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，CP0 模块 cp0_inst。

ID 段执行译码操作：主要包括从寄存器、CP0 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。

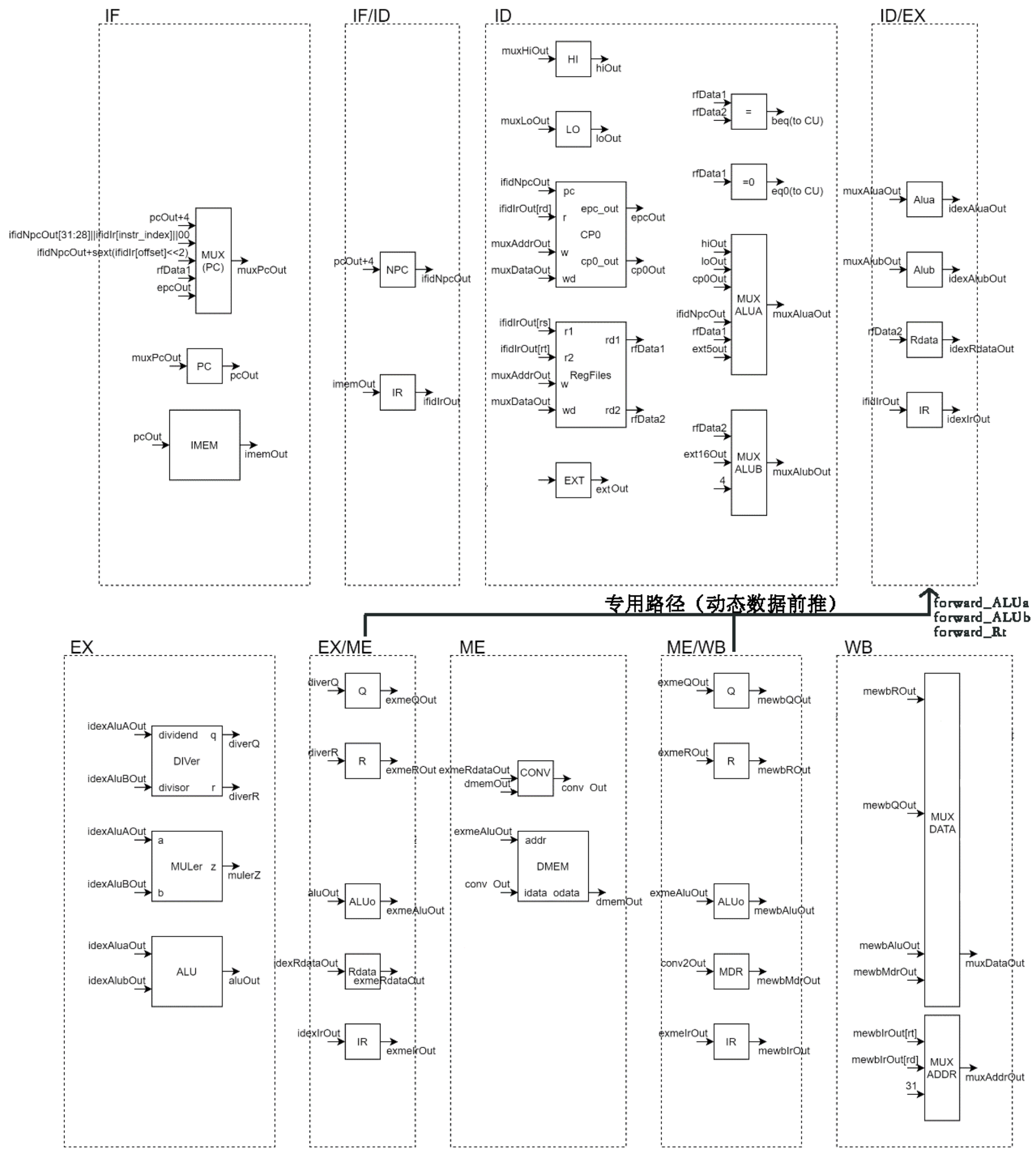
ME 段执行存储交互操作： 上升沿使用 Rt 寄存器写到内存指定单元，或内存读出数据写道 MEM 流水寄存器，HI、LO、Z 寄存器内容写到下一段流水寄存器。组合逻辑生成数据有符号、无符号扩展结果。

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

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

```
cond[3]<=`PARTS_COND_FLOW;  //ME
cond[4]<=`PARTS_COND_FLOW;  //WB
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

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

- 发生溢出时，排空一个流水段

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

- 发生读写冲突时，ID 前流水段先暂停：

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

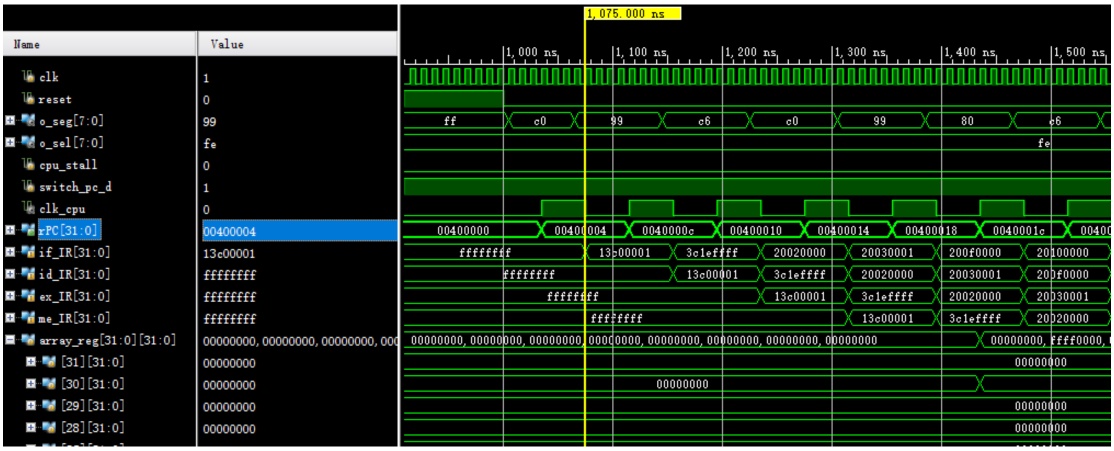
1、 测试样例说明

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）。

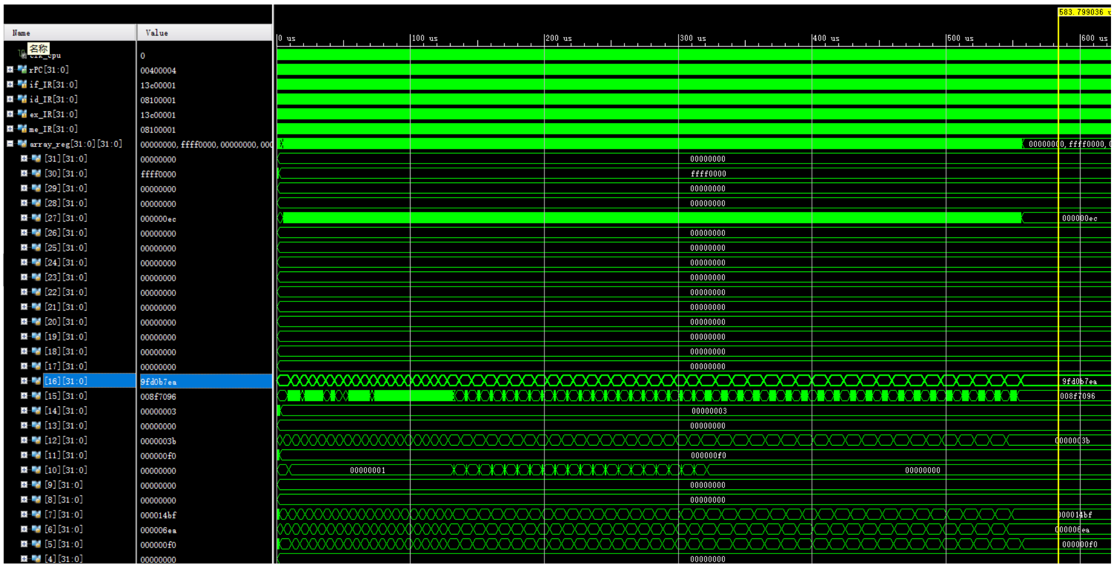
前仿真过程采用波形图输出和寄存器输出到文件的方式查看结果。后仿真无法访问寄存器，因此只采用波形图形式。

2、 动态流水线的前仿真过程

1) 仿真结果

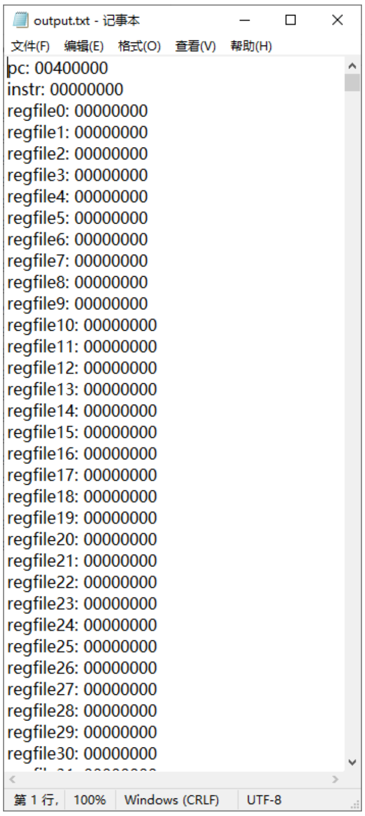


通过 if_IR、id_IR、ex_IR、me_IR 可观察各流水段流动和停止情况。

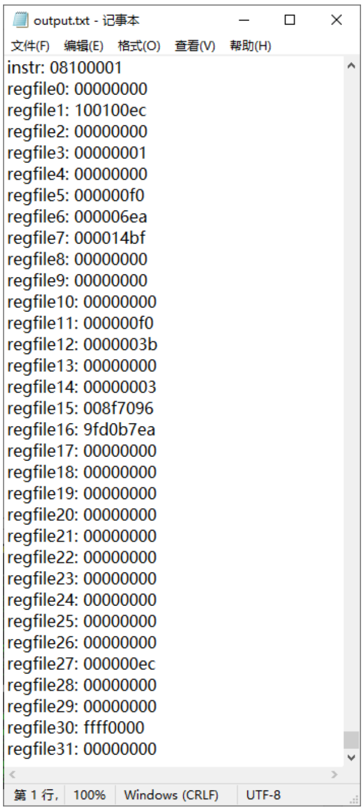


最终结果 array_reg[16]= 9FD0B7EA。结果正确。

2) 寄存器结果输出

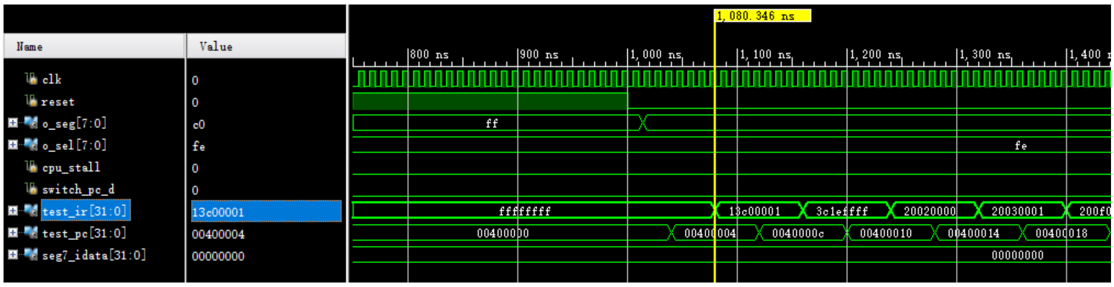


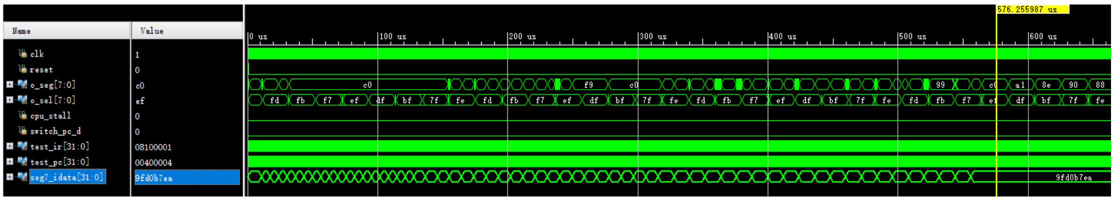
```
pc: 00400000
instr: 00000000
regfile0: 00000000
regfile1: 00000000
regfile2: 00000000
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: 00000000
regfile11: 00000000
regfile12: 00000000
regfile13: 00000000
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000000
regfile21: 00000000
regfile22: 00000000
regfile23: 00000000
regfile24: 00000000
regfile25: 00000000
regfile26: 00000000
regfile27: 00000000
regfile28: 00000000
regfile29: 00000000
regfile30: 00000000
```



