计算机系统结构课程实验
总结报告

实验题目：静态流水线设计与性能分析

学号：1951443
姓名：罗劲桐
指导教师：陆有军

日期：2021.11.21

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、 静态流水线的总体结构

此次实验实现的 CPU 为 54 条 MIPS 静态流水线 CPU，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。

静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。

2、 静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）

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。上升沿乘除法输出写 HI、LO 寄存器，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。

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。

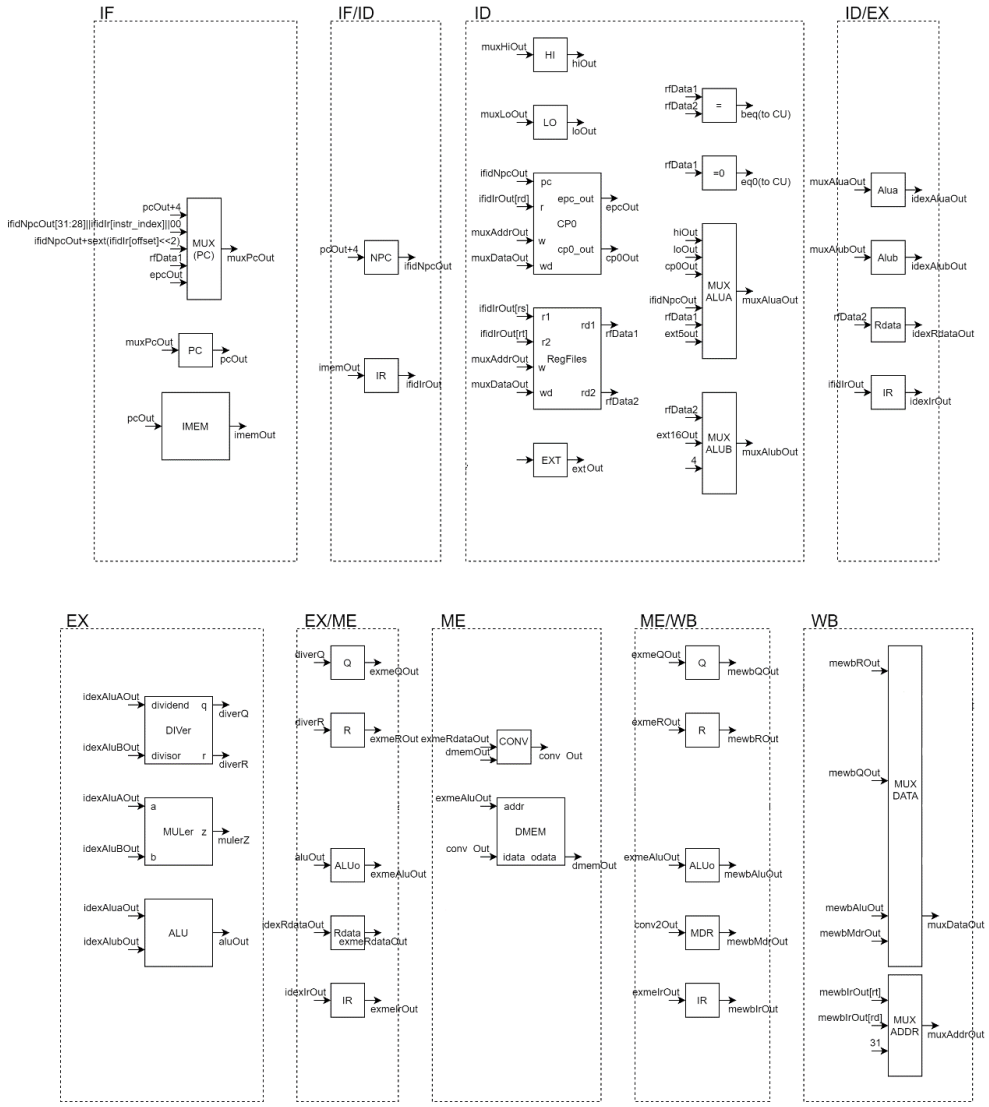
ME 段执行存储交互操作： 上升沿使用 Rt 寄存器写到内存指定单元，或内存读出数据写道 MEM 流水寄存器，HI、LO、Z 寄存器内容写到下一段流水寄存器。组合逻辑生成数据有符号、无符号扩展结果。

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。

3、 静态流水线的总体结构图



4、 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

5、 七段数码管的总体结构

将寄存器#12，#13（表示摔的总次数、摔的总鸡蛋数）低位拼接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、 静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 test_egg program out
    output [31:0] reg_12,       //#12 寄存器
    output [31:0] reg_13,       //#13 寄存器
    output [31:0] reg_14        //#14 寄存器
);
```

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

        //for test_egg program out
        output [31:0] reg_12,         //#12 寄存器
        output [31:0] reg_13,         //#13 寄存器
        output [31:0] reg_14          //#14 寄存器
    );
```

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
);
```

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
);
```

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
    );
```

2、 静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
    );
```

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,                                          //流水控制条件，读、写地址

        /*-----------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、 静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
```

```
);
```

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
    );
```

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
    );
```

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
    );
```

