case (ext_switch)
15.        `EXT5_Z: oData<={27'b0,iData_len5};
16.        `EXT16_SL2_S:
17.            oData=iData_len16[15]?{14'h3fff,iData_len16,2'b0}:{14'b0,iData_len16,2'b0};
18.        `EXT16_Z: oData<={16'b0,iData_len16};
19.        `EXT16_S: oData<={{16{iData_len16[15]}},iData_len16};
20.        `EXT8_Z: oData<={24'b0,iData_len8};
21.        `EXT8_S: oData<={{24{iData_len8[7]}},iData_len8};
22.        `EXT32_NON: oData<=iData_len32;
23.        default: oData<=32'b0;
24.    endcase
25. end
26. endmodule

```

11) flow_control.v

```

1. `include "define.vh"
2.
3. module flow_control(
4.     input clk,input reset,
5.     input [6:0] raddr1,input [6:0] raddr2,input [6:0] waddr1,input [6:0] waddr2,input
6.     [6:0] waddr3,
7.     input mult_div_stall,input mult_div_over,input overflow_stall,
8.
9.     /*-----flow_control-----*/
10.    output [1:0] cond0,output [1:0] cond1,output [1:0] cond2,output [1:0]
11.    cond3,output [1:0] cond4,
12.
13.    /*-----flow_data-----*/
14.    output reg [31:0] id_ALUa,output reg [31:0] id_ALUb,output reg [31:0] id_Rt,output
15.    id_ALUa_w,output id_ALUb_w,output id_Rt_w,
16.    input [31:0] ex_HI,input [31:0] ex_LO,input [31:0] ex_Z,input ex_from_mem,input
17.    ex_mul,
18.    input [31:0] me_HI,input [31:0] me_LO,input [31:0] me_Z,input [31:0] me_MEM,input
19.    me_from_mem,input me_mul,
20.
21.    //for program in
22.    input cpu_stall
23.);
24.
25. reg [1:0] cond[0:4];
26. assign cond0=cond[0];
27. assign cond1=cond[1];
28. assign cond2=cond[2];
29. assign cond3=cond[3];

```



```

25.     assign cond4=cond[4];
26.
27.     reg [1:0] state,nextstate;reg [4:0] cnt;
28.     wire
        violation1=(waddr1!=`VIOLATION_NON)&&(raddr1==waddr1 || raddr2==waddr1 || (waddr1==`V
        IOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_
        HI || raddr2==`VIOLATION_LO)));
29.     wire
        violation2=(waddr2!=`VIOLATION_NON)&&(raddr1==waddr2 || raddr2==waddr2 || (waddr2==`V
        IOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_
        HI || raddr2==`VIOLATION_LO)));
30.     wire
        violation3=(waddr3!=`VIOLATION_NON)&&(raddr1==waddr3 || raddr2==waddr3 || (waddr3==`V
        IOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO || raddr2==`VIOLATION_
        HI || raddr2==`VIOLATION_LO)));
31.     wire violation=violation1|(violation2&ex_from_mem);
32.
33.     always @(posedge clk or posedge reset) begin
34.         if(reset)
35.             state<=`FLOW_NORMAL;
36.         else
37.             state<=nextstate;
38.     end
39.
40.     assign
        id_ALUa_w=(!violation)&&(raddr1!=`VIOLATION_NON)&&(raddr1==waddr2 || raddr1==waddr3
        || (waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO)) || (wadr
        dr3==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIOLATION_LO)));
41.     assign
        id_ALUb_w=(!violation)&&(raddr2!=`VIOLATION_NON)&&(raddr2==waddr2 || raddr2==waddr3
        );
42.     assign id_Rt_w=id_ALUb_w;
43.     always @(*) begin
44.         if(!violation) begin
45.             if(id_ALUa_w) begin //use alua
46.
47.                 if(raddr1==waddr2 || (waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI || raddr1==`VIO
                    LATION_LO))) begin //use ex
48.                     case (waddr2)
49.                         `VIOLATION_HI: id_ALUa<=ex_HI;
50.                         `VIOLATION_LO: id_ALUa<=ex_LO;
51.                         `VIOLATION_HILO: begin
52.                             if (raddr1==`VIOLATION_HI)
53.                                 id_ALUa<=ex_HI;
54.                             else
55.                                 id_ALUa<=ex_LO;
56.                         end
57.                         default: begin
58.                             case (waddr2[6:5])
59.                                 `VIOLATION_REGFILE_HEAD: id_ALUa<=ex_Z;
60.                                 `VIOLATION_CP0REG_HEAD: id_ALUa<=ex_Z;
61.                                 default: begin end
62.                             endcase
63.                         end
64.                     endcase
65.                 else begin //use me
66.                     if (me_from_mem) begin

```

```

67.         id_ALUa<=me_MEM;
68.     end
69.     case (waddr2)
70.         `VIOLATION_HI: id_ALUa<=me_HI;
71.         `VIOLATION_LO: id_ALUa<=me_LO;
72.         `VIOLATION_HILO: begin
73.             if (raddr1==`VIOLATION_HI)
74.                 id_ALUa<=me_HI;
75.             else
76.                 id_ALUa<=me_LO;
77.             end
78.         default: begin
79.             case (waddr2[6:5])
80.                 `VIOLATION_REGFILE_HEAD: id_ALUa<=me_Z;
81.                 `VIOLATION_CP0REG_HEAD: id_ALUa<=me_Z;
82.                 default: begin end
83.             endcase
84.         end
85.     endcase
86. end
87. end
88.
89. if (id_ALUb_w) begin //use alub
90.     if (raddr2==waddr2) //use ex
91.         id_ALUb<=ex_mul?ex_LO:ex_Z;
92.     else begin //use me
93.         if (me_from_mem)
94.             id_ALUb<=me_MEM;
95.         else
96.             id_ALUb<=me_mul?me_LO:me_Z;
97.         end
98.     end
99.
100.    if (id_Rt_w) begin //use Rt
101.        if (raddr2==waddr2) //use ex
102.            id_Rt<=ex_mul?ex_LO:ex_Z;
103.        else begin //use me
104.            if (me_from_mem)
105.                id_Rt<=me_MEM;
106.            else
107.                id_Rt<=me_mul?me_LO:me_Z;
108.            end
109.        end
110.    end
111. end
112.
113. always @(*) begin
114.     if(cpu_stall)begin
115.         cond[0]<=`PARTS_COND_STALL; //IF
116.         cond[1]<=`PARTS_COND_STALL; //ID
117.         cond[2]<=`PARTS_COND_STALL; //EX
118.         cond[3]<=`PARTS_COND_STALL; //ME
119.         cond[4]<=`PARTS_COND_STALL; //WB
120.         nextstate<=state;
121.     end else begin
122.         case (state)
123.             `FLOW_NORMAL: begin
124.                 if (overflow_stall) begin

```