```
instr: 08100001
regfile0: 00000000
regfile1: 100100ec
regfile2: 00000000
regfile3: 00000001
regfile4: 00000000
regfile5: 000000f0
regfile6: 000006ea
regfile7: 000014bf
regfile8: 00000000
regfile9: 00000000
regfile10: 00000000
regfile11: 000000f0
regfile12: 0000003b
regfile13: 00000000
regfile14: 00000003
regfile15: 008f7096
regfile16: 9fd0b7ea
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000000
regfile21: 00000000
regfile22: 00000000
regfile23: 00000000
regfile24: 00000000
regfile25: 00000000
regfile26: 00000000
regfile27: 000000ec
regfile28: 00000000
regfile29: 00000000
regfile30: ffff0000
regfile31: 00000000
```

3、 动态流水线的后仿真过程（分频 CLK_PERIOD=3）





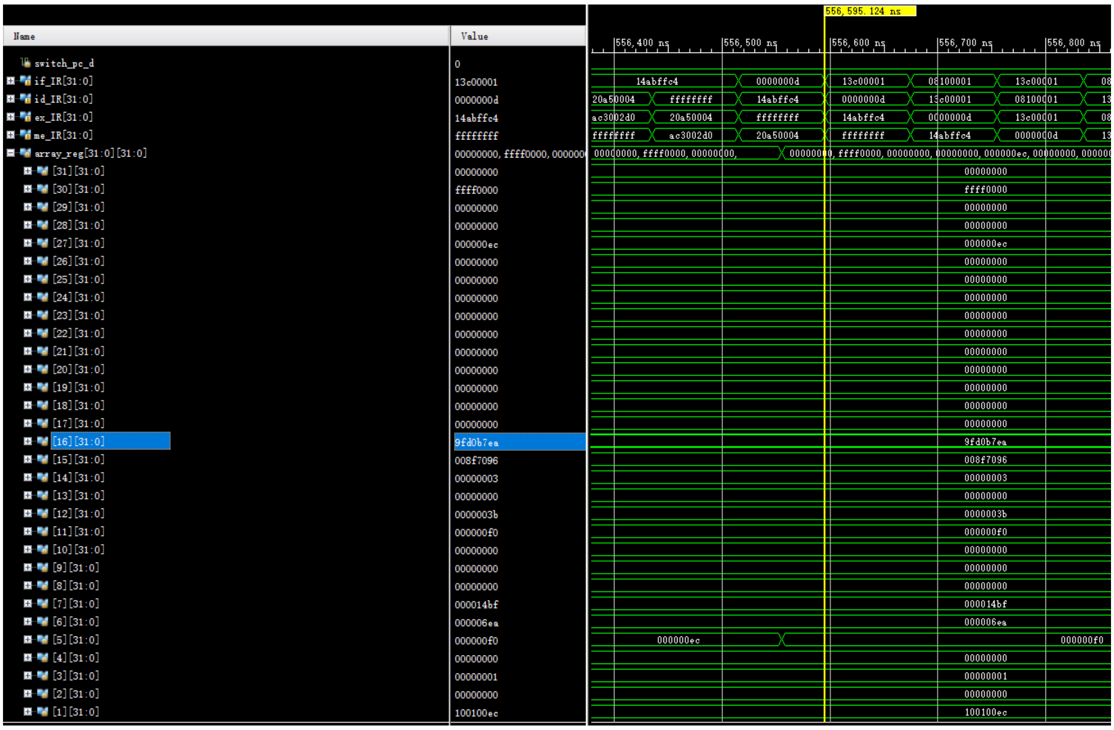
最终结果 9FD0B7EA。结果正确。

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

循环中的取值。中间结果寄存器#1，临时存储所有计算的中间结果。循环寄存器，#5 是循环次数寄存，#10 是跳转条件寄存，#27 是循环内存位移寄存。（另：#30 表示中断判断标志）



Name	Value
switch_pc_d	0
if_IR[31:0]	13c00001
id_IR[31:0]	0000000d
ex_IR[31:0]	14abffc4
me_IR[31:0]	ffffffff
array_reg[31:0][31:0]	00000000, ffff0000, 000000
[31][31:0]	00000000
[30][31:0]	ffff0000
[29][31:0]	00000000
[28][31:0]	00000000
[27][31:0]	000000ec
[26][31:0]	00000000
[25][31:0]	00000000
[24][31:0]	00000000
[23][31:0]	00000000
[22][31:0]	00000000
[21][31:0]	00000000
[20][31:0]	00000000
[19][31:0]	00000000
[18][31:0]	00000000
[17][31:0]	00000000
[16][31:0]	9fd0b7ea
[15][31:0]	008f7096
[14][31:0]	00000003
[13][31:0]	00000000
[12][31:0]	0000003b
[11][31:0]	000000f0
[10][31:0]	00000000
[9][31:0]	00000000
[8][31:0]	00000000
[7][31:0]	000014bf
[6][31:0]	000006ea
[5][31:0]	000000f0
[4][31:0]	00000000
[3][31:0]	00000001
[2][31:0]	00000000
[1][31:0]	100100ec

最终结果#16=9FD0B7EA。结果正确。

六、实验验算数学模型及算法程序

1) 算法模型

如图，利用循环完成以下所有值的计算，并存入内存中， #16 寄存器显示 D[i]的数据变化过程，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。

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;

$$c[i]=\begin{cases} a[i], & 0\le i\le 19 \\ a[i]+b[i], & 20\le i\le 39 \\ a[i]*b[i], & 40\le i\le 59 \end{cases}$$

$$d[i]=\begin{cases} b[i], & 0\le i\le 19 \\ a[i]*c[i], & 20\le i\le 39 \\ c[i]*b[i], & 40\le i\le 59 \end{cases}$$

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。

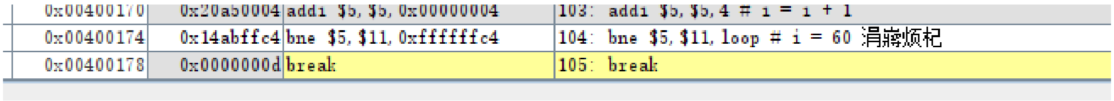
```
.data
A:.space 240
B:.space 240
C:.space 240
D:.space 240
.text
sll $0,$0,0
exc:          #exception
beq $30,$0,main
j exc

main:
lui $30,0xffff    #enable exception
addi $2,$0,0    #a[i]
addi $3,$0,1    #b[i]
addi $15,$0,0    #c[i]
addi $16,$0,0   #d[i]
addi $5,$0,4    #counter
addi $6,$0,0    #a[i-1]
addi $7,$0,1    #b[i-1]
addi $10,$0,0   #flag for i<20 || i<40
addi $11,$0,240 #sum counts
addi $14,$0,3
#addi $30,$0,0

# 把 0 1 0 0 ($2,...,$16) 分别存入 A B C D
lui $27,0x0000
addu $27,$27,$0
sw $2,A($27)
lui $27,0x0000
```

```
addu $27,$27,$0
sw $3,B($27)
lui $27,0x0000
addu $27,$27,$0
sw $2,C($27)
lui $27,0x0000
addu $27,$27,$0
sw $3,D($27)

# 循环
loop:
## $5(4) 除以 4 (=i) 存入 $12
srl $12,$5,2
# $6 加 i. 自此，$6 就是 a[i] 而不是 a[i-1] 了
add $6,$6,$12
# 把 a[i] 的内容存入 A[i] 中
lui $27,0x0000
addu $27,$27,$5
sw $6,A($27)
# $14 (3) 乘以 $5/4 (i) ( = 3i )
mul $15,$14,$12
# 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]
add $7,$7,$15
lui $27,0x0000
addu $27,$27,$5
sw $7,B($27)
# $5 是否小于 80 (i 是否小于 20)? 记入 $10
slti $10,$5,80
# 若不是，跳转
bne $10,1,c1

# (0<=i<=19)
# 把 $6 的内容存入 C[i] 中 (c[i] = a[i])
lui $27,0x0000
addu $27,$27,$5
sw $6,C($27)
# 把 $7 的内容存入 D[i] 中 (d[i] = b[i])
lui $27,0x0000
addu $27,$27,$5
sw $7,D($27)
addi $15,$6,0 # $15 $16 分别赋值为 c[i] d[i]
addi $16,$7,0
j endc
c1: # (20<=i<=39)
# i 是否小于 40 ？ 若不是，跳转到 c2
slti $10,$5,160
addi $27,$0,1
bne $10,$27,c2
# C[i] = a[i] + b[i]
add $15,$6,$7
lui $27,0x0000
addu $27,$27,$5
sw $15,C($27)
# D[i] = a[i] * b[i]
mul $16, $15,$6
lui $27,0x0000
addu $27,$27,$5
sw $16,D($27)
```

```
j endc
c2: # (i>=40)
# C[i] = a[i] * b[i]
mul $15,$6,$7
lui $27,0x0000
addu $27,$27,$5
sw $15,C($27)
# D[i] = c[i] * b[i]
mul $16,$15,$7
lui $27,0x0000
addu $27,$27,$5
sw $16,D($27)

endc:
addi $5,$5,4 # i = i + 1
bne $5,$11,loop # i = 60 不跳转
break
# 最后可通过验证 $16 的正确性来验证正确性
```

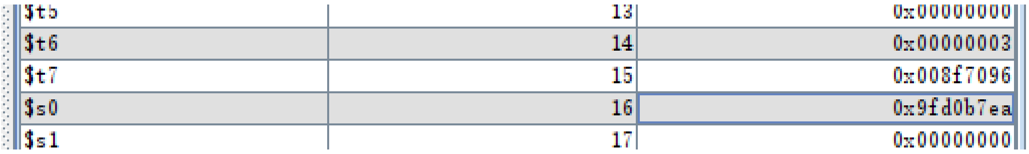
3) 验证程序结果

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  0x20a50004  addi $5,$5,0x00000004   addi $5,$5,4 # i = i + 1
0x00400174  0x14abffc4  bne $5,$11,0xffffffc4   bne $5,$11,loop # i = 60 涓嶈烦杞
0x00400178  0x0000000d  break                   break
```

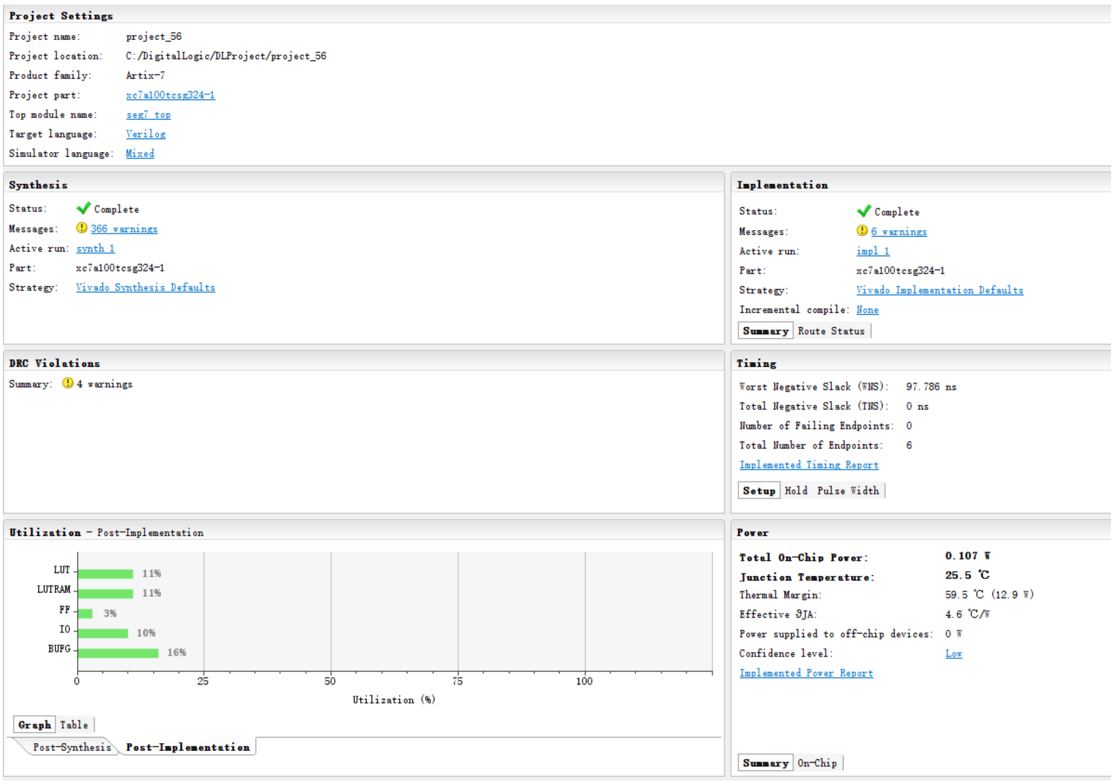
MIPS 程序输出：



```
$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



Project Settings

Project name:	project_56
Project location:	C:/DigitalLogic/DLProject/project_56
Product family:	Artix-7
Project part:	xc7a100tcsg324-1
Top module name:	seg7_top
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	✓ Complete
Messages:	! 366 warnings
Active run:	synth_1
Part:	xc7a100tcsg324-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	✓ Complete
Messages:	! 6 warnings
Active run:	impl_1
Part:	xc7a100tcsg324-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

DRC Violations

Summary: ! 4 warnings

Timing

Worst Negative Slack (WNS):	97.786 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	6

Implemented Timing Report

Utilization - Post-Implementation

	Utilization (%)
LUT	11%
LUTRAM	11%
FF	3%
IO	10%
BUFG	16%

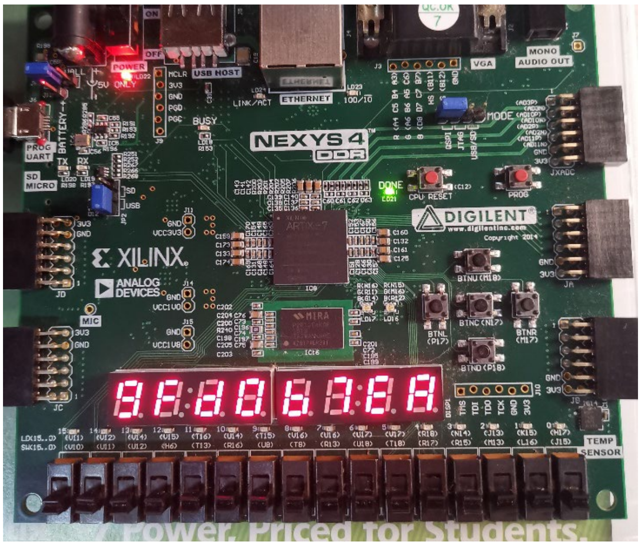
Power

Total On-Chip Power:	0.107 W
Junction Temperature:	25.5 ℃
Thermal Margin:	59.5 ℃ (12.9 W)
Effective θJA:	4.6 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

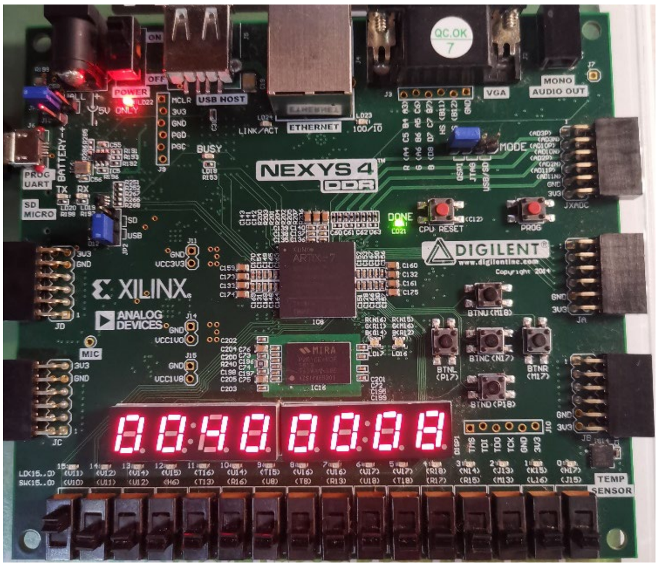
Implemented Power Report

2) 下板结果

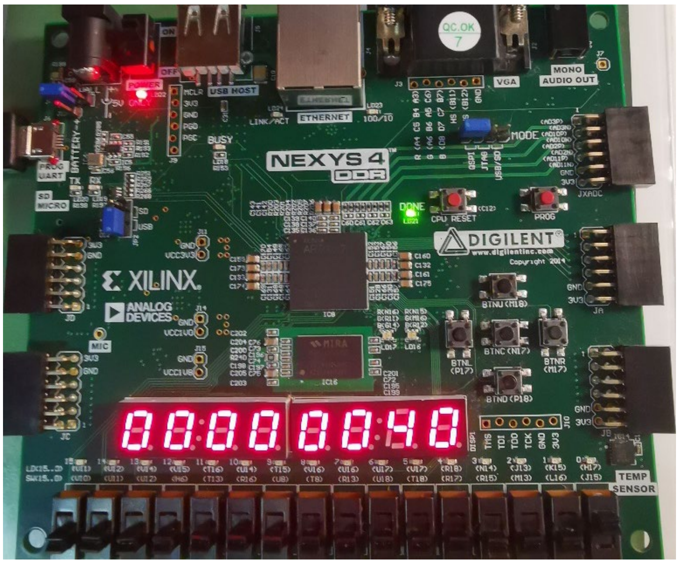
最终结果 9FD0B7EA，结果正确：



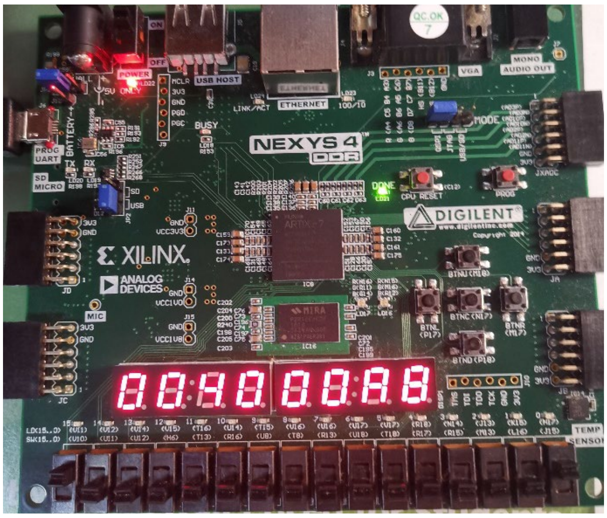
使用最左侧的开关（V10），显示最终的 PC 值，结果正确：



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$TP=n/T_{\mathrm{m}}$=2022/278.936us =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）$S=T_0/T_{\mathrm{m}}$=2022*32*40ns/278.936us=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/278.936us*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：