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
    );
```

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
);
```

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
    output egg_break            //最后一次是否摔碎输出
);
```

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

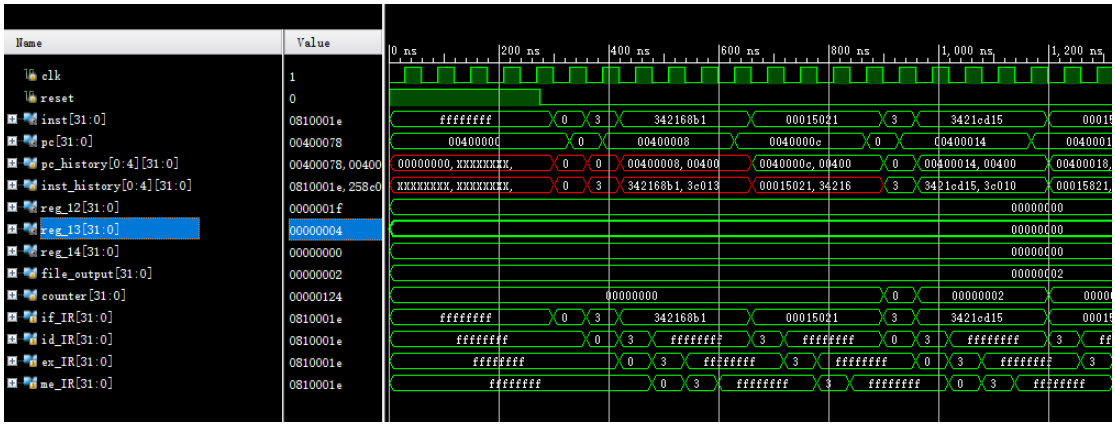
1、 测试样例说明

仿真测试的程序位摔鸡蛋模型程序，默认设置比萨塔层数 total_level= 987654321，耐摔值 break_level= 123456789。对应的结果为摔的次数 drop_cnt=31，摔破的鸡蛋数 drop_egg=4，最后一次的结果 drop_final=0。

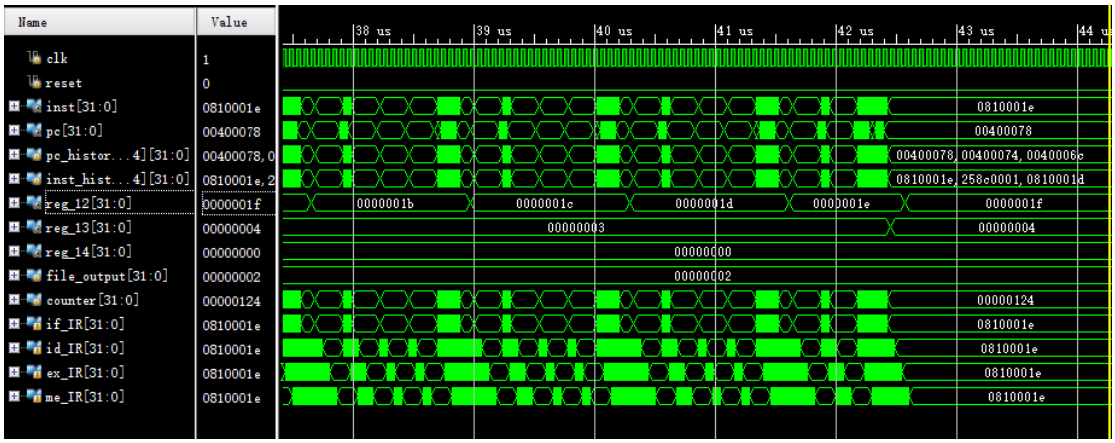
前仿真过程采用波形图输出和寄存器输出到文件的方式查看结果。后仿真无法访问寄存器，因此只采用波形图形式。

2、 静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 reg_12=0000001f（drop_cnt），reg_13=00000004（drop_egg），reg_14=0（egg_break）。结果正确。

2) 寄存器结果输出

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

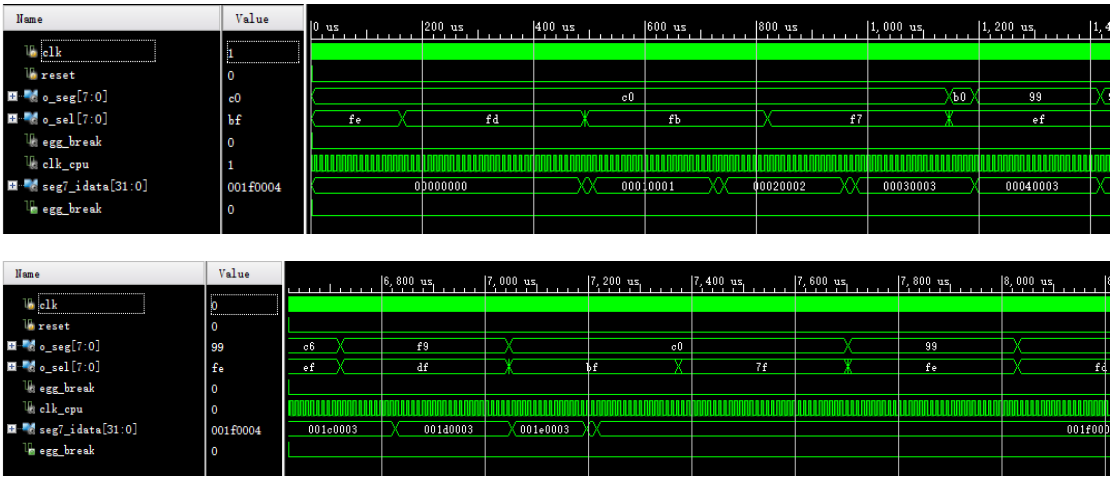