```

125.         if (violation) begin
126.             cond[0]<=`PARTS_COND_FLOW; //IF
127.             cond[1]<=`PARTS_COND_ZERO; //ID
128.             cond[2]<=`PARTS_COND_ZERO; //EX
129.             cond[3]<=`PARTS_COND_FLOW; //ME
130.             cond[4]<=`PARTS_COND_FLOW; //WB
131.             nextstate<=`FLOW_NORMAL;
132.         end else begin
133.             cond[0]<=`PARTS_COND_FLOW; //IF
134.             cond[1]<=`PARTS_COND_FLOW; //ID
135.             cond[2]<=`PARTS_COND_ZERO; //EX
136.             cond[3]<=`PARTS_COND_FLOW; //ME
137.             cond[4]<=`PARTS_COND_FLOW; //WB
138.             nextstate<=`FLOW_NORMAL;
139.         end
140.     end
141. else if (!mult_div_over&mult_div_stall) begin
142.     cond[0]<=`PARTS_COND_STALL; //IF
143.     cond[1]<=`PARTS_COND_STALL; //ID
144.     cond[2]<=`PARTS_COND_STALL; //EX
145.     cond[3]<=`PARTS_COND_STALL; //ME
146.     cond[4]<=`PARTS_COND_STALL; //WB
147.     nextstate<=`FLOW_MULDIV;
148. end
149. else if (violation) begin
150.     cond[0]<=`PARTS_COND_STALL; //IF
151.     cond[1]<=`PARTS_COND_ZERO; //ID
152.     cond[2]<=`PARTS_COND_FLOW; //EX
153.     cond[3]<=`PARTS_COND_FLOW; //ME
154.     cond[4]<=`PARTS_COND_FLOW; //WB
155.     nextstate<=`FLOW_NORMAL;
156. end else begin
157.     cond[0]<=`PARTS_COND_FLOW; //IF
158.     cond[1]<=`PARTS_COND_FLOW; //ID
159.     cond[2]<=`PARTS_COND_FLOW; //EX
160.     cond[3]<=`PARTS_COND_FLOW; //ME
161.     cond[4]<=`PARTS_COND_FLOW; //WB
162.     nextstate<=`FLOW_NORMAL;
163. end
164. end
165. `FLOW_MULDIV: begin
166.     if (mult_div_over) begin
167.         if (violation) begin
168.             cond[0]<=`PARTS_COND_STALL; //IF
169.             cond[1]<=`PARTS_COND_ZERO; //ID
170.             cond[2]<=`PARTS_COND_FLOW; //EX
171.             cond[3]<=`PARTS_COND_FLOW; //ME
172.             cond[4]<=`PARTS_COND_FLOW; //WB
173.             nextstate<=`FLOW_NORMAL;
174.         end else begin
175.             cond[0]<=`PARTS_COND_FLOW; //IF
176.             cond[1]<=`PARTS_COND_FLOW; //ID
177.             cond[2]<=`PARTS_COND_FLOW; //EX
178.             cond[3]<=`PARTS_COND_FLOW; //ME
179.             cond[4]<=`PARTS_COND_FLOW; //WB
180.             nextstate<=`FLOW_NORMAL;
181.         end
182.     end else begin

```

```

183.             cond[0]<=`PARTS_COND_STALL; //IF
184.             cond[1]<=`PARTS_COND_STALL; //ID
185.             cond[2]<=`PARTS_COND_STALL; //EX
186.             cond[3]<=`PARTS_COND_STALL; //ME
187.             cond[4]<=`PARTS_COND_STALL; //WB
188.             nextstate<=`FLOW_MULDIV;
189.         end
190.     end
191.     default: begin
192.         cond[0]<=`PARTS_COND_FLOW; //IF
193.         cond[1]<=`PARTS_COND_FLOW; //ID
194.         cond[2]<=`PARTS_COND_FLOW; //EX
195.         cond[3]<=`PARTS_COND_FLOW; //ME
196.         cond[4]<=`PARTS_COND_FLOW; //WB
197.         nextstate<=`FLOW_NORMAL;
198.     end
199. endcase
200. end
201. end
202. endmodule

```

12) MULT.v

```

1. module MULT(
2.     input clk,
3.     input reset,    //active high
4.     input start,
5.     input [31:0] a, //multiplicand
6.     input [31:0] b, //multiplier
7.     output [63:0] z,
8.     output reg busy,
9.     output reg finish,
10.
11.     //for program in
12.     input cpu_stall
13. );
14.
15.     reg [5:0] cnt;
16.     reg [32:0] multa,multb;
17.     reg [32:0] multpart;
18.     reg shiftr;
19.     wire [32:0] add;
20.
21.     wire [32:0] complementa;
22.     assign complementa=~multa+1'b1;
23.
24.     assign z={multpart,multb[32:2]};
25.     assign
26.         add=multpart+(multb[1]==1?(multb[0]==1?33'b0:complementa):(multb[0]==1?multa:33'b
27.         0));
28.     always@(posedge clk or posedge reset)
29.     begin
30.         if(reset) begin
31.             cnt<=0;
32.             multa<=0;
33.             multb<=0;
34.             multpart<=0;
35.             busy<=0;
36.             finish<=0;

```

```

35.     end
36.     else begin
37.         if(start) begin
38.             cnt<=1;
39.             multa<={a[31],a};
40.             multb<={b,1'b0};
41.             multpart<=0;
42.             busy<=1;
43.             finish<=0;
44.         end else if(busy) begin
45.             if (!cpu_stall) begin
46.                 case(cnt)
47.
48. 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
31: begin
49.                     {multpart,multb,shiftr}<={add[32],add,multb};
50.                     cnt<=cnt+1;
51.                     finish<=0;
52.                 end
53. 32: begin
54.                     {multpart,multb}<={add,multb};
55.                     cnt<=cnt+1;
56.                     busy<=0;
57.                     finish<=1;
58.                 end
59.                 default: begin end
60.             endcase
61.         end
62.     else
63.         finish<=0;
64.     end
65. end
66. endmodule

```

13) MULTU.v