```
//judge read/write violation
`define VIOLATION_REGFILE_HEAD 2'b00
`define VIOLATION_CP0REG_HEAD 2'b01
`define VIOLATION_HI 7'b100_0000
`define VIOLATION_LO 7'b100_0001
`define VIOLATION_HILO 7'b100_0010
`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。

```
    wire violation1=(waddr1!=`VIOLATION_NON)&&(raddr1==waddr1||raddr2==waddr1||(waddr1==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    wire violation2=(waddr2!=`VIOLATION_NON)&&(raddr1==waddr2||raddr2==waddr2||(waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    wire violation3=(waddr3!=`VIOLATION_NON)&&(raddr1==waddr3||raddr2==waddr3||(waddr3==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    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。

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

```
            cond[1]<=`PARTS_COND_ZERO;  //ID
            cond[2]<=`PARTS_COND_FLOW;  //EX
            cond[3]<=`PARTS_COND_FLOW;  //ME
            cond[4]<=`PARTS_COND_FLOW;  //WB
            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：

```
    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)));
    assign id_ALUb_w=(!violation)&&(raddr2!=`VIOLATION_NON)&&(raddr2==waddr2||raddr2==waddr3);
    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：

```
    always @(*) begin
        if(!violation) begin
            if(id_ALUa_w) begin //use alua
                if(raddr1==waddr2||(waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO))) begin //use ex
                    case (waddr2)
                        `VIOLATION_HI: id_ALUa<=ex_HI;
                        `VIOLATION_LO: id_ALUa<=ex_LO;
                        `VIOLATION_HILO: begin
                            if (raddr1==`VIOLATION_HI)
                                id_ALUa<=ex_HI;
                            else
                                id_ALUa<=ex_LO;
                        end
                        default: begin
                            case (waddr2[6:5])
                                `VIOLATION_REGFILE_HEAD: id_ALUa<=ex_Z;
                                `VIOLATION_CP0REG_HEAD: id_ALUa<=ex_Z;
                                default: begin end
                            endcase
                        end
                    endcase
                end
                else begin //use me
                    if (me_from_mem) begin
                        id_ALUa<=me_MEM;
                    end
                    case (waddr2)
```

```
                        `VIOLATION_HI: id_ALUa<=me_HI;
                        `VIOLATION_LO: id_ALUa<=me_LO;
                        `VIOLATION_HILO: begin
                            if (raddr1==`VIOLATION_HI)
                                id_ALUa<=me_HI;
                            else
                                id_ALUa<=me_LO;
                        end
                        default: begin
                            case (waddr2[6:5])
                                `VIOLATION_REGFILE_HEAD: id_ALUa<=me_Z;
                                `VIOLATION_CP0REG_HEAD: id_ALUa<=me_Z;
                                default: begin end
                            endcase
                        end
                    endcase
                end
            end

            if (id_ALUb_w) begin    //use alub
                if (raddr2==waddr2) //use ex
                    id_ALUb<=ex_mul?ex_LO:ex_Z;
                else begin  //use me
                    if (me_from_mem)
                        id_ALUb<=me_MEM;
                    else
                        id_ALUb<=me_mul?me_LO:me_Z;
                end
            end

            if (id_Rt_w) begin    //use Rt
                if (raddr2==waddr2) //use ex
                    id_Rt<=ex_mul?ex_LO:ex_Z;
                else begin  //use me
                    if (me_from_mem)
                        id_Rt<=me_MEM;
                    else
                        id_Rt<=me_mul?me_LO:me_Z;
                end
            end
        end
    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 calculater，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

```
//----------Instruction----------

`define PARTS_COND_FLOW 2'b00
`define PARTS_COND_STALL 2'b01
`define PARTS_COND_ZERO 2'b10
`define PC_ADDR_INIT 32'h00400000
`define IR_NON 32'hffffffff
`define IR_NON_31_26 6'b111111

//----------Operation----------

`define CAL_MULTU 2'b00
`define CAL_MULT 2'b01
`define CAL_DIVU 2'b10
`define CAL_DIV 2'b11

```

```
//----------ALU----------

`define ALU_ADDU 4'b0000
`define ALU_ADD 4'b0010
`define ALU_SUBU 4'b0001
`define ALU_SUB 4'b0011
`define ALU_AND 4'b0100
`define ALU_OR 4'b0101
`define ALU_XOR 4'b0110
`define ALU_NOR 4'b0111
`define ALU_LUI 4'b1000
`define ALU_SLT 4'b1011
`define ALU_SLTU 4'b1010
`define ALU_SRA 4'b1100
`define ALU_SLL 4'b1110
`define ALU_SLA 4'b1111
`define ALU_SRL 4'b1101
`define ALU_CLZ 4'b1001

//----------CP0----------

`define EXC_SYSCALL 4'b1000
`define EXC_BREAK 4'b1001
`define EXC_TEQ 4'b1101

//----------Extend----------

`define EXT5_Z 3'b000
`define EXT16_SL2_S 3'b001
`define EXT16_Z 3'b010
`define EXT16_S 3'b011
`define EXT8_Z 3'b100
`define EXT8_S 3'b101
`define EXT32_NON 3'b110

//----------Mux----------

`define MUX_PC_NPCEXT 3'b000
`define MUX_PC_RS 3'b001
`define MUX_PC_INTR_ADDR 3'b010
`define MUX_PC_EPC 3'b011
`define MUX_PC_CONNECT 3'b100
`define MUX_PC_NPC 3'b101

`define MUX_ALUa_HI 3'b000
`define MUX_ALUa_LO 3'b001
`define MUX_ALUa_NPC 3'b010
`define MUX_ALUa_RS 3'b011
`define MUX_ALUa_EXT 3'b100
`define MUX_ALUa_CP0 3'b101
`define MUX_ALUa_IMM0 3'b110

`define MUX_ALUb_IMM0 2'b00
`define MUX_ALUb_RT 2'b01
`define MUX_ALUb_EXT 2'b10

`define MUX_RDC_IR_15_11 2'b00
`define MUX_RDC_IR_20_16 2'b01
```

```
`define MUX_RDC_IMM31 2'b10

`define MUX_RD_Z 2'b00
`define MUX_RD_MEM 2'b01
`define MUX_RD_HI 2'b10
`define MUX_RD_LO 2'b11

//----------RAM----------

`define RAM_WIDTH_32 2'b00
`define RAM_WIDTH_16 2'b01
`define RAM_WIDTH_8 2'b10

//----------CtrlUnit----------

`define FLOW_NORMAL 2'b00
`define FLOW_OVERFLOW 2'b01
`define FLOW_MULDIV 2'b10
`define FLOW_VIOLATE 2'b11

//----------Others----------

//judge read/write violation
`define VIOLATION_REGFILE_HEAD 2'b00
`define VIOLATION_CP0REG_HEAD 2'b01
`define VIOLATION_HI 7'b100_0000
`define VIOLATION_LO 7'b100_0001
`define VIOLATION_HILO 7'b100_0010
`define VIOLATION_NON 7'b110_0000
```

2) alu.v

```
`include "define.vh"

module alu(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output reg [31:0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
    reg [1:0] carry2;
    reg [31:0] tmp;
    always@(*)
    begin
        case(aluc)
            `ALU_ADDU:    //unsigned +
            begin
                {carry,r[31:0]}={1'b0,a}+{1'b0,b};
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_ADD:    //signed +
            begin
                {carry2[0],r}={1'b0,a}+{1'b0,b};
                zero=(0==r)?1:0;
                negative=r[31];
                overflow=(r[31]+a[31]+b[31])^carry2[0];
            end
            `ALU_SUBU:    //unsigned -
            begin
                {carry,r[31:0]}={1'b0,a}-{1'b0,b};
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_SUB:    //signed -
            begin
                {carry2[0],r}={1'b0,a}-{1'b0,b};
                zero=(0==r)?1:0;
                negative=r[31];
                overflow=(r[31]+a[31]+b[31])^carry2[0];
            end

            `ALU_AND:    //and
            begin
                r=a&b;
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_OR:    //or
            begin
                r=a|b;
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_XOR:    //xor
            begin
                r=a^b;
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_NOR:    //nor
            begin
                r=~(a|b);
                zero=(0==r)?1:0;
                negative=r[31];
            end

            `ALU_LUI:    //lui
            begin
                r={b[15:0],16'b0};
                zero=(0==r)?1:0;
                negative=r[31];
            end
            `ALU_SLT:    //signed <
            begin
                {carry2[0],tmp}={a[31],a}-{b[31],b};
                zero=(a==b)?1:0;
                negative=carry2[0];
                r={31'b0,negative};
            end
            `ALU_SLTU:    //unsigned <
            begin
                carry=(a<b)?1:0;
                zero=(a==b)?1:0;
                negative=0;
                r={31'b0,carry};
            end

            `ALU_SRA:    //shiftR arithmetic
            begin
                if(a[4:0]>0)
                begin
                    carry=b[a[4:0]-1];
                    tmp={32{b[31]}};
                    r={tmp,b}>>a[4:0];
                end
                else
                  begin
                      carry=b[0];
                      r=b;    //fix
                  end
                  zero=(0==r)?1:0;
                  negative=r[31];
              end
              `ALU_SLL,`ALU_SLA:    //shiftL
              begin
                  if(a[4:0]>0)
                      carry=b[32-a[4:0]];
                  else
                  begin
                      carry=b[31];
                      r=b;    //fix
                  end
                  r=b<<a[4:0];
                  zero=(0==r)?1:0;
                  negative=r[31];
              end
              `ALU_SRL:    //shiftR logic
              begin
                  if(a[4:0]>0)
                      carry=b[a[4:0]-1];
                  else
                  begin
                      carry=b[0];
                      r=b;    //fix
                  end
                  r=b>>a[4:0];
                  zero=(0==r)?1:0;
                  negative=r[31];
              end
              `ALU_CLZ:    //count leading zero
              begin
                  if(a==32'b0) begin
                      r=32'd32;
                  end
                  else begin
                      r[31:5]=27'b0;
                      tmp[31:0]=a;
                      if(tmp[31:16]==16'b0) begin
                          r[4]=1;
                      end
```

```
                    else begin
                        r[4]=0;
                        tmp[15:0]=tmp[31:16];
                    end
                    if(tmp[15:8]==8'b0) begin
                        r[3]=1;
                    end
                    else begin
                        r[3]=0;
                        tmp[7:0]=tmp[15:8];
                    end
                    if(tmp[7:4]==4'b0) begin
                        r[2]=1;
                    end
                    else begin
                        r[2]=0;
                        tmp[3:0]=tmp[7:4];
                    end
                    if(tmp[3:2]==2'b0) begin
                        r[1]=1;
                    end
                    else begin
                        r[1]=0;
                        tmp[1:0]=tmp[3:2];
                    end
                    r[0]=(tmp[1]==1'b0)?1:0;
                end
            end
            default:begin end
        endcase
    end
endmodule
```