```
regfile31: 00000000
pc: 00400004
instr: 3c013ade
regfile0: 00000000
regfile1: 3ade68b1
regfile2: 00000000
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: 00000000
regfile11: 00000000
regfile12: 00000000
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
```

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

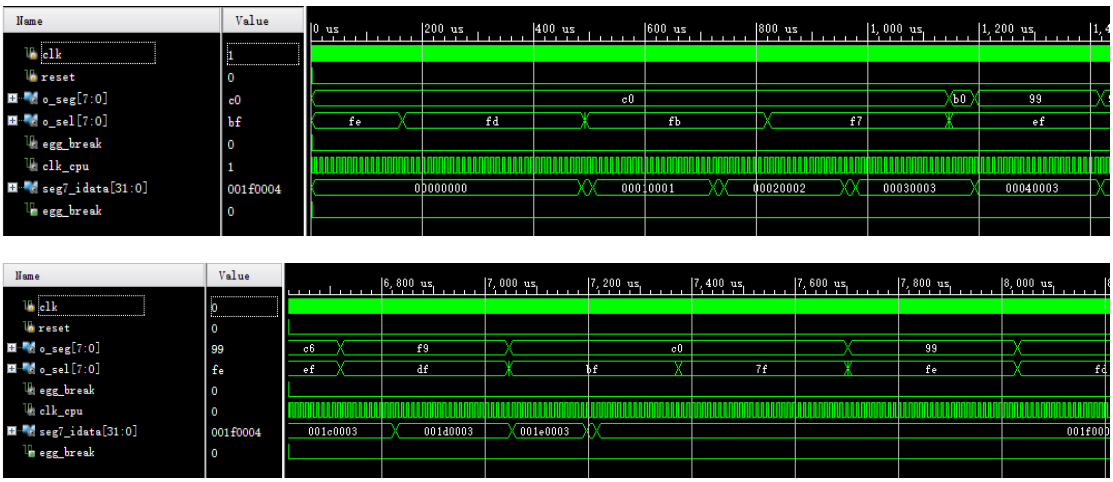
```
pc: 00400054
instr: 258c0001
regfile0: 00000000
regfile1: 075bcd08
regfile2: 075bcd07
regfile3: 075bcd16
regfile4: 075bcd07
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: 3ade68b1
regfile11: 075bcd15
regfile12: 0000001a
regfile13: 00000003
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
```

3、 静态流水线的后仿真过程（后综合仿真）



最终结果 001f0004（drop_cnt 和 drop_egg），egg_break=0。结果正确。

4、 静态流水线的后仿真过程（后实现仿真）