```

1. module MULTU(
2.     input clk,
3.     input reset,    //active high
4.     input start,
5.     input [31:0] a, //multiplicand
6.     input [31:0] b, //multiplier
7.     output [63:0] z,
8.     output reg busy,
9.     output reg finish,
10.
11.     //for program in
12.     input cpu_stall
13. );
14.
15. reg [5:0] cnt;
16. reg [31:0] multa,multb;
17. reg [31:0] multpart;
18. reg shiftr;
19. wire cf;
20. wire [31:0] add;
21.
22. assign z={multpart,multb};

```

```

23.    assign {cf,add}={1'b0,multipart}+{1'b0,(multb[0]?multa:32'b0)};
24.    always@(posedge clk or posedge reset)
25.    begin
26.        if(reset) begin
27.            cnt<=0;
28.            multa<=0;
29.            multb<=0;
30.            multipart<=0;
31.            busy<=0;
32.            finish<=0;
33.        end
34.        else begin
35.            if(start) begin
36.                cnt<=1;
37.                multa<=a;
38.                multb<=b;
39.                multipart<=0;
40.                busy<=1;
41.                finish<=0;
42.            end else if(busy) begin
43.                if (!cpu_stall) begin
44.                    case(cnt)
45.
46.                        1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
47.                        31: begin
48.
49.                            {multipart,multb,shiftr}<={cf,add,multb};
50.                            cnt<=cnt+1;
51.                            finish<=0;
52.                        end
53.                        32: begin
54.                            {multipart,multb,shiftr}<={cf,add,multb};
55.                            cnt<=cnt+1;
56.                            busy<=0;
57.                            finish<=1;
58.                        end
59.                        default: begin end
60.                    endcase
61.                end
62.            end
63.        end
64.    endmodule

```

14) mux.v

```

1. module mux_len32_sel8(
2.     input [2:0] sel,
3.     input [31:0] iData_1,
4.     input [31:0] iData_2,
5.     input [31:0] iData_3,
6.     input [31:0] iData_4,
7.     input [31:0] iData_5,
8.     input [31:0] iData_6,
9.     input [31:0] iData_7,
10.    input [31:0] iData_8,
11.    output reg [31:0] oData
12. );

```

```

13.
14.     //reg oData; //wire in implementation
15.
16.     always @(*) begin
17.         case(sel)
18.             3'b000: oData<=iData_1;
19.             3'b001: oData<=iData_2;
20.             3'b010: oData<=iData_3;
21.             3'b011: oData<=iData_4;
22.             3'b100: oData<=iData_5;
23.             3'b101: oData<=iData_6;
24.             3'b110: oData<=iData_7;
25.             3'b111: oData<=iData_8;
26.             default: begin end
27.         endcase
28.     end
29. endmodule
30.
31.
32. module mux_len32_sel4(
33.     input [1:0] sel,
34.     input [31:0] iData_1,
35.     input [31:0] iData_2,
36.     input [31:0] iData_3,
37.     input [31:0] iData_4,
38.     output reg [31:0] oData
39. );
40.
41.     //reg oData; //wire in implementation
42.
43.     always @(*) begin
44.         case(sel)
45.             2'b00: oData<=iData_1;
46.             2'b01: oData<=iData_2;
47.             2'b10: oData<=iData_3;
48.             2'b11: oData<=iData_4;
49.             default: begin end
50.         endcase
51.     end
52. endmodule
53.
54. // module mux_len32_sel2(
55. //     input sel,
56. //     input [31:0] iData_1,
57. //     input [31:0] iData_2,
58. //     output reg [31:0] oData
59. // );
60.
61. //     //reg oData; //wire in implementation
62.
63. //     always @(*) begin
64. //         case(sel)
65. //             1'b0: oData<=iData_1;
66. //             1'b1: oData<=iData_2;
67. //             default: begin end
68. //         endcase
69. //     end
70. // endmodule

```

```

71.
72.
73. module mux_len5_sel4(
74.     input [1:0] sel,
75.     input [4:0] iData_1,
76.     input [4:0] iData_2,
77.     input [4:0] iData_3,
78.     input [4:0] iData_4,
79.     output reg [4:0] oData
80. );
81.
82.     //reg oData; //wire in implementation
83.
84.     always @(*) begin
85.         case(sel)
86.             2'b00: oData<=iData_1;
87.             2'b01: oData<=iData_2;
88.             2'b10: oData<=iData_3;
89.             2'b11: oData<=iData_4;
90.             default: begin end
91.         endcase
92.     end
93. endmodule

```

15) parts.v

```

1. `include "define.vh"
2.
3. module instruction_fetch(
4.     input clk, //posedge write-active
5.     input reset, //active-high asynchronous
6.
7.     input [31:0] connect,
8.     input [31:0] npc_ext,
9.     input [31:0] regfile_Rs,
10.    input [31:0] cp0_EPC,
11.    input [31:0] cp0_intr_addr,
12.
13.    output [31:0] oNPC,
14.    output reg [31:0] rPC,
15.    output reg [31:0] rIR,
16.
17.    //control signal
18.    input [1:0] cond,
19.    input [2:0] mux_pc_sel
20. );
21.
22.    wire [31:0] imem_out,mux_pc_out;
23.    assign oNPC=rPC+32'h4;
24.
25.    mux_len32_sel8 mux_Pc(
26.        .sel(mux_pc_sel),
27.        .iData_1(npc_ext),
28.        .iData_2(regfile_Rs),
29.        .iData_3(cp0_intr_addr),
30.        .iData_4(cp0_EPC),
31.        .iData_5(connect),
32.        .iData_6(oNPC),
33.        .oData(mux_pc_out)

```