3) calculator.v

```
`include "define.vh"

module calculator(
    input clk,
    input [31:0] a, //multiplicand/dividend
    input [31:0] b, //multiplier/divisor
    input [1:0] calc,
    input reset,      //high-active,at the beginning of test
    input ena,  //high-active
    output reg [31:0] oLO,
    output reg [31:0] oHI,
    output reg sum_finish,

    //for program in
    input cpu_stall
    );

    wire cal_clk=~clk;
    reg start,busy;
    wire busyDivu,busyDiv,busyMult,busyMultu,finishDivu,finishDiv,finishMult,finishMultu;
    wire [63:0] oMult,oMultu,oDiv,oDivu;

    reg [1:0] inner_calc;
    always @(negedge clk) begin
        inner_calc<=calc;
    end

    always @(*) begin
        case (inner_calc)
           `CAL_MULTU: begin
               oLO<=oMultu[31:0];
               oHI<=oMultu[63:32];
               sum_finish<=finishMultu;
           end
           `CAL_MULT: begin
               oLO<=oMult[31:0];
               oHI<=oMult[63:32];
               sum_finish<=finishMult;
           end
           `CAL_DIVU: begin
               oLO<=oDivu[31:0];
               oHI<=oDivu[63:32];
               sum_finish<=finishDivu;
           end
           `CAL_DIV: begin
               oLO<=oDiv[31:0];
               oHI<=oDiv[63:32];
               sum_finish<=finishDiv;
           end
           default: begin end
        endcase
    end

        always @(*) begin
        case (calc)
           `CAL_MULTU: begin
               busy<=busyMultu;
           end
           `CAL_MULT: begin
               busy<=busyMult;
           end
           `CAL_DIVU: begin
               busy<=busyDivu;
           end
           `CAL_DIV: begin
               busy<=busyDiv;
           end
           default: begin end
        endcase
    end

    always @(posedge ena or posedge busy or posedge reset) begin
        if (reset) begin
           start<=0;
        end else begin
           if (busy)
               start<=0;
           else if(ena)
               start<=1;
        end
    end

    (* KEEP = "{TRUE|FALSE|SOFT}" *) wire start_mult,start_multu,start_div,start_divu;
    assign start_mult=start&&(calc==`CAL_MULT);
    assign start_multu=start&&(calc==`CAL_MULTU);
    assign start_div=start&&(calc==`CAL_DIV);
    assign start_divu=start&&(calc==`CAL_DIVU);
    MULT MULT_inst(
        .clk(cal_clk),
        .reset(reset),    //active high
        .start(start_mult),
        .a(a), //multiplicand
        .b(b), //multiplier
        .z(oMult),
        .busy(busyMult),
        .finish(finishMult),
        .cpu_stall(cpu_stall)
    );
    MULTU MULTU_inst(
        .clk(cal_clk),
        .reset(reset),    //active high
        .start(start_multu),
        .a(a), //multiplicand
        .b(b), //multiplier
        .z(oMultu),
        .busy(busyMultu),
        .finish(finishMultu),
        .cpu_stall(cpu_stall)
    );
    DIV DIV_inst(
        .dividend(a),
        .divisor(b),
        .start(start_div),
        .clock(cal_clk),
        .reset(reset),    //active-high
        .q(oDiv[31:0]),
        .r(oDiv[63:32]),
        .busy(busyDiv),
        .finish(finishDiv),
        .cpu_stall(cpu_stall)
    );
    DIVU DIVU_inst(
        .dividend(a),
        .divisor(b),
        .start(start_divu),
        .clock(cal_clk),
        .reset(reset),    //active-high
        .q(oDivu[31:0]),
        .r(oDivu[63:32]),
        .busy(busyDivu),
        .finish(finishDivu),
        .cpu_stall(cpu_stall)
    );
endmodule
```

4) controller.v

```
`include "define.vh"

```

```
module controller(
    input [31:0] inst,
    input judge_beq, //judge BEQ BNE condition to jump
    input judge_bgez, //judge BGEZ condition to jump

    /*-----------flow_control-----------*/
    output reg [6:0] raddr1,output reg [6:0] raddr2,output reg [6:0] waddr,output reg dmem_r,

    /*-----------control-----------*/
    //ID
    output reg [2:0] mux_pc_sel,
    output reg [2:0] mux_ALUa_sel,output reg [1:0] mux_ALUb_sel,output reg [2:0] ext_sel,
    output reg cp0_exception,output reg cp0_eret,output reg [4:0] cp0_cause,
    //EX
    output reg [3:0] alu_sel,output reg [1:0] cal_sel,output reg cal_ena/*also for stall*/,output reg use_overflow,
    //ME
    output reg dmem_w,output reg [1:0] dmem_width,output reg [2:0] mem_sel,
    //WB
    output reg [1:0] mux_Rdc_sel,output reg [1:0] mux_Rd_sel,
    output reg hi_w,output reg lo_w,output reg regfile_w,output reg cp0_w
    );

    always @(*) begin
        case (inst[31:26])
            `IR_NON_31_26: begin    //do nothing
                //FLOW
                raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
                alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                //WB
                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
            end
            6'b000000: begin
                case (inst[5:0])
                    6'b100000: begin    //add
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
```

```
                      //EX
                      alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
                      dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                      //WB
                      mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                      hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                  end
                  6'b100001: begin    //addu
                      //FLOW
                      raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                      //IF
                      mux_pc_sel<=`MUX_PC_NPC;
                      //ID
                      mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                      cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                      //EX
                      alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                      dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                      //WB
                      mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                      hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                  end
                  6'b100010: begin    //sub
                      //FLOW
                      raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                      //IF
                      mux_pc_sel<=`MUX_PC_NPC;
                      //ID
                      mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                      cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                      //EX
                      alu_sel<=`ALU_SUB;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
                      dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                      //WB
                      mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                      hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                  end
                  6'b100011: begin    //subu
                      //FLOW
                      raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                      //IF
                      mux_pc_sel<=`MUX_PC_NPC;
                      //ID
                      mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                      cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                      //EX
```

```
                        alu_sel<=`ALU_SUBU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                      dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b100100: begin    //and
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b100101: begin    //or
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b100110: begin    //xor
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
```

```
alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
            //WB
            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
          end
          6'b100111: begin    //nor
            //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
            //IF
            mux_pc_sel<=`MUX_PC_NPC;
            //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
            //EX
alu_sel<=`ALU_NOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
            //WB
            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
          end
          6'b101010: begin    //slt
            //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
            //IF
            mux_pc_sel<=`MUX_PC_NPC;
            //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
            //EX
alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
            dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
            //WB
            mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
            hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
          end
          6'b101011: begin    //sltu
            //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
            //IF
            mux_pc_sel<=`MUX_PC_NPC;
            //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
            cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
            //EX
```

```
                        alu_sel<=`ALU_SLTU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000000: begin    //sll
                        //FLOW
                        raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000010: begin    //srl
                        //FLOW
                        raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000011: begin    //sra
                        //FLOW
                        raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000100: begin    //sllv
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000110: begin    //srlv
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b000111: begin    //srav
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
```

```
alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b001000: begin    //jr
                        //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_RS;
                        //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b011010: begin    //div
                        //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_HILO;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
alu_sel<=`ALU_SLL;cal_sel<=`CAL_DIV;cal_ena<=1'b1;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b011011: begin    //divu
                        //FLOW
raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_HILO;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SLL;cal_sel<=`CAL_DIVU;cal_ena<=1'b1;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b011001: begin    //multu
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_HILO;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULTU;cal_ena<=1'b1;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b001001: begin    //jarl
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_RS;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b001101: begin    //break
                        //FLOW
                        raddr1<={`VIOLATION_CP0REG_HEAD,5'd12};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b001100: begin    //syscall
                        //FLOW
                        raddr1<={`VIOLATION_CP0REG_HEAD,5'd12};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<=`EXC_SYSCALL;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b010000: begin    //mfhi
                        //FLOW
                        raddr1<=`VIOLATION_HI;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_HI;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b010010: begin    //mflo
                        //FLOW
                        raddr1<=`VIOLATION_LO;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_LO;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                    end
                    6'b010001: begin    //mthi
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_HI;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b1;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b010011: begin    //mtlo
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_LO;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    6'b110100: begin    //teq
                        //FLOW
                        raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=judge_beq;cp0_eret<=1'b0;cp0_cause<=`EXC_TEQ;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                    default: begin    //do nothing
                        //FLOW
                        raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                        //IF
                        mux_pc_sel<=`MUX_PC_NPC;
                        //ID
                        mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                        //EX
                        alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                        //WB
                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                    end
                endcase
            end
            6'b001000: begin    //addi
                //FLOW
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
                alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                //WB
                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
            end
            6'b001001: begin    //addiu
                //FLOW
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
                alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                //WB
                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
            end
            6'b001100: begin    //andi
                //FLOW
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
                alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                //WB
                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
            end
            6'b001101: begin    //ori
                //FLOW
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
                alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                //WB
                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
            end
            6'b001110: begin    //xori
                //FLOW
                raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                //IF
                mux_pc_sel<=`MUX_PC_NPC;
                //ID
                mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                //EX
```

```
                    alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b001111: begin    //lui
                    //FLOW
                    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_LUI;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b100011: begin    //lw
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b101011: begin    //sw
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b000100: begin    //beq
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=judge_beq?`MUX_PC_NPCEXT:`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b000101: begin    //bne
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=judge_beq?`MUX_PC_NPC:`MUX_PC_NPCEXT;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b001010: begin    //slti
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b001011: begin    //sltiu
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_SLTU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b000010: begin    //j
                    //FLOW
                    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_CONNECT;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b000011: begin    //jal
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,5'd31};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_CONNECT;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b000001: begin    //bgez
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=judge_bgez?`MUX_PC_NPCEXT:`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT16_SL2_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b100100: begin    //lbu
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b100101: begin    //lhu
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b100000: begin    //lb
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b100001: begin    //lh
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b1;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b101000: begin    //sb
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b101001: begin    //sh
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                6'b010000: begin
                    case (inst[25:21])
                5'b10000: begin     //eret
                    //FLOW
                    raddr1<={`VIOLATION_CP0REG_HEAD,5'd14};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_EPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b1;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                5'b00000: begin     //mfc0
                    //FLOW
                    raddr1<={`VIOLATION_CP0REG_HEAD,inst[15:11]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[20:16]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_CP0;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                5'b00100: begin     //mtc0
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[20:16]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_CP0REG_HEAD,inst[15:11]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_IMM0;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b1;
                end
                default: begin    //do nothing
                    //FLOW
                    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                    endcase
                end
                6'b011100: begin
                    case (inst[5:0])
                6'b000010: begin    //mul
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b1;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_LO;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                6'b100000: begin    //clz
                    //FLOW
                    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_CLZ;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
                end
                default: begin    //do nothing
                    //FLOW
                    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
                    endcase
                end
                default: begin    //do nothing
                    //FLOW
                    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;dmem_r<=1'b0;
                    //IF
                    mux_pc_sel<=`MUX_PC_NPC;
                    //ID
                    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
                    cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
                    //EX
                    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
                    dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
                    //WB
                    mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
                    hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
                end
            endcase
        end
    endmodule
```

5) cp0.v

```
`include "define.vh"

module CP0(
    input clk,
    input rst,
    input mtc0, //cpu instruction mtc0,high-active
    input [31:0] pc,

    input [4:0] waddr, //specifies CP0 reg to write
    input [31:0] wdata, //data from GP reg to place CP0 reg
    input exception,    //instruction syscall,break,teq,high-active
    input eret, //instruction eret,high-active
    input [4:0] cause,

    input [4:0] raddr, //specifies CP0 reg to read
    output [31:0] rdata,    //data from CP0 reg for GP reg
    output [31:0] status,   //mask,low-active
    output [31:0] exc_addr,  //address for PC at the beginning of an exception
    output [31:0] intr_addr    //exception check
    );

    reg [31:0] cp0_reg[31:0];   //regfiles
    wire [31:0] status_left,status_right;
    assign status_left=cp0_reg[12]<<5;
    assign status_right=cp0_reg[12]>>5;
    assign status=cp0_reg[12];
    assign exc_addr=cp0_reg[14];
    assign rdata=cp0_reg[raddr];

    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]));
    assign intr_addr=timer_int?32'h00400004:pc;

    always@(posedge clk,posedge rst) begin
        if(rst) begin
            cp0_reg[0]<=32'b0;
            cp0_reg[1]<=32'b0;
            cp0_reg[2]<=32'b0;
            cp0_reg[3]<=32'b0;
            cp0_reg[4]<=32'b0;
            cp0_reg[5]<=32'b0;
            cp0_reg[6]<=32'b0;
            cp0_reg[7]<=32'b0;
            cp0_reg[8]<=32'b0;
            cp0_reg[9]<=32'b0;
            cp0_reg[10]<=32'b0;
            cp0_reg[11]<=32'b0;
            cp0_reg[12]<=32'h0000_000f;        //reg_status
            cp0_reg[13]<=32'h0;        //reg_cause
            cp0_reg[14]<=32'h0;        //reg_epc
            cp0_reg[15]<=32'b0;
            cp0_reg[16]<=32'b0;
            cp0_reg[17]<=32'b0;
            cp0_reg[18]<=32'b0;
            cp0_reg[19]<=32'b0;
            cp0_reg[20]<=32'b0;
            cp0_reg[21]<=32'b0;
            cp0_reg[22]<=32'b0;
            cp0_reg[23]<=32'b0;
            cp0_reg[24]<=32'b0;
            cp0_reg[25]<=32'b0;
            cp0_reg[26]<=32'b0;
            cp0_reg[27]<=32'b0;
            cp0_reg[28]<=32'b0;
            cp0_reg[29]<=32'b0;
            cp0_reg[30]<=32'b0;
            cp0_reg[31]<=32'b0;
        end
        else if(eret) begin
            cp0_reg[12]<=status_right;
        end
        else if(exception) begin
            case(cause[3:0])
                `EXC_SYSCALL: if(status[1]&status[0]) begin  //syscall
                    cp0_reg[12]<=status_left;
                    cp0_reg[13][6:2]=cause;
                    cp0_reg[14]<=pc-32'h4;
                end
                `EXC_BREAK: if(status[2]&status[0]) begin  //break
                    cp0_reg[12]<=status_left;
                    cp0_reg[13][6:2]=cause;
                    cp0_reg[14]<=pc-32'h4;
                end
                `EXC_TEQ: if(status[3]&status[0]) begin  //teq
                    cp0_reg[12]<=status_left;
                    cp0_reg[13][6:2]=cause;
                    cp0_reg[14]<=pc-32'h4;
                end
                default: begin end
            endcase
        end
        else if(mtc0) begin
            cp0_reg[waddr]<=wdata;
        end
        else begin end
    end
endmodule
```