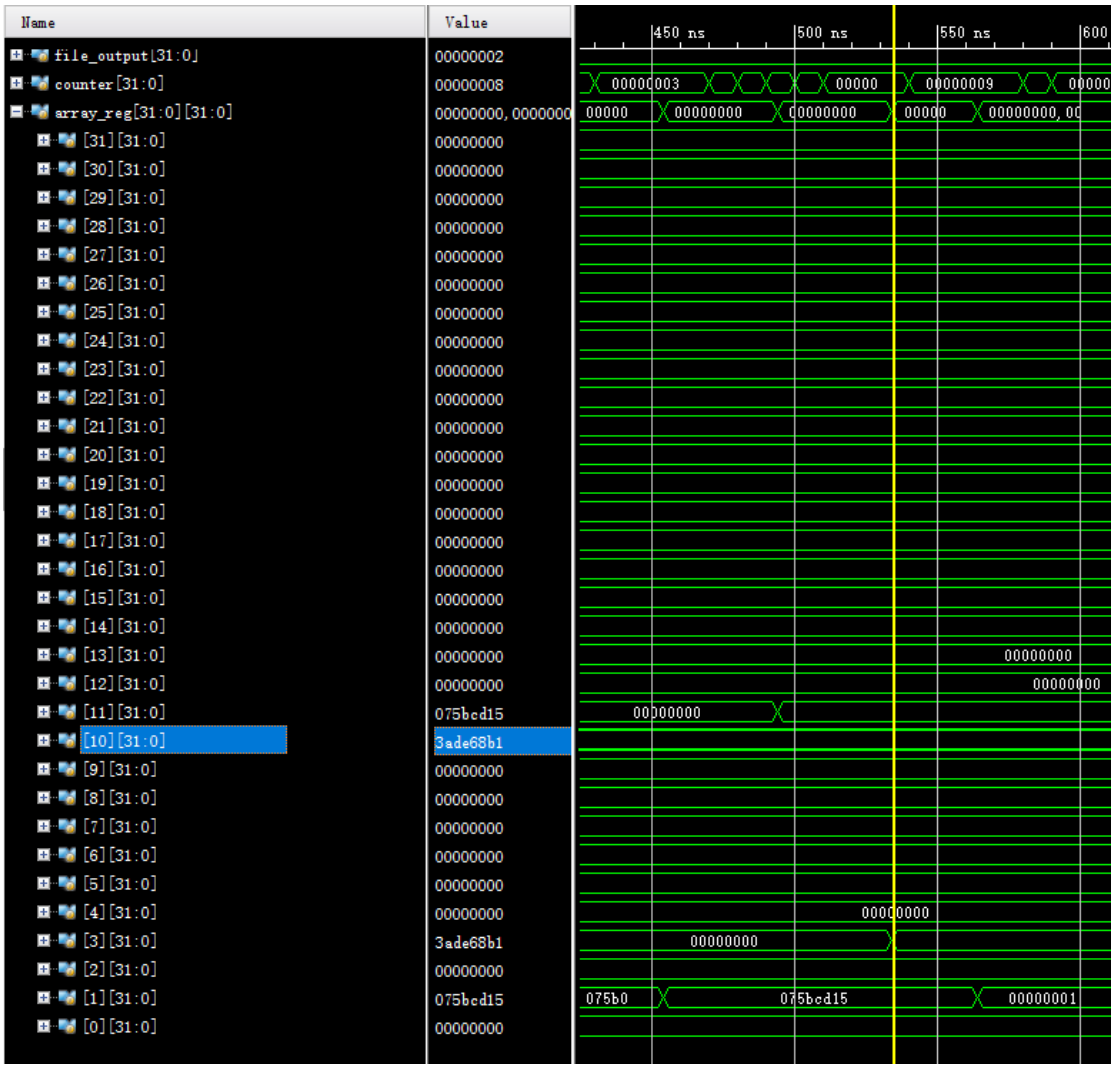
最终结果 001f0004（drop_cnt 和 drop_egg），egg_break=0。结果正确。

五、实验仿真的波形图及某时刻寄存器值的物理意义

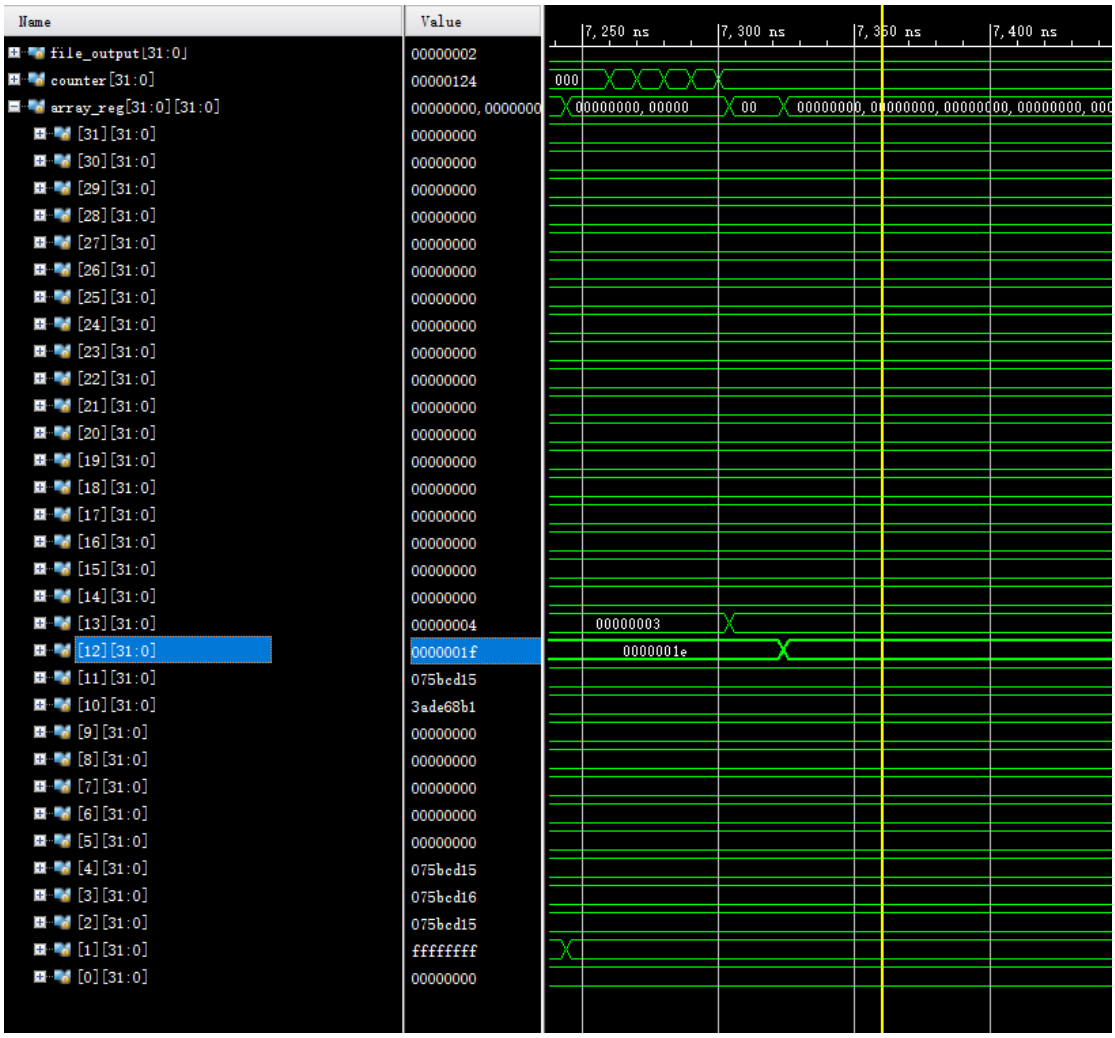
1、 静态流水线的波形图及某时刻寄存器值的物理意义

程序中用到的所有寄存器分为三部分。输入输出数据：$10 比萨塔总层数 total_level， $11 鸡蛋耐摔值 break_level， $12 摔的次数 drop_cnt， $13 摔碎的鸡蛋数 drop_egg， $14 最后鸡蛋是否摔碎 drop_final（drop_final 为 0 表示最后一个鸡蛋摔碎了）。变量：$2 二分法中逼近的下限 low，$3 二分法中逼近的上限 high，$4 二分查找时上下限中间值 center。中间结果寄存器：$1 存储比较、跳转指令的临时变量。

程序开始时，初始化$10、$11 寄存器，分别赋值为比萨塔总层数 987654321（0x3ade68b1）、鸡蛋耐摔值 123456789（0x075bcd15），如下图。



运行结束时，七段数码管显示$12、$13 数值，得到摔的次数 31（0x1f），摔碎的鸡蛋数 4（0x4）。灯显示$14 数值，最后鸡蛋是否摔碎 0（表示摔碎）。同时$2、$3 寄存器距离小于 1，达到二分结束条件退出循环。如下图。



六、实验验算数学模型及算法程序

1) 算法模型

利用二分法查找耐摔值对应的位置，并在范围缩小到两层时停止。此时选择更高的一层测试摔鸡蛋，得出最终结果。

其中 drop_cnt 表示摔的次数，drop_egg 表示摔碎的鸡蛋数，drop_final 为 0 表示最后一个鸡蛋摔碎了。

2) 算法 C 语言程序