```

34. );
35. // ram imem(
36. //   .clk(clk), //posedge read/write-active
37. //   .wena(1'b0), //high:write low:read
38. //   .width(`RAM_WIDTH_32), //0:word,1:hword,2:byte
39. //   .addr(rPC),
40. //   .data_in(32'b0), //use zero side
41. //   .data_out(imem_out) //use zero side
42. // );
43. wire [31:0] imem_addr=(rPC-32'h00400000)>>2;
44. imem imem_inst(
45.   .a(imem_addr[10:0]),
46.   .spo(imem_out)
47. );
48.
49. always @(posedge clk or posedge reset) begin //execute
50.   if (reset) begin
51.     rPC<=`PC_ADDR_INIT;
52.   end else begin
53.     case (cond)
54.       `PARTS_COND_FLOW: rPC<=mux_pc_out;
55.       `PARTS_COND_STALL: rPC<=rPC;
56.       `PARTS_COND_ZERO: rPC<=`PC_ADDR_INIT;
57.       default: begin end
58.     endcase
59.   end
60. end
61.
62. always @(negedge clk or posedge reset) begin //flow
63.   if (reset) begin
64.     rIR<=`IR_NON;
65.   end else begin
66.     case (cond)
67.       `PARTS_COND_FLOW: rIR<=imem_out;
68.       `PARTS_COND_STALL: rIR<=rIR;
69.       `PARTS_COND_ZERO: rIR<=`IR_NON;
70.       default: begin end
71.     endcase
72.   end
73. end
74.
75. endmodule
76.
77. module instruction_decode(
78.   input clk,
79.   input reset,
80.
81.   input [31:0] if_IR, //to read
82.   input [31:0] if_NPC,
83.   input [4:0] regfile_Rdc, //also cp0
84.   input [31:0] regfile_Rd, //also cp0,hi,lo
85.   input [31:0] Rd_out_for_LO, //add for lo *only when hi also write
86.
87.   output reg [31:0] rALUa,
88.   output reg [31:0] rALUb,
89.   output reg [31:0] rRt,
90.   output reg [31:0] rIR,
91.   output [31:0] cp0_EPC,

```

```

92.     output [31:0] cp0_intr_addr,
93.     output [31:0] ext_out,
94.     output [31:0] connect,
95.     output [31:0] regfile_Rs,
96.     output [31:0] regfile_Rt,
97.
98.     //control signal
99.     input [1:0] cond,
100.     output [2:0] mux_pc_sel,
101.     input hi_w,
102.     input lo_w,
103.     input regfile_w,
104.     input cp0_w,
105.     output [6:0] flow_raddr1,
106.     output [6:0] flow_raddr2,
107.
108.     //forward
109.     input [31:0] forward_ALUa,
110.     input [31:0] forward_ALUb,
111.     input [31:0] forward_Rt,
112.     input forward_ALUa_w,
113.     input forward_ALUb_w,
114.     input forward_Rt_w,
115.
116.     //for program out
117.     output [31:0] reg_16
118. );
119.
120.     assign connect={if_NPC[31:28],if_IR[25:0],2'b0};
121.
122.     reg [31:0] rHI,rLO;
123.
124.     wire mux_lo_sel,mux_hi_sel;
125.     wire [31:0] cal_lo_out,cal_hi_out,mux_lo_out,mux_hi_out;
126.     wire [31:0] alua,alub;
127.     wire [31:0] cp0_out;wire [4:0] cp0_cause;
128.
129.     wire [2:0] mux_ALUa_sel;wire [1:0] mux_ALUb_sel;wire [2:0] ext_sel;
130.     wire cp0_eret,cp0_exception;
131.
132.     wire judge_beq,judge_bgez;
133.     assign
        judge_beq=((forward_ALUa_w?forward_ALUa:regfile_Rs)==(forward_ALUb_w?forward_ALUb
        :regfile_Rt));
134.     assign
        judge_bgez=(forward_ALUa_w?(forward_ALUa[31]==1'b0):(regfile_Rs[31]==1'b0));
135.
136.     controller controller_id(
137.         .inst(if_IR),
138.         .judge_beq(judge_beq), //judge BEQ BNE condition to jump
139.         .judge_bgez(judge_bgez), //judge BGEZ condition to jump
140.
141.         /*-----flow_control-----*/
142.         .raddr1(flow_raddr1),.raddr2(flow_raddr2),
143.         /*-----control-----*/
144.         //ID
145.         .mux_pc_sel(mux_pc_sel),

```

```

146.         .mux_ALUa_sel(mux_ALUa_sel), .mux_ALUb_sel(mux_ALUb_sel), .ext_sel(ext_se
1) ),
147.         .cp0_exception(cp0_exception), .cp0_eret(cp0_eret), .cp0_cause(cp0_cause)
148.     );
149.
150.     mux_len32_sel8 mux_ALUa(
151.         .sel(mux_ALUa_sel),
152.         .iData_1(rHI),
153.         .iData_2(rLO),
154.         .iData_3(if_NPC),
155.         .iData_4(regfile_Rs),
156.         .iData_5(ext_out),
157.         .iData_6(cp0_out),
158.         .iData_7(32'b0),
159.         .oData(alua)
160.     );
161.     mux_len32_sel4 mux_ALUb(
162.         .sel(mux_ALUb_sel),
163.         .iData_1(32'd0),
164.         .iData_2(regfile_Rt),
165.         .iData_3(ext_out),
166.         .oData(alub)
167.     );
168.     ext ext_id(
169.         .ext_switch(ext_sel),
170.         .iData_len5(if_IR[10:6]),
171.         .iData_len8(),
172.         .iData_len16(if_IR[15:0]),
173.         .iData_len32(),
174.         .oData(ext_out)
175.     );
176.
177.
178.     regfile cpu_ref(
179.         .clk(clk), //posedge write-active
180.         .rst(reset), //active-high asynchronous
181.         .we(regfile_w), //high:write low:read
182.         .raddr1({27'b0,if_IR[25:21]}),
183.         .raddr2({27'b0,if_IR[20:16]}),
184.         .rdata1(regfile_Rs),
185.         .rdata2(regfile_Rt),
186.         .waddr({27'b0,regfile_Rdc}),
187.         .wdata(regfile_Rd),
188.
189.         //for program out
190.         .reg_16(reg_16)
191.     );
192.     CP0 cp0_inst(
193.         .clk(clk),
194.         .rst(reset),
195.         .mtc0(cp0_w), //cpu instruction mtc0,high-active
196.         .pc(if_NPC),
197.         .waddr(regfile_Rdc), //specifies CP0 reg to write
198.         .wdata(regfile_Rd), //data from GP reg to place CP0 reg
199.         .exception(cp0_exception&&(cond==`PARTS_COND_FLOW)), //instruction
syscall,break,teq,high-active
200.         .eret(cp0_eret&&(cond==`PARTS_COND_FLOW)), //instruction
eret,high-active

```