6) cpu_regfile.v

```
module regfile(
    input clk,  //posedge write-active
    input rst,  //active-high asynchronous
    input we,   //high:write low:read
    input [31:0] raddr1,
    input [31:0] raddr2,
    output [31:0] rdata1,
    output [31:0] rdata2,
    input [31:0] waddr,
    input [31:0] wdata,

    //for program out
    output [31:0] reg_16
    );

    (* KEEP = "{TRUE|FALSE|SOFT}" *) reg [31:0] array_reg[31:0]; //regfiles

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            array_reg[0]<=32'b0;
            array_reg[1]<=32'b0;
            array_reg[2]<=32'b0;
            array_reg[3]<=32'b0;
            array_reg[4]<=32'b0;
            array_reg[5]<=32'b0;
            array_reg[6]<=32'b0;
            array_reg[7]<=32'b0;
            array_reg[8]<=32'b0;
            array_reg[9]<=32'b0;
            array_reg[10]<=32'b0;
            array_reg[11]<=32'b0;
            array_reg[12]<=32'b0;
            array_reg[13]<=32'b0;
            array_reg[14]<=32'b0;
            array_reg[15]<=32'b0;
            array_reg[16]<=32'b0;
            array_reg[17]<=32'b0;
            array_reg[18]<=32'b0;
            array_reg[19]<=32'b0;
            array_reg[20]<=32'b0;
            array_reg[21]<=32'b0;
            array_reg[22]<=32'b0;
            array_reg[23]<=32'b0;
            array_reg[24]<=32'b0;
            array_reg[25]<=32'b0;
            array_reg[26]<=32'b0;
            array_reg[27]<=32'b0;
            array_reg[28]<=32'b0;
            array_reg[29]<=32'b0;
            array_reg[30]<=32'b0;
            array_reg[31]<=32'b0;
        end
        else if(we&&waddr) begin    //cannot modify $0
            array_reg[waddr]<=wdata;
        end
    end

    assign rdata1=array_reg[raddr1];
    assign rdata2=array_reg[raddr2];

    //for program out
    assign reg_16=array_reg[16];
endmodule
```

7) cpu_top.v

```
`include "define.vh"

module top_parts(
    input clk,  //posedge write-active
    input reset,    //active-high asynchronous
    output [31:0] top_pc,
    output [31:0] top_ir,

    //for program out
    output [31:0] reg_16,
    input cpu_stall
    );

    wire [1:0] cond0,cond1,cond2,cond3,cond4;
    wire [6:0] flow_raddr1,flow_raddr2,flow_waddr1,flow_waddr2,flow_waddr3;
    wire [31:0] forward_ALUa,forward_ALUb,forward_Rt;
    wire forward_ALUa_w,forward_ALUb_w,forward_Rt_w;

    wire [31:0] connect,npc_ext,regfile_Rs,cp0_EPC,cp0_intr_addr;
    wire [31:0] if_NPC,if_IR;
    wire [2:0] mux_pc_sel;
    assign top_ir=if_IR;

    instruction_fetch if_inst(
        .clk(clk),  //posedge write-active
        .reset(reset),    //active-high asynchronous
        .connect(connect),
        .npc_ext(npc_ext),
        .regfile_Rs(regfile_Rs),
        .cp0_EPC(cp0_EPC),
        .cp0_intr_addr(cp0_intr_addr),
        .oNPC(if_NPC),
        .rPC(top_pc),
        .rIR(if_IR),
        .cond(cond0),
        .mux_pc_sel(mux_pc_sel)
    );

    wire hi_w,lo_w,cp0_w,regfile_w;
    wire [4:0] regfile_Rdc;wire [31:0] regfile_Rd,Rd_out_for_LO;
    wire [31:0] id_ALUa,id_ALUb,id_Rt,id_IR,ext_out;
    assign npc_ext = ext_out+if_NPC;

    instruction_decode id_inst(
        .clk(clk),
        .reset(reset),
        .if_IR(if_IR),
        .if_NPC(if_NPC),
        .regfile_Rdc(regfile_Rdc),    //also cp0
        .regfile_Rd(regfile_Rd),    //also cp0
        .Rd_out_for_LO(Rd_out_for_LO),
        .rALUa(id_ALUa),
        .rALUb(id_ALUb),
        .rRt(id_Rt),
        .rIR(id_IR),
        .cp0_EPC(cp0_EPC),
        .cp0_intr_addr(cp0_intr_addr),
        .ext_out(ext_out),
        .connect(connect),
        .regfile_Rs(regfile_Rs),
        .regfile_Rt(),
        .cond(cond1),
        .mux_pc_sel(mux_pc_sel),
        .hi_w(hi_w),
        .lo_w(lo_w),
        .regfile_w(regfile_w),
        .cp0_w(cp0_w),
        .flow_raddr1(flow_raddr1),
        .flow_raddr2(flow_raddr2),
        .forward_ALUa(forward_ALUa),
        .forward_ALUb(forward_ALUb),
        .forward_Rt(forward_Rt),
        .forward_ALUa_w(forward_ALUa_w),
        .forward_ALUb_w(forward_ALUb_w),
        .forward_Rt_w(forward_Rt_w),

        //for program out
        .reg_16(reg_16)
    );

    wire [31:0] ex_HI,ex_LO,ex_Z,ex_Rt,ex_IR;
    wire mult_div_stall,cal_finish,overflow_stall;

    execute ex_inst(
        .clk(clk),
        .reset(reset),
        .alua(id_ALUa),
        .alub(id_ALUb),
        .id_Rt(id_Rt),
        .id_IR(id_IR),
        .rHI(ex_HI),
        .rLO(ex_LO),
        .rZ(ex_Z),
        .rRt(ex_Rt),
        .rIR(ex_IR),
        .cond(cond2),
        .mult_div_stall(mult_div_stall),   //cause controller to stall 32 periods
        .cal_finish(cal_finish),
        .overflow_stall(overflow_stall),
        .flow_waddr(flow_waddr1),

        //for program in
        .cpu_stall(cpu_stall)
    );

    wire [31:0] me_HI,me_LO,me_Z,me_MEM,me_IR;
    wire ex_from_mem,me_from_mem;
    wire ex_mul,me_mul;

    memory_access me_inst(
        .clk(clk),
        .reset(reset),
        .ex_HI(ex_HI),
        .ex_LO(ex_LO),
        .ex_Z(ex_Z),
        .ex_Rt(ex_Rt),
        .ex_IR(ex_IR),
        .rHI(me_HI),
        .rLO(me_LO),
        .rZ(me_Z),
        .rMEM(me_MEM),
        .rIR(me_IR),
        .cond(cond3),
        .flow_waddr(flow_waddr2),
        .dmem_r(ex_from_mem),
        .use_mul(ex_mul)
    );

    write_back wb_inst(
        .clk(clk),
        .reset(reset),
        .me_HI(me_HI),
        .me_LO(me_LO),
        .me_Z(me_Z),
        .me_MEM(me_MEM),
        .me_IR(me_IR),
        .mux_Rdc_out(regfile_Rdc),
        .mux_Rd_out(regfile_Rd),
        .Rd_out_for_LO(Rd_out_for_LO),
        .cond(cond4),
        .hi_w(hi_w),
        .lo_w(lo_w),
        .cp0_w(cp0_w),
        .regfile_w(regfile_w),
        .flow_waddr(flow_waddr3),
        .dmem_r(me_from_mem),
        .use_mul(me_mul)
    );

    flow_control flow_control_inst(
        .clk(clk),.reset(reset),
        .raddr1(flow_raddr1),.raddr2(flow_raddr2),.waddr1(flow_waddr1),.waddr2(flow_waddr2),.waddr3(flow_waddr3),
        .mult_div_stall(mult_div_stall),.mult_div_over(cal_finish),.overflow_stall(overflow_stall),
        /*-----------flow_control-----------*/
        .cond0(cond0),.cond1(cond1),.cond2(cond2),.cond3(cond3),.cond4(cond4),
        /*-----------flow_data-----------*/
        .id_ALUa(forward_ALUa),.id_ALUb(forward_ALUb),.id_Rt(forward_Rt),.id_ALUa_w(forward_ALUa_w),.id_ALUb_w(forward_ALUb_w),.id_Rt_w(forward_Rt_w),
        .ex_HI(ex_HI),.ex_LO(ex_LO),.ex_Z(ex_Z),.ex_from_mem(ex_from_mem),.ex_mul(ex_mul),
        .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),
        //for program in
        .cpu_stall(cpu_stall)
    );
endmodule
```

8) DIV.v

```
module DIV(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,
    input clock,
    input reset,    //active-high
    output [31:0] q,
    output [31:0] r,
    output reg busy,
    output reg finish,

    //for program in
    input cpu_stall
    );

    reg sign_dnd,sign_vsr;
    wire [31:0] udividend,udivisor;
    wire [31:0] uq,ur;
    assign udividend=dividend[31]?(~dividend+1'b1):dividend;
    assign udivisor=divisor[31]?(~divisor+1'b1):divisor;
    assign q=(sign_dnd^sign_vsr)?(~uq+1'b1):uq;
    assign r=sign_dnd?(~ur+1'b1):ur;

    reg [5:0] cnt;
    reg [32:0] inner_sr;
    reg [31:0] rmdr;    //remainder
    reg [31:0] qtnt;    //quotient
    reg sign;

    wire [32:0] inner_complement_sr;
    assign inner_complement_sr=~inner_sr+1'b1;
    wire [32:0] add;
    assign add={rmdr,qtnt[31]}+(sign?inner_complement_sr:inner_sr);

    assign ur=rmdr;
    assign uq=qtnt;

    always@(posedge(clock) or posedge(reset)) begin
        if(reset==1) begin
            cnt<=0;
            busy<=0;
            rmdr<=0;
            qtnt<=0;
            inner_sr<=0;
            sign<=0;

            sign_dnd<=0;
            sign_vsr<=0;
            finish<=0;
        end
        else begin
            if(start) begin
                rmdr<=0;
                qtnt<=udividend;
                inner_sr<={1'b0,udivisor};
                busy<=1;
                cnt<=1;
                sign<=1;

                sign_dnd<=dividend[31];
                sign_vsr<=divisor[31];
                finish<=0;
            end
            else if(busy) begin
                if (!cpu_stall) begin
                    case(cnt)
                        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
                            {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]};
                            sign<=~add[32];
                            cnt<=cnt+1;
                            finish<=0;
                        end
                        32: begin
                            {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]}+(add[32]?{inner_sr[31:0],32'b0}:64'b0);
                            sign<=~add[32];
                            cnt<=cnt+1;
                            busy<=0;
                            finish<=1;
                        end
                        default: begin end
                    endcase
                end
            end
            else
                finish<=0;
        end
    end
endmodule
```

9) DIVU.v

```
module DIVU(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,
    input clock,
    input reset,    //active-high
    output [31:0] q,
    output [31:0] r,
    output reg busy,
    output reg finish,

    //for program in
    input cpu_stall
    );

    reg [5:0] cnt;
    reg [32:0] inner_sr;
    reg [31:0] rmdr;    //remainder
    reg [31:0] qtnt;    //quotient
    reg sign;

    wire [32:0] inner_complement_sr;
    assign inner_complement_sr=~inner_sr+1'b1;
    wire [32:0] add;
    assign add={rmdr,qtnt[31]}+(sign?inner_complement_sr:inner_sr);

    assign r=rmdr;
    assign q=qtnt;

    always@(posedge(clock) or posedge(reset)) begin
        if(reset==1) begin
            cnt<=0;
            busy<=0;
            rmdr<=0;
            qtnt<=0;
            inner_sr<=0;
            sign<=0;
            finish<=0;
        end
        else begin
            if(start) begin
                rmdr<=0;
                qtnt<=dividend;
                inner_sr<={1'b0,divisor};
                busy<=1;
                cnt<=1;
                sign<=1;
                finish<=0;
            end
            else if(busy) begin
                if (!cpu_stall) begin
                    case(cnt)
                        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
                            {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]};
                            sign<=~add[32];
                            cnt<=cnt+1;
                            finish<=0;
                        end
                        32: begin
                            {rmdr,qtnt}<={add[31:0],qtnt[30:0],~add[32]}+(add[32]?{inner_sr[31:0],32'b0}:64'b0);
                            sign<=~add[32];
                            cnt<=cnt+1;
                            busy<=0;
                            finish<=1;
                        end
                        default: begin end
                    endcase
                end
            end
            else
                finish<=0;
        end
    end
endmodule
```