```
1.  int main()
2.  {
3.    int total_level = 987654321, break_level = 123456789;
4.    int drop_cnt = 0, drop_egg = 0, drop_final;
5.
6.    int low = 0, high = total_level, center;
7.    while (1)
8.    {
9.              if (low +1 == high){
10.                       if (high > break_level)
11.                       {
```

```
                                drop_final = 0;
                                drop_egg++;
                        }
                        else
                                drop_final = 1;

                        drop_cnt++;
                        break;
                }
                center = (low + high) / 2;
                if (center > break_level) {
                        high = center;
                        drop_egg++;
                }
                else
                        low = center;

                drop_cnt++;
        }
        return 0;
}
```

3) 算法 54 条指令 MIPS 程序

摔鸡蛋的次数，摔破的次数，最后一次摔的结果分别存储在$12，$13，$14寄存器中。

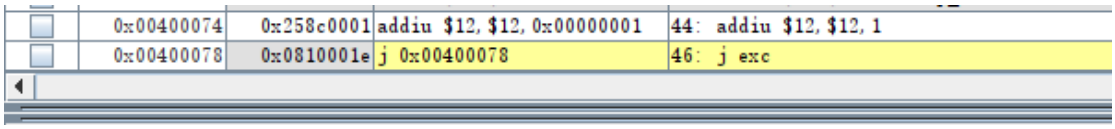
```
.text
sll $0,$0,0

addiu $10,$0,987654321          #total_level
addiu $11,$0,123456789          #break_level
addiu $12,$0,0          #drop_cnt
addiu $13,$0,0          #drop_egg
#addiu $14,$0,0         #drop_final

addiu $2,$0,0           #low
addu $3,$0,$10          #high
#addu $4,$0,0           #center
loop:
addiu $1,$2,1
beq $1,$3,end_loop

#drop egg
addu $1,$2,$3
srl $4,$1,1             #center
sub $1,$11,$4
bgez $1,no_break

#break
addu $3,$0,$4
addiu $13,$13,1
j end_break
no_break:
addu $2,$0,$4
end_break:
addiu $12,$12,1
j loop
```