```

201.         .cause(cp0_cause),
202.         .raddr(if_IR[15:11]), //specifies CP0 reg to read
203.         .rdata(cp0_out),      //data from CP0 reg for GP reg
204.         .exc_addr(cp0_EPC),    //address for PC at the beginning of an exception
205.         .intr_addr(cp0_intr_addr)
206.     );
207.
208.     always @(posedge clk or posedge reset) begin //execute
209.         if (reset) begin
210.             rALUa<=32'b0;
211.             rALUb<=32'b0;
212.             rRt<=32'b0;
213.             rHI<=32'b0;
214.             rLO<=32'b0;
215.         end else begin
216.             case (cond)
217.                 `PARTS_COND_FLOW: begin
218.                     rALUa<=forward_ALUa_w?forward_ALUa:alua;
219.                     rALUb<=forward_ALUb_w?forward_ALUb:alub;
220.                     rRt<=forward_Rt_w?forward_Rt:regfile_Rt;
221.                 end
222.                 `PARTS_COND_STALL: begin
223.                     rALUa<=rALUa;
224.                     rALUb<=rALUb;
225.                     rRt<=rRt;
226.                 end
227.                 `PARTS_COND_ZERO: begin
228.                     rALUa<=32'b0;
229.                     rALUb<=32'b0;
230.                     rRt<=32'b0;
231.                 end
232.                 default: begin end
233.             endcase
234.             rHI<=hi_w?regfile_Rd:rHI;
235.             rLO<=lo_w?(hi_w?Rd_out_for_LO:regfile_Rd):rLO;
236.         end
237.     end
238.
239.     always @(negedge clk or posedge reset) begin //flow
240.         if (reset) begin
241.             rIR<=`IR_NON;
242.         end else begin
243.             case (cond)
244.                 `PARTS_COND_FLOW: rIR<=if_IR;
245.                 `PARTS_COND_STALL: rIR<=rIR;
246.                 `PARTS_COND_ZERO: rIR<=`IR_NON;
247.                 default: begin end
248.             endcase
249.         end
250.     end
251. endmodule
252.
253. module execute(
254.     input clk,
255.     input reset,
256.
257.     input [31:0] alua,
258.     input [31:0] alub,

```

```

259.     input [31:0] id_Rt,
260.     input [31:0] id_IR,
261.
262.     output reg [31:0] rHI,
263.     output reg [31:0] rLO,
264.     output reg [31:0] rZ,
265.     output reg [31:0] rRt,
266.     output reg [31:0] rIR,
267.
268.     //control signal
269.     input [1:0] cond,
270.     output mult_div_stall,    //cause controller to stall 32 periods
271.     output cal_finish,
272.     output overflow_stall,    //next IR_NON
273.     output [6:0] flow_waddr,
274.
275.     //for program in
276.     input cpu_stall
277. );
278.
279.     wire [31:0] cal_lo,cal_hi;
280.     wire [1:0] cal_sel;
281.     (* KEEP = "{TRUE|FALSE|SOFT}" *) wire cal_ena;
282.     wire [31:0] alu_z;
283.     wire [3:0] alu_sel;
284.
285.     wire use_overflow;
286.     (* KEEP = "{TRUE|FALSE|SOFT}" *) assign mult_div_stall=cal_ena;
287.     assign overflow_stall=use_overflow&alu_overflow;
288.
289.     controller controller_ex(
290.         .inst(id_IR),
291.         .judge_beq(1'b0), //judge BEQ BNE condition to jump
292.         .judge_bgez(1'b0), //judge BGEZ condition to jump
293.         /*-----flow_control-----*/
294.         .waddr(flow_waddr),
295.         /*-----control-----*/
296.         //EX
297.         .alu_sel(alu_sel),.cal_sel(cal_sel),.cal_ena(cal_ena)/*also for
stall*/,.use_overflow(use_overflow)
298.     );
299.
300.     calculator calculator_inst(
301.         .clk(clk), //negedge calculate
302.         .a(alua), //multiplicand/dividend
303.         .b(alub), //multiplier/divisor
304.         .calc(cal_sel),
305.         .reset(reset),    //high-active,at the beginning of test
306.         .ena(cal_ena),    //high-active
307.         .oLO(cal_lo),
308.         .oHI(cal_hi),
309.         .sum_finish(cal_finish),
310.
311.         //for program in
312.         .cpu_stall(cpu_stall)
313.     );
314.
315.     alu alu_inst(

```

```

316.         .a(alua),
317.         .b(alub),
318.         .aluc(alu_sel),
319.         .r(alu_z),
320.         .overflow(alu_overflow)
321.     );
322.
323.     always @(posedge clk or posedge reset) begin //execute
324.         if (reset) begin
325.             rZ<=32'b0;
326.             rRt<=32'b0;
327.         end else begin
328.             case (cond)
329.                 `PARTS_COND_FLOW: begin
330.                     rZ<=alu_z;
331.                     rRt<=id_Rt;
332.                 end
333.                 `PARTS_COND_STALL: begin
334.                     rZ<=rZ;
335.                     rRt<=rRt;
336.                 end
337.                 `PARTS_COND_ZERO: begin
338.                     rZ<=32'b0;
339.                     rRt<=32'b0;
340.                 end
341.                 default: begin end
342.             endcase
343.         end
344.     end
345.
346.     always @(negedge clk or posedge reset) begin //flow
347.         if (reset) begin
348.             rHI<=32'b0;
349.             rLO<=32'b0;
350.             rIR<=`IR_NON;
351.         end else begin
352.             case (cond)
353.                 `PARTS_COND_FLOW: begin
354.                     rHI<=cal_hi;
355.                     rLO<=cal_lo;
356.                     rIR<=id_IR;
357.                 end
358.                 `PARTS_COND_STALL: begin
359.                     rHI<=rHI;
360.                     rLO<=rLO;
361.                     rIR<=rIR;
362.                 end
363.                 `PARTS_COND_ZERO: begin
364.                     rHI<=32'b0;
365.                     rLO<=32'b0;
366.                     rIR<=`IR_NON;
367.                 end
368.                 default: begin end
369.             endcase
370.         end
371.     end
372. endmodule
373.

```