10) extend.v

```
`include "define.vh"

module ext(
    input [2:0] ext_switch,

    input [4:0] iData_len5,
    input [7:0] iData_len8,
    input [15:0] iData_len16,
    input [31:0] iData_len32,
    output reg [31:0] oData
);

always @(*) begin
    case (ext_switch)
        `EXT5_Z: oData<={27'b0,iData_len5};
        `EXT16_SL2_S: oData=iData_len16[15]?{14'h3fff,iData_len16,2'b0}:{14'b0,iData_len16,2'b0};
        `EXT16_Z: oData<={16'b0,iData_len16};
        `EXT16_S: oData<={{16{iData_len16[15]}},iData_len16};
        `EXT8_Z: oData<={24'b0,iData_len8};
        `EXT8_S: oData<={{24{iData_len8[7]}},iData_len8};
        `EXT32_NON: oData<=iData_len32;
        default: oData<=32'b0;
    endcase
end
endmodule
```

11) flow_control.v

```
`include "define.vh"

module flow_control(
    input clk,input reset,
    input [6:0] raddr1,input [6:0] raddr2,input [6:0] waddr1,input [6:0] waddr2,input [6:0] waddr3,
    input mult_div_stall,input mult_div_over,input overflow_stall,

    /*-----------flow_control-----------*/
    output [1:0] cond0,output [1:0] cond1,output [1:0] cond2,output [1:0] cond3,output [1:0] cond4,

    /*-----------flow_data-----------*/
    output reg [31:0] id_ALUa,output reg [31:0] id_ALUb,output reg [31:0] id_Rt,output id_ALUa_w,output id_ALUb_w,output id_Rt_w,
    input [31:0] ex_HI,input [31:0] ex_LO,input [31:0] ex_Z,input ex_from_mem,input ex_mul,
    input [31:0] me_HI,input [31:0] me_LO,input [31:0] me_Z,input [31:0] me_MEM,input me_from_mem,input me_mul,

    //for program in
    input cpu_stall
    );

    reg [1:0] cond[0:4];
    assign cond0=cond[0];
    assign cond1=cond[1];
    assign cond2=cond[2];
    assign cond3=cond[3];
    assign cond4=cond[4];

    reg [1:0] state,nextstate;reg [4:0] cnt;
    wire violation1=(waddr1!=`VIOLATION_NON)&&(raddr1==waddr1||raddr2==waddr1||(waddr1==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    wire violation2=(waddr2!=`VIOLATION_NON)&&(raddr1==waddr2||raddr2==waddr2||(waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    wire violation3=(waddr3!=`VIOLATION_NON)&&(raddr1==waddr3||raddr2==waddr3||(waddr3==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_HI||raddr2==`VIOLATION_LO)));
    wire violation=violation1|(violation2&ex_from_mem);

    always @(posedge clk or posedge reset) begin
        if(reset)
            state<=`FLOW_NORMAL;
        else
            state<=nextstate;
    end

    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)));
    assign id_ALUb_w=(!violation)&&(raddr2!=`VIOLATION_NON)&&(raddr2==waddr2||raddr2==waddr3);
    assign id_Rt_w=id_ALUb_w;
    always @(*) begin
        if(!violation) begin
            if(id_ALUa_w) begin //use alua
                if(raddr1==waddr2||(waddr2==`VIOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO))) begin //use ex
                    case (waddr2)
                        `VIOLATION_HI: id_ALUa<=ex_HI;
                        `VIOLATION_LO: id_ALUa<=ex_LO;
                        `VIOLATION_HILO: begin
                            if (raddr1==`VIOLATION_HI)
                                id_ALUa<=ex_HI;
                            else
                                id_ALUa<=ex_LO;
                        end
                        default: begin
                            case (waddr2[6:5])
                                `VIOLATION_REGFILE_HEAD: id_ALUa<=ex_Z;
                                `VIOLATION_CP0REG_HEAD: id_ALUa<=ex_Z;
                                default: begin end
                            endcase
                        end
                    endcase
                end
                else begin //use me
                    if (me_from_mem) begin
```

```
                      id_ALUa<=me_MEM;
                  end
                  case (waddr2)
                      `VIOLATION_HI: id_ALUa<=me_HI;
                      `VIOLATION_LO: id_ALUa<=me_LO;
                      `VIOLATION_HILO: begin
                          if (raddr1==`VIOLATION_HI)
                              id_ALUa<=me_HI;
                          else
                              id_ALUa<=me_LO;
                      end
                      default: begin
                          case (waddr2[6:5])
                              `VIOLATION_REGFILE_HEAD: id_ALUa<=me_Z;
                              `VIOLATION_CP0REG_HEAD: id_ALUa<=me_Z;
                              default: begin end
                          endcase
                      end
                  endcase
              end
          end

          if (id_ALUb_w) begin    //use alub
              if (raddr2==waddr2) //use ex
                  id_ALUb<=ex_mul?ex_LO:ex_Z;
              else begin  //use me
                  if (me_from_mem)
                      id_ALUb<=me_MEM;
                  else
                      id_ALUb<=me_mul?me_LO:me_Z;
              end
          end

            if (id_Rt_w) begin    //use Rt
                if (raddr2==waddr2) //use ex
                    id_Rt<=ex_mul?ex_LO:ex_Z;
                else begin  //use me
                    if (me_from_mem)
                        id_Rt<=me_MEM;
                    else
                        id_Rt<=me_mul?me_LO:me_Z;
                end
            end
        end
    end

    always @(*) begin
        if(cpu_stall)begin
            cond[0]<=`PARTS_COND_STALL;  //IF
            cond[1]<=`PARTS_COND_STALL;  //ID
            cond[2]<=`PARTS_COND_STALL;  //EX
            cond[3]<=`PARTS_COND_STALL;  //ME
            cond[4]<=`PARTS_COND_STALL;  //WB
            nextstate<=state;
        end else begin
            case (state)
                `FLOW_NORMAL: begin
                    if (overflow_stall) begin
```

```
                        if (violation) begin
                            cond[0]<=`PARTS_COND_FLOW;  //IF
                            cond[1]<=`PARTS_COND_ZERO;  //ID
                            cond[2]<=`PARTS_COND_ZERO;  //EX
                            cond[3]<=`PARTS_COND_FLOW;  //ME
                            cond[4]<=`PARTS_COND_FLOW;  //WB
                            nextstate<=`FLOW_NORMAL;
                        end else begin
                            cond[0]<=`PARTS_COND_FLOW;  //IF
                            cond[1]<=`PARTS_COND_FLOW;  //ID
                            cond[2]<=`PARTS_COND_ZERO;  //EX
                            cond[3]<=`PARTS_COND_FLOW;  //ME
                            cond[4]<=`PARTS_COND_FLOW;  //WB
                            nextstate<=`FLOW_NORMAL;
                        end
                    end
                    else if (!mult_div_over&mult_div_stall) begin
                        cond[0]<=`PARTS_COND_STALL;  //IF
                        cond[1]<=`PARTS_COND_STALL;  //ID
                        cond[2]<=`PARTS_COND_STALL;  //EX
                        cond[3]<=`PARTS_COND_STALL;  //ME
                        cond[4]<=`PARTS_COND_STALL;  //WB
                        nextstate<=`FLOW_MULDIV;
                    end
                    else if (violation) begin
                        cond[0]<=`PARTS_COND_STALL;  //IF
                        cond[1]<=`PARTS_COND_ZERO;  //ID
                        cond[2]<=`PARTS_COND_FLOW;  //EX
                        cond[3]<=`PARTS_COND_FLOW;  //ME
                        cond[4]<=`PARTS_COND_FLOW;  //WB
                        nextstate<=`FLOW_NORMAL;
                    end else begin
                        cond[0]<=`PARTS_COND_FLOW;  //IF
                        cond[1]<=`PARTS_COND_FLOW;  //ID
                        cond[2]<=`PARTS_COND_FLOW;  //EX
                        cond[3]<=`PARTS_COND_FLOW;  //ME
                        cond[4]<=`PARTS_COND_FLOW;  //WB
                        nextstate<=`FLOW_NORMAL;
                    end
                end
                `FLOW_MULDIV: begin
                    if (mult_div_over) begin
                        if (violation) begin
                            cond[0]<=`PARTS_COND_STALL;  //IF
                            cond[1]<=`PARTS_COND_ZERO;  //ID
                            cond[2]<=`PARTS_COND_FLOW;  //EX
                            cond[3]<=`PARTS_COND_FLOW;  //ME
                            cond[4]<=`PARTS_COND_FLOW;  //WB
                            nextstate<=`FLOW_NORMAL;
                        end else begin
                            cond[0]<=`PARTS_COND_FLOW;  //IF
                            cond[1]<=`PARTS_COND_FLOW;  //ID
                            cond[2]<=`PARTS_COND_FLOW;  //EX
                            cond[3]<=`PARTS_COND_FLOW;  //ME
                            cond[4]<=`PARTS_COND_FLOW;  //WB
                            nextstate<=`FLOW_NORMAL;
                        end
                    end else begin
```

```
                        cond[0]<=`PARTS_COND_STALL;  //IF
                        cond[1]<=`PARTS_COND_STALL;  //ID
                        cond[2]<=`PARTS_COND_STALL;  //EX
                        cond[3]<=`PARTS_COND_STALL;  //ME
                        cond[4]<=`PARTS_COND_STALL;  //WB
                        nextstate<=`FLOW_MULDIV;
                    end
                end
                default: begin
                    cond[0]<=`PARTS_COND_FLOW;  //IF
                    cond[1]<=`PARTS_COND_FLOW;  //ID
                    cond[2]<=`PARTS_COND_FLOW;  //EX
                    cond[3]<=`PARTS_COND_FLOW;  //ME
                    cond[4]<=`PARTS_COND_FLOW;  //WB
                    nextstate<=`FLOW_NORMAL;
                end
            endcase
        end
    end
endmodule
```

12) MULT.v

```
module MULT(
    input clk,
    input reset,    //active high
    input start,
    input [31:0] a, //multiplicand
    input [31:0] b, //multiplier
    output [63:0] z,
    output reg busy,
    output reg finish,

    //for program in
    input cpu_stall
    );

    reg [5:0] cnt;
    reg [32:0] multa,multb;
    reg [32:0] multpart;
    reg shiftr;
    wire [32:0] add;

    wire [32:0] complementa;
    assign complementa=~multa+1'b1;

    assign z={multpart,multb[32:2]};
    assign add=multpart+(multb[1]==1?(multb[0]==1?33'b0:complementa):(multb[0]==1?multa:33'b0));
    always@(posedge clk or posedge reset)
    begin
        if(reset) begin
            cnt<=0;
            multa<=0;
            multb<=0;
            multpart<=0;
            busy<=0;
            finish<=0;
        end
        else begin
            if(start) begin
                cnt<=1;
                multa<={a[31],a};
                multb<={b,1'b0};
                multpart<=0;
                busy<=1;
                finish<=0;
            end else if(busy) begin
                if (!cpu_stall) begin
                    case(cnt)
                        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
                            {multpart,multb,shiftr}<={add[32],add,multb};
                            cnt<=cnt+1;
                            finish<=0;
                        end
                        32: begin
                            {multpart,multb}<={add,multb};
                            cnt<=cnt+1;
                            busy<=0;
                            finish<=1;
                        end
                        default: begin end
                    endcase
                end
            end
            else
                finish<=0;
        end
    end
endmodule
```

13) MULTU.v

```
module MULTU(
    input clk,
    input reset,    //active high
    input start,
    input [31:0] a, //multiplicand
    input [31:0] b, //multiplier
    output [63:0] z,
    output reg busy,
    output reg finish,

    //for program in
    input cpu_stall
    );

    reg [5:0] cnt;
    reg [31:0] multa,multb;
    reg [31:0] multpart;
    reg shiftr;
    wire cf;
    wire [31:0] add;

    assign z={multpart,multb};
    assign {cf,add}={1'b0,multpart}+{1'b0,(multb[0]?multa:32'b0)};
    always@(posedge clk or posedge reset)
    begin
        if(reset) begin
            cnt<=0;
            multa<=0;
            multb<=0;
            multpart<=0;
            busy<=0;
            finish<=0;
        end
        else begin
            if(start) begin
                cnt<=1;
                multa<=a;
                multb<=b;
                multpart<=0;
                busy<=1;
                finish<=0;
            end else if(busy) begin
                if (!cpu_stall) begin
                    case(cnt)
                        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
                            {multpart,multb,shiftr}<={cf,add,multb};
                            cnt<=cnt+1;
                            finish<=0;
                        end
                        32: begin
                            {multpart,multb,shiftr}<={cf,add,multb};
                            cnt<=cnt+1;
                            busy<=0;
                            finish<=1;
                        end
                        default: begin end
                    endcase
                end
            end
            else
                finish<=0;
        end
    end
endmodule
```

14) mux.v

```
module mux_len32_sel8(
    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
    );

    //reg oData;  //wire in implementation

    always @(*) begin
        case(sel)
            3'b000: oData<=iData_1;
            3'b001: oData<=iData_2;
            3'b010: oData<=iData_3;
            3'b011: oData<=iData_4;
            3'b100: oData<=iData_5;
            3'b101: oData<=iData_6;
            3'b110: oData<=iData_7;
            3'b111: oData<=iData_8;
            default: begin end
        endcase
    end
endmodule


module mux_len32_sel4(
    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
    );

    //reg oData;  //wire in implementation

    always @(*) begin
        case(sel)
            2'b00: oData<=iData_1;
            2'b01: oData<=iData_2;
            2'b10: oData<=iData_3;
            2'b11: oData<=iData_4;
            default: begin end
        endcase
    end
endmodule

// module mux_len32_sel2(
//     input sel,
//     input [31:0] iData_1,
//     input [31:0] iData_2,
//     output reg [31:0] oData
//     );

//     //reg oData;  //wire in implementation

//     always @(*) begin
//         case(sel)
//             1'b0: oData<=iData_1;
//             1'b1: oData<=iData_2;
//             default: begin end
//         endcase
//     end
// endmodule
```

```


module mux_len5_sel4(
    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
    );

    //reg oData;  //wire in implementation

    always @(*) begin
        case(sel)
            2'b00: oData<=iData_1;
            2'b01: oData<=iData_2;
            2'b10: oData<=iData_3;
            2'b11: oData<=iData_4;
            default: begin end
        endcase
    end
endmodule
```