```
32.
33. end_loop:
34. sub $1,$11,$3
35. bgez $1,final_no_break
36.
37. #final_break
38. addiu $14,$0,0        #drop_final
39. addiu $13,$13,1
40. j final_end
41. final_no_break:
42. addiu $14,$0,1        #drop_final
43. final_end:
44. addiu $12,$12,1
45. exc:        #jump_out
46. j exc
```

4) 验证程序结果

C 程序输出：

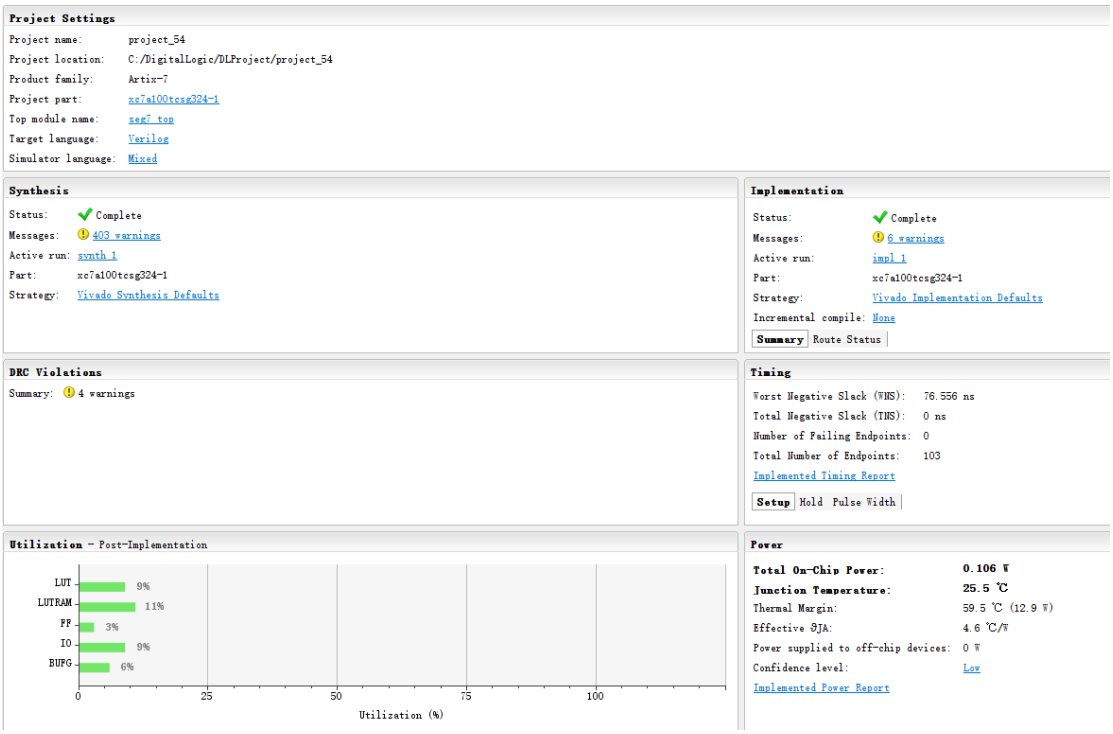
MIPS 程序结束位置：



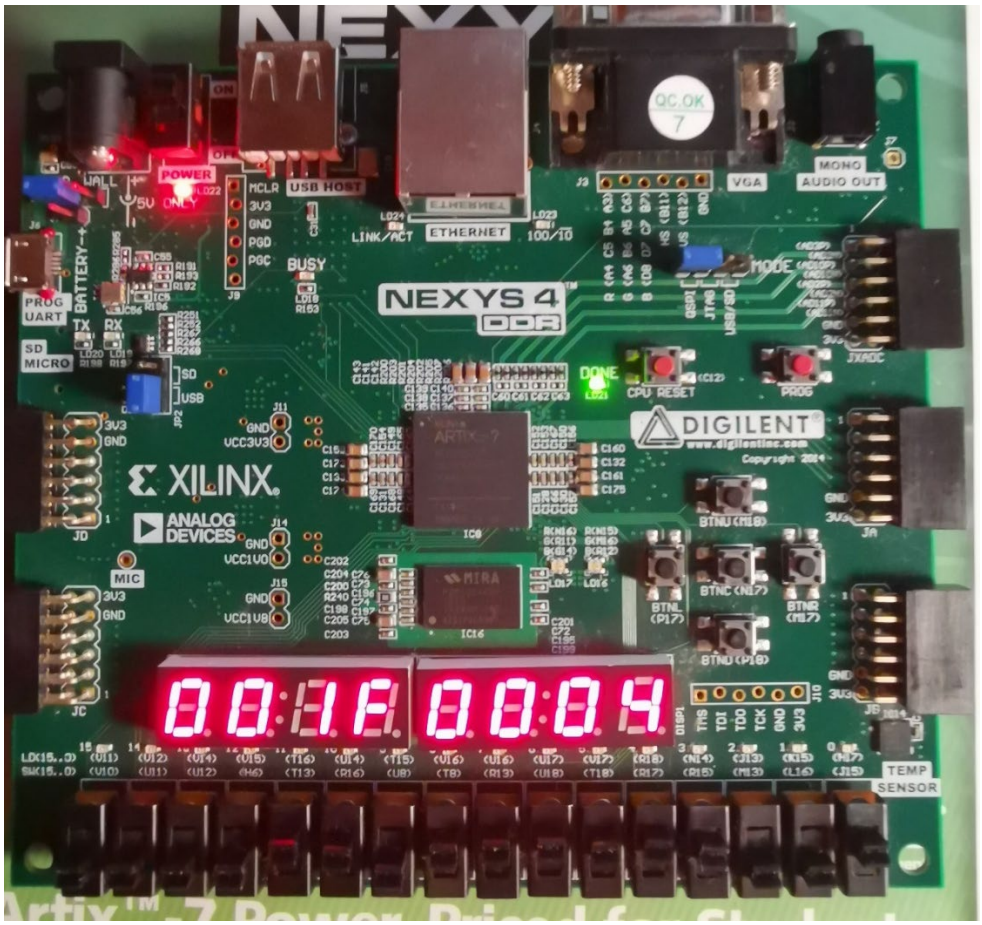
MIPS 程序输出：

七、实验验算程序下板测试过程与实现

1) 综合、实现、下板的运行结果



2) 下板结果



最终结果 001f0004（drop_cnt 和 drop_egg），左下角 V11 灯熄灭（egg_break=0）。结果正确。

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、 静态流水线的性能指标定性分析

吞吐率$TP=n/T_m$=295/7.01us =4.2083*10^7(s^{-1})
加速比$S=T_0/T_m$=295*5*10ns/7.01us=2.104
效率$E=n$个任务占用的时空区/m个功能段的总的时空区
=295*5*10ns/7.01us*5=0.4208

2、 冲突分析

静态流水线只会产生先写后读冲突。其中 ID 段取数值，WB 段写回数值，因此要保证 ID 段取值时不会与 EX、ME、WB 段指令产生先写后读冲突。

这里使用 raddr 和 waddr 组合逻辑来比较指令是否冲突，定义如下：

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

使用一个 7 位数据表示使用到的寄存器，对于使用通用寄存器和 CP0 寄存器的情况，前两位设置为 2’b00、2’b01，并在后五位填入对应的地址，使用 HI、LO 和不使用的情况对应后四项定义。判别逻辑如下。

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

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

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

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 静态流水线 CPU，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。

除了原先设计的 ALU 算术单元、乘除法器、CP0 协处理器、REGFILES 通用寄存器堆、EXT 数据扩展模块，MUX 多路选择器、七段数码管的设计之外，新设计了静态流水线的五段流水结构模块、组合逻辑控制器和流水控制器，同时修改了乘除法器、七段数码管的信号输入输出以适配流水线 CPU 的时序逻辑。

另外，在书中介绍的五段流水线的基础上，我做出了部分修改，使转移指令的判断和地址生成提前到 ID 段进行，这样就不需要在预测失败时排空流水线，提高了 CPU 吞吐率、效率。

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。

在本次 CPU 设计实验中，我学习到流水线 CPU 的流水处理方式。与多周期 CPU 不同，流水线 CPU 各段可能有暂停、排空等操作，同时要侦测各种流水条件来控制各流水段的运行。这要求运行时同时考虑多条指令运行的兼容性，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 C 代码：test_prog.c
验算程序 MIPS 汇编代码：test_prog.asm
验算程序 COE 汇编结果：test_prog.coe

2、 静态流水线的设计程序

1) define.vh