```

374. module memory_access(
375.     input clk,
376.     input reset,
377.
378.     input [31:0] ex_HI,
379.     input [31:0] ex_LO,
380.     input [31:0] ex_Z,
381.     input [31:0] ex_Rt,
382.     input [31:0] ex_IR,
383.
384.     output reg [31:0] rHI,
385.     output reg [31:0] rLO,
386.     output reg [31:0] rZ,
387.     output reg [31:0] rMEM,
388.     output reg [31:0] rIR,
389.
390.     //control signal
391.     input [1:0] cond,
392.     output [6:0] flow_waddr,
393.     output dmem_r,
394.     output use_mul
395. );
396.
397.     assign use_mul=(ex_IR[31:26]==6'b011100&&ex_IR[5:0]==6'b000010);
398.     wire dmem_w;wire [1:0] dmem_width;wire [31:0] dmem_out,ext_out;wire [2:0]
mem_sel;
399.
400.     controller controller_me(
401.         .inst(ex_IR),
402.         .judge_beq(1'b0), //judge BEQ BNE condition to jump
403.         .judge_bgez(1'b0), //judge BGEZ condition to jump
404.         /*-----flow_control-----*/
405.         .waddr(flow_waddr),.dmem_r(dmem_r),
406.         /*-----control-----*/
407.         //ME
408.         .dmem_w(dmem_w),.dmem_width(dmem_width),.mem_sel(mem_sel)
409.     );
410.
411.     ram dmem_inst(
412.         .clk(clk), //posedge read/write-active
413.         .wena(dmem_w), //high:write low:read
414.         .width(dmem_width), //0:word,1:hword,2:byte
415.         .addr(ex_Z-32'h10010000),
416.         .data_in(ex_Rt), //use zero side
417.         .data_out(dmem_out) //use zero side
418.     );
419.
420.     ext ext_me(
421.         .ext_switch(mem_sel),
422.         .iData_len5(),
423.         .iData_len8(dmem_out[7:0]),
424.         .iData_len16(dmem_out[15:0]),
425.         .iData_len32(dmem_out),
426.         .oData(ext_out)
427.     );
428.
429.     always @(posedge clk or posedge reset) begin //execute
430.         if (reset) begin

```

```

431.         rHI<=32'b0;
432.         rLO<=32'b0;
433.         rZ<=32'b0;
434.         rMEM<=32'b0;
435.     end else begin
436.         case (cond)
437.             `PARTS_COND_FLOW: begin
438.                 rHI<=ex_HI;
439.                 rLO<=ex_LO;
440.                 rZ<=ex_Z;
441.                 rMEM<=ext_out;
442.             end
443.             `PARTS_COND_STALL: begin
444.                 rHI<=rHI;
445.                 rLO<=rLO;
446.                 rZ<=rZ;
447.                 rMEM<=rMEM;
448.             end
449.             `PARTS_COND_ZERO: begin
450.                 rHI<=32'b0;
451.                 rLO<=32'b0;
452.                 rZ<=32'b0;
453.                 rMEM<=32'b0;
454.             end
455.             default: begin end
456.         endcase
457.     end
458. end
459.
460. always @(negedge clk or posedge reset) begin    //flow
461.     if (reset) begin
462.         rIR<=`IR_NON;
463.     end else begin
464.         case (cond)
465.             `PARTS_COND_FLOW: rIR<=ex_IR;
466.             `PARTS_COND_STALL: rIR<=rIR;
467.             `PARTS_COND_ZERO: rIR<=`IR_NON;
468.             default: begin end
469.         endcase
470.     end
471. end
472. endmodule
473.
474. module write_back(
475.     input clk,
476.     input reset,
477.
478.     input [31:0] me_HI,
479.     input [31:0] me_LO,
480.     input [31:0] me_Z,
481.     input [31:0] me_MEM,
482.     input [31:0] me_IR,
483.
484.     output [4:0] mux_Rdc_out,
485.     output [31:0] mux_Rd_out,
486.     output [31:0] Rd_out_for_LO,
487.
488.     //control signal

```



```

489.     input [1:0] cond,
490.     output hi_w,
491.     output lo_w,
492.     output cp0_w,
493.     output regfile_w,
494.     output [6:0] flow_waddr,
495.     output dmem_r,
496.     output use_mul
497. );
498.
499.     assign use_mul=(me_IR[31:26]==6'b011100&&me_IR[5:0]==6'b000010);
500.     assign Rd_out_for_LO=me_LO;
501.
502.     wire [1:0] mux_Rdc_sel,mux_Rd_sel;
503.     wire hi_w_in,lo_w_in,cp0_w_in,regfile_w_in;
504.     wire write_ena=(cond==`PARTS_COND_FLOW);
505.     assign hi_w=write_ena&hi_w_in;
506.     assign lo_w=write_ena&lo_w_in;
507.     assign cp0_w=write_ena&cp0_w_in;
508.     assign regfile_w=write_ena&file_w_in;
509.
510.     controller controller_wb(
511.         .inst(me_IR),
512.         .judge_beq(1'b0), //judge BEQ BNE condition to jump
513.         .judge_bgez(1'b0), //judge BGEZ condition to jump
514.
515.         /*-----flow_control-----*/
516.         .waddr(flow_waddr),.dmem_r(dmem_r),
517.         /*-----control-----*/
518.         //WB
519.         .mux_Rdc_sel(mux_Rdc_sel),.mux_Rd_sel(mux_Rd_sel),
520.         .hi_w(hi_w_in),.lo_w(lo_w_in),.regfile_w(regfile_w_in),.cp0_w(cp0_w_in)
521.     );
522.
523.     mux_len5_sel4 mux_Rdc(
524.         .sel(mux_Rdc_sel),
525.         .iData_1(me_IR[15:11]),
526.         .iData_2(me_IR[20:16]),
527.         .iData_3(5'd31),
528.         .oData(mux_Rdc_out)
529.     );
530.     mux_len32_sel4 mux_Rd(
531.         .sel(mux_Rd_sel),
532.         .iData_1(me_Z),
533.         .iData_2(me_MEM),
534.         .iData_3(me_HI),
535.         .iData_4(me_LO),
536.         .oData(mux_Rd_out)
537.     );
538. endmodule

```

16) ram.v

```

1.  `include "define.vh"
2.
3.  module ram(
4.      input clk, //posedge read/write-active
5.      input wena, //high:write low:read
6.

```