15) parts.v

```
`include "define.vh"

module instruction_fetch(
    input clk,  //posedge write-active
    input reset,    //active-high asynchronous

    input [31:0] connect,
    input [31:0] npc_ext,
    input [31:0] regfile_Rs,
    input [31:0] cp0_EPC,
    input [31:0] cp0_intr_addr,

    output [31:0] oNPC,
    output reg [31:0] rPC,
    output reg [31:0] rIR,

    //control signal
    input [1:0] cond,
    input [2:0] mux_pc_sel
    );

    wire [31:0] imem_out,mux_pc_out;
    assign oNPC=rPC+32'h4;

    mux_len32_sel8 mux_Pc(
        .sel(mux_pc_sel),
        .iData_1(npc_ext),
        .iData_2(regfile_Rs),
        .iData_3(cp0_intr_addr),
        .iData_4(cp0_EPC),
        .iData_5(connect),
        .iData_6(oNPC),
        .oData(mux_pc_out)
    );
    // ram imem(
    //     .clk(clk),  //posedge read/write-active
    //     .wena(1'b0), //high:write low:read
    //     .width(`RAM_WIDTH_32),  //0:word,1:hword,2:byte
    //     .addr(rPC),
    //     .data_in(32'b0),   //use zero side
    //     .data_out(imem_out)   //use zero side
    // );
    wire [31:0] imem_addr=(rPC-32'h00400000)>>2;
    imem imem_inst(
        .a(imem_addr[10:0]),
        .spo(imem_out)
    );

    always @(posedge clk or posedge reset) begin    //execute
        if (reset) begin
            rPC<=`PC_ADDR_INIT;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: rPC<=mux_pc_out;
                `PARTS_COND_STALL: rPC<=rPC;
                `PARTS_COND_ZERO: rPC<=`PC_ADDR_INIT;
                default: begin end
            endcase
        end
    end

    always @(negedge clk or posedge reset) begin    //flow
        if (reset) begin
            rIR<=`IR_NON;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: rIR<=imem_out;
                `PARTS_COND_STALL: rIR<=rIR;
                `PARTS_COND_ZERO: rIR<=`IR_NON;
                default: begin end
            endcase
        end
    end

endmodule

module instruction_decode(
    input clk,
    input reset,

    input [31:0] if_IR, //to read
    input [31:0] if_NPC,
    input [4:0] regfile_Rdc,    //also cp0
    input [31:0] regfile_Rd,    //also cp0,hi,lo
    input [31:0] Rd_out_for_LO, //add for lo *only when hi also write

    output reg [31:0] rALUa,
    output reg [31:0] rALUb,
    output reg [31:0] rRt,
    output reg [31:0] rIR,
    output [31:0] cp0_EPC,
    output [31:0] cp0_intr_addr,
    output [31:0] ext_out,
    output [31:0] connect,
    output [31:0] regfile_Rs,
    output [31:0] regfile_Rt,

    //control signal
    input [1:0] cond,
    output [2:0] mux_pc_sel,
    input hi_w,
    input lo_w,
    input regfile_w,
    input cp0_w,
    output [6:0] flow_raddr1,
    output [6:0] flow_raddr2,

    //forward
    input [31:0] forward_ALUa,
    input [31:0] forward_ALUb,
    input [31:0] forward_Rt,
    input forward_ALUa_w,
    input forward_ALUb_w,
    input forward_Rt_w,

    //for program out
    output [31:0] reg_16
    );

    assign connect={if_NPC[31:28],if_IR[25:0],2'b0};

    reg [31:0] rHI,rLO;

    wire mux_lo_sel,mux_hi_sel;
    wire [31:0] cal_lo_out,cal_hi_out,mux_lo_out,mux_hi_out;
    wire [31:0] alua,alub;
    wire [31:0] cp0_out;wire [4:0] cp0_cause;

    wire [2:0] mux_ALUa_sel;wire [1:0] mux_ALUb_sel;wire [2:0] ext_sel;
    wire cp0_eret,cp0_exception;

    wire judge_beq,judge_bgez;
    assign judge_beq=((forward_ALUa_w?forward_ALUa:regfile_Rs)==(forward_ALUb_w?forward_ALUb:regfile_Rt));
    assign judge_bgez=(forward_ALUa_w?(forward_ALUa[31]==1'b0):(regfile_Rs[31]==1'b0));

    controller controller_id(
        .inst(if_IR),
        .judge_beq(judge_beq), //judge BEQ BNE condition to jump
        .judge_bgez(judge_bgez), //judge BGEZ condition to jump

        /*-----------flow_control-----------*/
        .raddr1(flow_raddr1),.raddr2(flow_raddr2),
        /*-----------control-----------*/
        //ID
        .mux_pc_sel(mux_pc_sel),
        .mux_ALUa_sel(mux_ALUa_sel),.mux_ALUb_sel(mux_ALUb_sel),.ext_sel(ext_sel),
        .cp0_exception(cp0_exception),.cp0_eret(cp0_eret),.cp0_cause(cp0_cause)
    );

    mux_len32_sel8 mux_ALUa(
        .sel(mux_ALUa_sel),
        .iData_1(rHI),
        .iData_2(rLO),
        .iData_3(if_NPC),
        .iData_4(regfile_Rs),
        .iData_5(ext_out),
        .iData_6(cp0_out),
        .iData_7(32'b0),
        .oData(alua)
    );
    mux_len32_sel4 mux_ALUb(
        .sel(mux_ALUb_sel),
        .iData_1(32'd0),
        .iData_2(regfile_Rt),
        .iData_3(ext_out),
        .oData(alub)
    );
    ext ext_id(
        .ext_switch(ext_sel),
        .iData_len5(if_IR[10:6]),
        .iData_len8(),
        .iData_len16(if_IR[15:0]),
        .iData_len32(),
        .oData(ext_out)
    );


    regfile cpu_ref(
        .clk(clk),  //posedge write-active
        .rst(reset),  //active-high asynchronous
        .we(regfile_w),   //high:write low:read
        .raddr1({27'b0,if_IR[25:21]}),
        .raddr2({27'b0,if_IR[20:16]}),
        .rdata1(regfile_Rs),
        .rdata2(regfile_Rt),
        .waddr({27'b0,regfile_Rdc}),
        .wdata(regfile_Rd),

        //for program out
        .reg_16(reg_16)
    );
    CP0 cp0_inst(
        .clk(clk),
        .rst(reset),
        .mtc0(cp0_w), //cpu instruction mtc0,high-active
        .pc(if_NPC),
        .waddr(regfile_Rdc), //specifies CP0 reg to write
        .wdata(regfile_Rd), //data from GP reg to place CP0 reg
        .exception(cp0_exception&&(cond==`PARTS_COND_FLOW)),    //instruction syscall,break,teq,high-active
        .eret(cp0_eret&&(cond==`PARTS_COND_FLOW)), //instruction eret,high-active
        .cause(cp0_cause),
        .raddr(if_IR[15:11]), //specifies CP0 reg to read
        .rdata(cp0_out),    //data from CP0 reg for GP reg
        .exc_addr(cp0_EPC),  //address for PC at the beginning of an exception
        .intr_addr(cp0_intr_addr)
    );

    always @(posedge clk or posedge reset) begin    //execute
        if (reset) begin
            rALUa<=32'b0;
            rALUb<=32'b0;
            rRt<=32'b0;
            rHI<=32'b0;
            rLO<=32'b0;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: begin
                    rALUa<=forward_ALUa_w?forward_ALUa:alua;
                    rALUb<=forward_ALUb_w?forward_ALUb:alub;
                    rRt<=forward_Rt_w?forward_Rt:regfile_Rt;
                end
                `PARTS_COND_STALL: begin
                    rALUa<=rALUa;
                    rALUb<=rALUb;
                    rRt<=rRt;
                end
                `PARTS_COND_ZERO: begin
                    rALUa<=32'b0;
                    rALUb<=32'b0;
                    rRt<=32'b0;
                end
                default: begin end
            endcase
            rHI<=hi_w?regfile_Rd:rHI;
            rLO<=lo_w?(hi_w?Rd_out_for_LO:regfile_Rd):rLO;
        end
    end

    always @(negedge clk or posedge reset) begin    //flow
        if (reset) begin
            rIR<=`IR_NON;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: rIR<=if_IR;
                `PARTS_COND_STALL: rIR<=rIR;
                `PARTS_COND_ZERO: rIR<=`IR_NON;
                default: begin end
            endcase
        end
    end
endmodule

module execute(
    input clk,
    input reset,

    input [31:0] alua,
    input [31:0] alub,
    input [31:0] id_Rt,
    input [31:0] id_IR,

    output reg [31:0] rHI,
    output reg [31:0] rLO,
    output reg [31:0] rZ,
    output reg [31:0] rRt,
    output reg [31:0] rIR,

    //control signal
    input [1:0] cond,
    output mult_div_stall,   //cause controller to stall 32 periods
    output cal_finish,
    output overflow_stall,    //next IR_NON
    output [6:0] flow_waddr,

    //for program in
    input cpu_stall
    );

    wire [31:0] cal_lo,cal_hi;
    wire [1:0] cal_sel;
    (* KEEP = "{TRUE|FALSE|SOFT}" *) wire cal_ena;
    wire [31:0] alu_z;
    wire [3:0] alu_sel;

    wire use_overflow;
    (* KEEP = "{TRUE|FALSE|SOFT}" *) assign mult_div_stall=cal_ena;
    assign overflow_stall=use_overflow&alu_overflow;

    controller controller_ex(
        .inst(id_IR),
        .judge_beq(1'b0), //judge BEQ BNE condition to jump
        .judge_bgez(1'b0), //judge BGEZ condition to jump
        /*-----------flow_control-----------*/
        .waddr(flow_waddr),
        /*-----------control-----------*/
        //EX
        .alu_sel(alu_sel),.cal_sel(cal_sel),.cal_ena(cal_ena)/*also for stall*/,.use_overflow(use_overflow)
    );

    calculator calculator_inst(
        .clk(clk), //negedge calculate
        .a(alua), //multiplicand/dividend
        .b(alub), //multiplier/divisor
        .calc(cal_sel),
        .reset(reset),      //high-active,at the beginning of test
        .ena(cal_ena),  //high-active
        .oLO(cal_lo),
        .oHI(cal_hi),
        .sum_finish(cal_finish),

        //for program in
        .cpu_stall(cpu_stall)
    );

    alu alu_inst(
        .a(alua),
        .b(alub),
        .aluc(alu_sel),
        .r(alu_z),
        .overflow(alu_overflow)
    );

    always @(posedge clk or posedge reset) begin    //execute
        if (reset) begin
            rZ<=32'b0;
            rRt<=32'b0;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: begin
                    rZ<=alu_z;
                    rRt<=id_Rt;
                end
                `PARTS_COND_STALL: begin
                    rZ<=rZ;
                    rRt<=rRt;
                end
                `PARTS_COND_ZERO: begin
                    rZ<=32'b0;
                    rRt<=32'b0;
                end
                default: begin end
            endcase
        end
    end

    always @(negedge clk or posedge reset) begin    //flow
        if (reset) begin
            rHI<=32'b0;
            rLO<=32'b0;
            rIR<=`IR_NON;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: begin
                    rHI<=cal_hi;
                    rLO<=cal_lo;
                    rIR<=id_IR;
                end
                `PARTS_COND_STALL: begin
                    rHI<=rHI;
                    rLO<=rLO;
                    rIR<=rIR;
                end
                `PARTS_COND_ZERO: begin
                    rHI<=32'b0;
                    rLO<=32'b0;
                    rIR<=`IR_NON;
                end
                default: begin end
            endcase
        end
    end
endmodule