```
1.  //----------Instruction----------
2.
3.  `define PARTS_COND_FLOW 2'b00
4.  `define PARTS_COND_STALL 2'b01
5.  `define PARTS_COND_ZERO 2'b10
6.  `define PC_ADDR_INIT 32'h00400000
7.  `define IR_NON 32'hffffffff
```

```
8.  `define IR_NON_31_26 6'b111111
9.
10. //----------Operation----------
11.
12. `define CAL_MULTU 2'b00
13. `define CAL_MULT 2'b01
14. `define CAL_DIVU 2'b10
15. `define CAL_DIV 2'b11
16.
17. //----------ALU----------
18.
19. `define ALU_ADDU 4'b0000
20. `define ALU_ADD 4'b0010
21. `define ALU_SUBU 4'b0001
22. `define ALU_SUB 4'b0011
23. `define ALU_AND 4'b0100
24. `define ALU_OR 4'b0101
25. `define ALU_XOR 4'b0110
26. `define ALU_NOR 4'b0111
27. `define ALU_LUI 4'b1000
28. `define ALU_SLT 4'b1011
29. `define ALU_SLTU 4'b1010
30. `define ALU_SRA 4'b1100
31. `define ALU_SLL 4'b1110
32. `define ALU_SLA 4'b1111
33. `define ALU_SRL 4'b1101
34. `define ALU_CLZ 4'b1001
35.
36. //----------CP0----------
37.
38. `define EXC_SYSCALL 4'b1000
39. `define EXC_BREAK 4'b1001
40. `define EXC_TEQ 4'b1101
41.
42. //----------Extend----------
43.
44. `define EXT5_Z 3'b000
45. `define EXT16_SL2_S 3'b001
46. `define EXT16_Z 3'b010
47. `define EXT16_S 3'b011
48. `define EXT8_Z 3'b100
49. `define EXT8_S 3'b101
50. `define EXT32_NON 3'b110
51.
52. //----------Mux----------
53.
54. `define MUX_PC_NPCEXT 3'b000
55. `define MUX_PC_RS 3'b001
56. `define MUX_PC_INTR_ADDR 3'b010
57. `define MUX_PC_EPC 3'b011
58. `define MUX_PC_CONNECT 3'b100
59. `define MUX_PC_NPC 3'b101
60.
61. `define MUX_ALUa_HI 3'b000
62. `define MUX_ALUa_LO 3'b001
63. `define MUX_ALUa_NPC 3'b010
64. `define MUX_ALUa_RS 3'b011
65. `define MUX_ALUa_EXT 3'b100
```