```

7.     input [1:0] width, //0:word,1:hword,2:byte
8.     input [31:0] addr,
9.     input [31:0] data_in, //use zero side
10.    output reg [31:0] data_out //use zero side
11.    );
12.
13.    reg [31:0] memory [0:4095]; //16KB
14.    wire [31:0] ram_addr;
15.    assign ram_addr=addr>>2;
16.
17.    always @(posedge clk) begin
18.        if(wena) begin
19.            case (width)
20.                `RAM_WIDTH_32: memory[ram_addr]<=data_in; //addr[1:0]=00
21.                `RAM_WIDTH_16: begin //addr[0]=0
22.                    if(addr[1]==1'b0)
23.                        memory[ram_addr][15:0]<=data_in[15:0];
24.                    else
25.                        memory[ram_addr][31:16]<=data_in[15:0];
26.                end
27.                `RAM_WIDTH_8: begin
28.                    case (addr[1:0])
29.                        2'b00: memory[ram_addr][7:0]<=data_in[7:0];
30.                        2'b01: memory[ram_addr][15:8]<=data_in[7:0];
31.                        2'b10: memory[ram_addr][23:16]<=data_in[7:0];
32.                        2'b11: memory[ram_addr][31:24]<=data_in[7:0];
33.                        default: begin end
34.                    endcase
35.                end
36.                default: begin end
37.            endcase
38.        end
39.    end
40.
41.    always @(*) begin
42.        case (width)
43.            `RAM_WIDTH_32: data_out[31:0]<=memory[ram_addr]; //addr[1:0]=00
44.            `RAM_WIDTH_16: begin //addr[0]=0
45.                if(addr[1]==1'b0)
46.                    data_out<={16'b0,memory[ram_addr][15:0]};
47.                else
48.                    data_out<={16'b0,memory[ram_addr][31:16]};
49.            end
50.            `RAM_WIDTH_8: begin
51.                case (addr[1:0])
52.                    2'b00: data_out<={24'b0,memory[ram_addr][7:0]};
53.                    2'b01: data_out<={24'b0,memory[ram_addr][15:8]};
54.                    2'b10: data_out<={24'b0,memory[ram_addr][23:16]};
55.                    2'b11: data_out<={24'b0,memory[ram_addr][31:24]};
56.                    default: begin end
57.                endcase
58.            end
59.            default: data_out<=32'b0;
60.        endcase
61.    end
62.
63. endmodule

```

17) seg7x16.v

```

1. module seg7x16(
2.     input clk,
3.     input reset,
4.     input cs,
5.     input [31:0] i_data,
6.     output [7:0] o_seg,
7.     output [7:0] o_sel
8. );
9.
10.    reg [12:0] cnt;
11.    always @ (posedge clk, posedge reset)
12.        if (reset)
13.            cnt <= 0;
14.        else
15.            cnt <= cnt + 1'b1;
16.
17.    wire seg7_clk = cnt[12];
18.
19.    reg [2:0] seg7_addr;
20.
21.    always @ (posedge seg7_clk, posedge reset)
22.        if(reset)
23.            seg7_addr <= 0;
24.        else
25.            seg7_addr <= seg7_addr + 1'b1;
26.
27.    reg [7:0] o_sel_r;
28.
29.    always @ (*)
30.        case(seg7_addr)
31.            7 : o_sel_r = 8'b01111111;
32.            6 : o_sel_r = 8'b10111111;
33.            5 : o_sel_r = 8'b11011111;
34.            4 : o_sel_r = 8'b11101111;
35.            3 : o_sel_r = 8'b11110111;
36.            2 : o_sel_r = 8'b11111011;
37.            1 : o_sel_r = 8'b11111101;
38.            0 : o_sel_r = 8'b11111110;
39.        endcase
40.
41.    reg [31:0] i_data_store;
42.    always @ (posedge clk, posedge reset)
43.        if(reset)
44.            i_data_store <= 0;
45.        else if(cs)
46.            i_data_store <= i_data;
47.
48.    reg [7:0] seg_data_r;
49.    always @ (*)
50.        case(seg7_addr)
51.            0 : seg_data_r = i_data_store[3:0];
52.            1 : seg_data_r = i_data_store[7:4];
53.            2 : seg_data_r = i_data_store[11:8];
54.            3 : seg_data_r = i_data_store[15:12];
55.            4 : seg_data_r = i_data_store[19:16];
56.            5 : seg_data_r = i_data_store[23:20];

```

```

57.         6 : seg_data_r = i_data_store[27:24];
58.         7 : seg_data_r = i_data_store[31:28];
59.     endcase
60.
61.     reg [7:0] o_seg_r;
62.     always @ (posedge clk, posedge reset)
63.     if(reset)
64.         o_seg_r <= 8'hff;
65.     else
66.         case(seg_data_r)
67.             4'h0 : o_seg_r <= 8'hC0;
68.             4'h1 : o_seg_r <= 8'hF9;
69.             4'h2 : o_seg_r <= 8'hA4;
70.             4'h3 : o_seg_r <= 8'hB0;
71.             4'h4 : o_seg_r <= 8'h99;
72.             4'h5 : o_seg_r <= 8'h92;
73.             4'h6 : o_seg_r <= 8'h82;
74.             4'h7 : o_seg_r <= 8'hF8;
75.             4'h8 : o_seg_r <= 8'h80;
76.             4'h9 : o_seg_r <= 8'h90;
77.             4'hA : o_seg_r <= 8'h88;
78.             4'hB : o_seg_r <= 8'h83;
79.             4'hC : o_seg_r <= 8'hC6;
80.             4'hD : o_seg_r <= 8'hA1;
81.             4'hE : o_seg_r <= 8'h86;
82.             4'hF : o_seg_r <= 8'h8E;
83.         endcase
84.
85.     assign o_sel = o_sel_r;
86.     assign o_seg = o_seg_r;
87.
88. endmodule

```

18) seg7_top.v

```

1. `define CLK_PERIOD 20
2.
3. module seg7_top(
4.     input clk_in,
5.     input reset,
6.     output [7:0] o_seg,
7.     output [7:0] o_sel,
8.     input cpu_stall,
9.     input switch_pc_d
10.
11.     // output [31:0] test_pc,
12.     // output [31:0] test_ir
13. );
14.
15. (* KEEP = "{TRUE|FALSE|SOFT}" *) wire [31:0] test_pc,test_ir;
16.
17. wire clk_cpu, clk_seg7,clk_100mhz;
18. assign clk_100mhz=clk_in;
19.
20. reg [`CLK_PERIOD-1:0] counter;
21. wire [31:0] seg7_idata;
22. assign clk_cpu=counter[`CLK_PERIOD-1];
23. assign clk_seg7=counter[0];
24.

```