```

```
module memory_access(
    input clk,
    input reset,

    input [31:0] ex_HI,
    input [31:0] ex_LO,
    input [31:0] ex_Z,
    input [31:0] ex_Rt,
    input [31:0] ex_IR,

    output reg [31:0] rHI,
    output reg [31:0] rLO,
    output reg [31:0] rZ,
    output reg [31:0] rMEM,
    output reg [31:0] rIR,

    //control signal
    input [1:0] cond,
    output [6:0] flow_waddr,
    output dmem_r,
    output use_mul
    );

    assign use_mul=(ex_IR[31:26]==6'b011100&&ex_IR[5:0]==6'b000010);
    wire dmem_w;wire [1:0] dmem_width;wire [31:0] dmem_out,ext_out;wire [2:0] mem_sel;

    controller controller_me(
        .inst(ex_IR),
        .judge_beq(1'b0), //judge BEQ BNE condition to jump
        .judge_bgez(1'b0), //judge BGEZ condition to jump
        /*-----------flow_control-----------*/
        .waddr(flow_waddr),.dmem_r(dmem_r),
        /*-----------control-----------*/
        //ME
        .dmem_w(dmem_w),.dmem_width(dmem_width),.mem_sel(mem_sel)
    );

    ram dmem_inst(
        .clk(clk),  //posedge read/write-active
        .wena(dmem_w), //high:write low:read
        .width(dmem_width),  //0:word,1:hword,2:byte
        .addr(ex_Z-32'h10010000),
        .data_in(ex_Rt),   //use zero side
        .data_out(dmem_out)   //use zero side
    );

    ext ext_me(
        .ext_switch(mem_sel),
        .iData_len5(),
        .iData_len8(dmem_out[7:0]),
        .iData_len16(dmem_out[15:0]),
        .iData_len32(dmem_out),
        .oData(ext_out)
    );

    always @(posedge clk or posedge reset) begin    //execute
        if (reset) begin
```

```
            rHI<=32'b0;
            rLO<=32'b0;
            rZ<=32'b0;
            rMEM<=32'b0;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: begin
                    rHI<=ex_HI;
                    rLO<=ex_LO;
                    rZ<=ex_Z;
                    rMEM<=ext_out;
                end
                `PARTS_COND_STALL: begin
                    rHI<=rHI;
                    rLO<=rLO;
                    rZ<=rZ;
                    rMEM<=rMEM;
                end
                `PARTS_COND_ZERO: begin
                    rHI<=32'b0;
                    rLO<=32'b0;
                    rZ<=32'b0;
                    rMEM<=32'b0;
                end
                default: begin end
            endcase
        end
    end

    always @(negedge clk or posedge reset) begin    //flow
        if (reset) begin
            rIR<=`IR_NON;
        end else begin
            case (cond)
                `PARTS_COND_FLOW: rIR<=ex_IR;
                `PARTS_COND_STALL: rIR<=rIR;
                `PARTS_COND_ZERO: rIR<=`IR_NON;
                default: begin end
            endcase
        end
    end
endmodule

module write_back(
    input clk,
    input reset,

    input [31:0] me_HI,
    input [31:0] me_LO,
    input [31:0] me_Z,
    input [31:0] me_MEM,
    input [31:0] me_IR,

    output [4:0] mux_Rdc_out,
    output [31:0] mux_Rd_out,
    output [31:0] Rd_out_for_LO,

    //control signal
```

```verilog
    input [1:0] cond,
    output hi_w,
    output lo_w,
    output cp0_w,
    output regfile_w,
    output [6:0] flow_waddr,
    output dmem_r,
    output use_mul
    );

    assign use_mul=(me_IR[31:26]==6'b011100&&me_IR[5:0]==6'b000010);
    assign Rd_out_for_LO=me_LO;

    wire [1:0] mux_Rdc_sel,mux_Rd_sel;
    wire hi_w_in,lo_w_in,cp0_w_in,regfile_w_in;
    wire write_ena=(cond==`PARTS_COND_FLOW);
    assign hi_w=write_ena&hi_w_in;
    assign lo_w=write_ena&lo_w_in;
    assign cp0_w=write_ena&cp0_w_in;
    assign regfile_w=write_ena®file_w_in;

    controller controller_wb(
        .inst(me_IR),
        .judge_beq(1'b0), //judge BEQ BNE condition to jump
        .judge_bgez(1'b0), //judge BGEZ condition to jump

        /*-----------flow_control-----------*/
        .waddr(flow_waddr),.dmem_r(dmem_r),
        /*-----------control-----------*/
        //WB
        .mux_Rdc_sel(mux_Rdc_sel),.mux_Rd_sel(mux_Rd_sel),
        .hi_w(hi_w_in),.lo_w(lo_w_in),.regfile_w(regfile_w_in),.cp0_w(cp0_w_in)
    );

    mux_len5_sel4 mux_Rdc(
        .sel(mux_Rdc_sel),
        .iData_1(me_IR[15:11]),
        .iData_2(me_IR[20:16]),
        .iData_3(5'd31),
        .oData(mux_Rdc_out)
    );
    mux_len32_sel4 mux_Rd(
        .sel(mux_Rd_sel),
        .iData_1(me_Z),
        .iData_2(me_MEM),
        .iData_3(me_HI),
        .iData_4(me_LO),
        .oData(mux_Rd_out)
    );
endmodule
```

16) ram.v

```verilog
`include "define.vh"

module ram(
    input clk,  //posedge read/write-active
    input wena, //high:write low:read

    input [1:0] width,  //0:word,1:hword,2:byte
    input [31:0] addr,
    input [31:0] data_in,   //use zero side
    output reg [31:0] data_out   //use zero side
    );

    reg [31:0] memory [0:4095]; //16KB
    wire [31:0] ram_addr;
    assign ram_addr=addr>>2;

    always @(posedge clk) begin
        if(wena) begin
            case (width)
                `RAM_WIDTH_32: memory[ram_addr]<=data_in;   //addr[1:0]=00
                `RAM_WIDTH_16: begin     //addr[0]=0
                    if(addr[1]==1'b0)
                        memory[ram_addr][15:0]<=data_in[15:0];
                    else
                        memory[ram_addr][31:16]<=data_in[15:0];
                end
                `RAM_WIDTH_8: begin
                    case (addr[1:0])
                        2'b00: memory[ram_addr][7:0]<=data_in[7:0];
                        2'b01: memory[ram_addr][15:8]<=data_in[7:0];
                        2'b10: memory[ram_addr][23:16]<=data_in[7:0];
                        2'b11: memory[ram_addr][31:24]<=data_in[7:0];
                        default: begin end
                    endcase
                end
                default: begin end
            endcase
        end
    end

    always @(*) begin
        case (width)
            `RAM_WIDTH_32: data_out[31:0]<=memory[ram_addr];   //addr[1:0]=00
            `RAM_WIDTH_16: begin      //addr[0]=0
                if(addr[1]==1'b0)
                    data_out<={16'b0,memory[ram_addr][15:0]};
                else
                    data_out<={16'b0,memory[ram_addr][31:16]};
            end
            `RAM_WIDTH_8: begin
                case (addr[1:0])
                    2'b00: data_out<={24'b0,memory[ram_addr][7:0]};
                    2'b01: data_out<={24'b0,memory[ram_addr][15:8]};
                    2'b10: data_out<={24'b0,memory[ram_addr][23:16]};
                    2'b11: data_out<={24'b0,memory[ram_addr][31:24]};
                    default: begin end
                endcase
            end
            default: data_out<=32'b0;
        endcase
    end

endmodule
```

17) seg7x16.v

```
module seg7x16(
    input clk,
  input reset,
  input cs,
  input [31:0] i_data,
  output [7:0] o_seg,
  output [7:0] o_sel
    );

    reg [12:0] cnt;
   always @ (posedge clk, posedge reset)
      if (reset)
        cnt <= 0;
      else
        cnt <= cnt + 1'b1;

    wire seg7_clk = cnt[12];

   reg [2:0] seg7_addr;

   always @ (posedge seg7_clk, posedge reset)
     if(reset)
              seg7_addr <= 0;
            else
              seg7_addr <= seg7_addr + 1'b1;

   reg [7:0] o_sel_r;

   always @ (*)
     case(seg7_addr)
              7 : o_sel_r = 8'b01111111;
              6 : o_sel_r = 8'b10111111;
              5 : o_sel_r = 8'b11011111;
              4 : o_sel_r = 8'b11101111;
              3 : o_sel_r = 8'b11110111;
              2 : o_sel_r = 8'b11111011;
              1 : o_sel_r = 8'b11111101;
              0 : o_sel_r = 8'b11111110;
            endcase

   reg [31:0] i_data_store;
   always @ (posedge clk, posedge reset)
     if(reset)
              i_data_store <= 0;
            else if(cs)
              i_data_store <= i_data;

   reg [7:0] seg_data_r;
   always @ (*)
     case(seg7_addr)
              0 : seg_data_r = i_data_store[3:0];
              1 : seg_data_r = i_data_store[7:4];
              2 : seg_data_r = i_data_store[11:8];
              3 : seg_data_r = i_data_store[15:12];
              4 : seg_data_r = i_data_store[19:16];
              5 : seg_data_r = i_data_store[23:20];
              6 : seg_data_r = i_data_store[27:24];
              7 : seg_data_r = i_data_store[31:28];
            endcase

   reg [7:0] o_seg_r;
   always @ (posedge clk, posedge reset)
     if(reset)
              o_seg_r <= 8'hff;
            else
              case(seg_data_r)
                4'h0 : o_seg_r <= 8'hC0;
         4'h1 : o_seg_r <= 8'hF9;
         4'h2 : o_seg_r <= 8'hA4;
         4'h3 : o_seg_r <= 8'hB0;
         4'h4 : o_seg_r <= 8'h99;
         4'h5 : o_seg_r <= 8'h92;
         4'h6 : o_seg_r <= 8'h82;
         4'h7 : o_seg_r <= 8'hF8;
         4'h8 : o_seg_r <= 8'h80;
         4'h9 : o_seg_r <= 8'h90;
         4'hA : o_seg_r <= 8'h88;
         4'hB : o_seg_r <= 8'h83;
         4'hC : o_seg_r <= 8'hC6;
         4'hD : o_seg_r <= 8'hA1;
         4'hE : o_seg_r <= 8'h86;
         4'hF : o_seg_r <= 8'h8E;
              endcase

   assign o_sel = o_sel_r;
   assign o_seg = o_seg_r;

endmodule
```

18) seg7_top.v

```
`define CLK_PERIOD 20

module seg7_top(
    input clk_in,
  input reset,
  output [7:0] o_seg,
  output [7:0] o_sel,
    input cpu_stall,
    input switch_pc_d

    // output [31:0] test_pc,
    // output [31:0] test_ir
    );

    (* KEEP = "{TRUE|FALSE|SOFT}" *) wire [31:0] test_pc,test_ir;

    wire clk_cpu, clk_seg7,clk_100mhz;
    assign clk_100mhz=clk_in;

    reg [`CLK_PERIOD-1:0] counter;
    wire [31:0] seg7_idata;
    assign clk_cpu=counter[`CLK_PERIOD-1];
    assign clk_seg7=counter[0];

    always @(posedge clk_100mhz or posedge reset) begin
        if(reset) begin
            counter<=0;
        end
        else begin
            counter<=counter+1'b1;
        end
    end

    wire [31:0] reg_16;
    assign seg7_idata=switch_pc_d?test_pc:reg_16;

    seg7x16 seg7x16_inst(
        .clk(clk_seg7),
      .reset(reset),
      .cs(1'b1),
      .i_data(seg7_idata),
      .o_seg(o_seg),
      .o_sel(o_sel)
    );

    top_parts cpu_inst(
        .clk(clk_cpu),  //posedge write-active
        .reset(reset),    //active-high asynchronous
        .top_pc(test_pc),
        .top_ir(test_ir),

        //for program out
        .reg_16(reg_16),
        .cpu_stall(cpu_stall)
    );
endmodule
```

3、 动态流水线的仿真程序

1) cpu_simulation_tb.v

```
module cpu_tb();
    reg clk,reset;
    wire [31:0] inst,pc;
    reg [31:0] pc_history[0:4],inst_history[0:4];
    wire [31:0] reg_16;
    reg cpu_stall;

    top_parts uut(
        clk,  //posedge write-active
        reset,    //active-high asynchronous
        pc,
        inst,

        reg_16,
        cpu_stall
    );


    integer file_output;
    integer counter=0;

    initial begin
```

```verilog
    file_output=$fopen("output.txt");
    //$readmemh("C:/DigitalLogic/DLProject/project_51/test.hex",cpu_tb.uut.mem.memory);
    pc_history[0]=32'h0000_0000;
    clk=0;
    reset=1;
    cpu_stall=0;
    #275;
    reset=0;
  end

  always begin
    #4;
    clk=~clk;
    #1;
    if(clk==1'b0&&reset==0) begin
      if(counter==5000/*||(inst_history[3]===32'h00000000&&pc_history[3]!=32'h00400000)*/) begin
        $fclose(file_output);
        $finish;
      end
      else if(pc_history[0]!=pc) begin
        pc_history[4]=pc_history[3];
        pc_history[3]=pc_history[2];
        pc_history[2]=pc_history[1];
        pc_history[1]=pc_history[0];
        pc_history[0]=pc;

        inst_history[4]=inst_history[3];
        inst_history[3]=inst_history[2];
        inst_history[2]=inst_history[1];
        inst_history[1]=inst_history[0];
        inst_history[0]=inst;

        if (pc_history[4]!=32'h0000_0000) begin
          counter=counter+1;
          $fdisplay(file_output,"pc: %h",pc_history[4]);
          $fdisplay(file_output,"instr: %h",inst_history[4]);
          $fdisplay(file_output,"regfile0: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[0]);
          $fdisplay(file_output,"regfile1: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[1]);
          $fdisplay(file_output,"regfile2: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[2]);
          $fdisplay(file_output,"regfile3: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[3]);
          $fdisplay(file_output,"regfile4: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[4]);
          $fdisplay(file_output,"regfile5: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[5]);
          $fdisplay(file_output,"regfile6: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[6]);
          $fdisplay(file_output,"regfile7: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[7]);
          $fdisplay(file_output,"regfile8: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[8]);
```

```
                    $fdisplay(file_output,"regfile9: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[9]);
                    $fdisplay(file_output,"regfile10: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[10]);
                    $fdisplay(file_output,"regfile11: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[11]);
                    $fdisplay(file_output,"regfile12: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[12]);
                    $fdisplay(file_output,"regfile13: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[13]);
                    $fdisplay(file_output,"regfile14: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[14]);
                    $fdisplay(file_output,"regfile15: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[15]);
                    $fdisplay(file_output,"regfile16: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[16]);
                    $fdisplay(file_output,"regfile17: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[17]);
                    $fdisplay(file_output,"regfile18: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[18]);
                    $fdisplay(file_output,"regfile19: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[19]);
                    $fdisplay(file_output,"regfile20: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[20]);
                    $fdisplay(file_output,"regfile21: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[21]);
                    $fdisplay(file_output,"regfile22: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[22]);
                    $fdisplay(file_output,"regfile23: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[23]);
                    $fdisplay(file_output,"regfile24: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[24]);
                    $fdisplay(file_output,"regfile25: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[25]);
                    $fdisplay(file_output,"regfile26: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[26]);
                    $fdisplay(file_output,"regfile27: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[27]);
                    $fdisplay(file_output,"regfile28: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[28]);
                    $fdisplay(file_output,"regfile29: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[29]);
                    $fdisplay(file_output,"regfile30: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[30]);
                    $fdisplay(file_output,"regfile31: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[31]);
                end
            end
        end
    end
endmodule
```

2) cpu_synthesis_implementation_tb.v

```
module cpu_tb();
    reg clk,reset;
    wire [7:0] o_seg,o_sel;
    reg cpu_stall,switch_pc_d;
    seg7_top top_inst(
        clk,
      reset,
      o_seg,
      o_sel,
        cpu_stall,
        switch_pc_d
    );

    initial begin
        clk=0;
        reset=1;
        cpu_stall=0;
        switch_pc_d=0;
        #1000;
        reset=0;
    end

    always begin
        #5;
        clk=~clk;
    end
endmodule
```