```

25.     always @(posedge clk_100mhz or posedge reset) begin
26.         if(reset) begin
27.             counter<=0;
28.         end
29.         else begin
30.             counter<=counter+1'b1;
31.         end
32.     end
33.
34.     wire [31:0] reg_16;
35.     assign seg7_idata=switch_pc_d?test_pc:reg_16;
36.
37.     seg7x16 seg7x16_inst(
38.         .clk(clk_seg7),
39.         .reset(reset),
40.         .cs(1'b1),
41.         .i_data(seg7_idata),
42.         .o_seg(o_seg),
43.         .o_sel(o_sel)
44.     );
45.
46.     top_parts cpu_inst(
47.         .clk(clk_cpu), //posedge write-active
48.         .reset(reset), //active-high asynchronous
49.         .top_pc(test_pc),
50.         .top_ir(test_ir),
51.
52.         //for program out
53.         .reg_16(reg_16),
54.         .cpu_stall(cpu_stall)
55.     );
56. endmodule

```

3、 动态流水线的仿真程序

1) cpu_simulation_tb.v

```

1. module cpu_tb();
2.     reg clk,reset;
3.     wire [31:0] inst,pc;
4.     reg [31:0] pc_history[0:4],inst_history[0:4];
5.     wire [31:0] reg_16;
6.     reg cpu_stall;
7.
8.     top_parts uut(
9.         clk, //posedge write-active
10.        reset, //active-high asynchronous
11.        pc,
12.        inst,
13.
14.        reg_16,
15.        cpu_stall
16.    );
17.
18.
19.    integer file_output;
20.    integer counter=0;
21.
22.    initial begin

```

```

23.         file_output=fopen("output.txt");
24.         //$readmemh("C:/DigitalLogic/DLProject/project_51/test.hex",cpu_tb.uut.mem.memory
);
25.         pc_history[0]=32'h0000_0000;
26.         clk=0;
27.         reset=1;
28.         cpu_stall=0;
29.         #275;
30.         reset=0;
31.     end
32.
33.     always begin
34.         #4;
35.         clk=~clk;
36.         #1;
37.         if(clk==1'b0&&reset==0) begin
38.             if(counter==5000/*|| (inst_history[3]==32'h00000000&&pc_history[3]!=32'h00400000)
*/) begin
39.                 $fclose(file_output);
40.                 $finish;
41.             end
42.             else if(pc_history[0]!=pc) begin
43.                 pc_history[4]=pc_history[3];
44.                 pc_history[3]=pc_history[2];
45.                 pc_history[2]=pc_history[1];
46.                 pc_history[1]=pc_history[0];
47.                 pc_history[0]=pc;
48.
49.                 inst_history[4]=inst_history[3];
50.                 inst_history[3]=inst_history[2];
51.                 inst_history[2]=inst_history[1];
52.                 inst_history[1]=inst_history[0];
53.                 inst_history[0]=inst;
54.
55.                 if (pc_history[4]!=32'h0000_0000) begin
56.                     counter=counter+1;
57.                     $fdisplay(file_output,"pc: %h",pc_history[4]);
58.                     $fdisplay(file_output,"instr: %h",inst_history[4]);
59.
60.                     $fdisplay(file_output,"regfile0: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[0]);
61.                     $fdisplay(file_output,"regfile1: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[1]);
62.                     $fdisplay(file_output,"regfile2: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[2]);
63.                     $fdisplay(file_output,"regfile3: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[3]);
64.                     $fdisplay(file_output,"regfile4: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[4]);
65.                     $fdisplay(file_output,"regfile5: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[5]);
66.                     $fdisplay(file_output,"regfile6: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[6]);
67.                     $fdisplay(file_output,"regfile7: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[7]);
68.                     $fdisplay(file_output,"regfile8: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[8]);

```

```

68.    $fdisplay(file_output,"regfile9: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[9]);
69.    $fdisplay(file_output,"regfile10: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[10]);
70.    $fdisplay(file_output,"regfile11: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[11]);
71.    $fdisplay(file_output,"regfile12: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[12]);
72.    $fdisplay(file_output,"regfile13: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[13]);
73.    $fdisplay(file_output,"regfile14: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[14]);
74.    $fdisplay(file_output,"regfile15: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[15]);
75.    $fdisplay(file_output,"regfile16: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[16]);
76.    $fdisplay(file_output,"regfile17: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[17]);
77.    $fdisplay(file_output,"regfile18: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[18]);
78.    $fdisplay(file_output,"regfile19: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[19]);
79.    $fdisplay(file_output,"regfile20: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[20]);
80.    $fdisplay(file_output,"regfile21: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[21]);
81.    $fdisplay(file_output,"regfile22: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[22]);
82.    $fdisplay(file_output,"regfile23: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[23]);
83.    $fdisplay(file_output,"regfile24: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[24]);
84.    $fdisplay(file_output,"regfile25: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[25]);
85.    $fdisplay(file_output,"regfile26: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[26]);
86.    $fdisplay(file_output,"regfile27: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[27]);
87.    $fdisplay(file_output,"regfile28: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[28]);
88.    $fdisplay(file_output,"regfile29: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[29]);
89.    $fdisplay(file_output,"regfile30: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[30]);
90.    $fdisplay(file_output,"regfile31: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[31]);
91.        end
92.    end
93. end
94. end
95. endmodule

```

2) cpu_synthesis_implementation_tb.v

```

1. module cpu_tb();
2.     reg clk,reset;
3.     wire [7:0] o_seg,o_sel;
4.     reg cpu_stall,switch_pc_d;
5.     seg7_top top_inst(

```

```
6.     clk,  
7.     reset,  
8.     o_seg,  
9.     o_sel,  
10.    cpu_stall,  
11.    switch_pc_d  
12. );  
13.  
14. initial begin  
15.     clk=0;  
16.     reset=1;  
17.     cpu_stall=0;  
18.     switch_pc_d=0;  
19.     #1000;  
20.     reset=0;  
21. end  
22.  
23. always begin  
24.     #5;  
25.     clk=~clk;  
26. end  
27. endmodule
```