```
66. `define MUX_ALUa_CP0 3'b101
67. `define MUX_ALUa_IMM0 3'b110
68.
69. `define MUX_ALUb_IMM0 2'b00
70. `define MUX_ALUb_RT 2'b01
71. `define MUX_ALUb_EXT 2'b10
72.
73. `define MUX_RDC_IR_15_11 2'b00
74. `define MUX_RDC_IR_20_16 2'b01
75. `define MUX_RDC_IMM31 2'b10
76.
77. `define MUX_RD_Z 2'b00
78. `define MUX_RD_MEM 2'b01
79. `define MUX_RD_HI 2'b10
80. `define MUX_RD_LO 2'b11
81.
82. //----------RAM----------
83.
84. `define RAM_WIDTH_32 2'b00
85. `define RAM_WIDTH_16 2'b01
86. `define RAM_WIDTH_8 2'b10
87.
88. //----------CtrlUnit----------
89.
90. `define FLOW_NORMAL 2'b00
91. `define FLOW_OVERFLOW 2'b01
92. `define FLOW_MULDIV 2'b10
93. `define FLOW_VIOLATE 2'b11
94.
95. //----------Others----------
96.
97. //judge read/write violation
98. `define VIOLATION_REGFILE_HEAD 2'b00
99. `define VIOLATION_CP0REG_HEAD 2'b01
100.  `define VIOLATION_HI 7'b100_0000
101.  `define VIOLATION_LO 7'b100_0001
102.  `define VIOLATION_HILO 7'b100_0010
103.  `define VIOLATION_NON 7'b110_0000
```

2) alu.v

```
1.  `include "define.vh"
2.
3.  module alu(
4.      input [31:0] a,
5.      input [31:0] b,
6.      input [3:0] aluc,
7.      output reg [31:0] r,
8.      output reg zero,
9.      output reg carry,
10.     output reg negative,
11.     output reg overflow
12.     );
13.     reg [1:0] carry2;
14.     reg [31:0] tmp;
15.     always@(*)
16.     begin
17.         case(aluc)
18.             `ALU_ADDU:    //unsigned +
```

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

```
77.             begin
78.                 {carry2[0],tmp}={a[31],a}-{b[31],b};
79.                 zero=(a==b)?1:0;
80.                 negative=carry2[0];
81.                 r={31'b0,negative};
82.             end
83.             `ALU_SLTU:    //unsigned <
84.             begin
85.                 carry=(a<b)?1:0;
86.                 zero=(a==b)?1:0;
87.                 negative=0;
88.                 r={31'b0,carry};
89.             end
90.
91.             `ALU_SRA:    //shiftR arithmetic
92.             begin
93.                 if(a[4:0]>0)
94.                 begin
95.                     carry=b[a[4:0]-1];
96.                     tmp={32{b[31]}};
97.                     r={tmp,b}>>a[4:0];
98.                 end
99.                 else
100.                begin
101.                    carry=b[0];
102.                    r=b;    //fix
103.                end
104.                zero=(0==r)?1:0;
105.                negative=r[31];
106.            end
107.            `ALU_SLL,`ALU_SLA:    //shiftL
108.            begin
109.                if(a[4:0]>0)
110.                    carry=b[32-a[4:0]];
111.                else
112.                begin
113.                    carry=b[31];
114.                    r=b;    //fix
115.                end
116.                r=b<<a[4:0];
117.                zero=(0==r)?1:0;
118.                negative=r[31];
119.            end
120.            `ALU_SRL:    //shiftR logic
121.            begin
122.                if(a[4:0]>0)
123.                    carry=b[a[4:0]-1];
124.                else
125.                begin
126.                    carry=b[0];
127.                    r=b;    //fix
128.                end
129.                r=b>>a[4:0];
130.                zero=(0==r)?1:0;
131.                negative=r[31];
132.            end
133.            `ALU_CLZ:    //count leading zero
134.            begin
```

```
                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 sum_finish
    );

    reg start,busy,finish;
```

```
16.     wire busyDivu,busyDiv,busyMult,busyMultu;
17.     wire [63:0] oMult,oMultu,oDiv,oDivu;
18.     always @(*) begin
19.         case (calc)
20.             `CAL_MULTU: begin
21.                 oLO<=oMultu[31:0];
22.                 oHI<=oMultu[63:32];
23.                 busy<=busyMultu;
24.             end
25.             `CAL_MULT: begin
26.                 oLO<=oMult[31:0];
27.                 oHI<=oMult[63:32];
28.                 busy<=busyMult;
29.             end
30.             `CAL_DIVU: begin
31.                 oLO<=oDivu[31:0];
32.                 oHI<=oDivu[63:32];
33.                 busy<=busyDivu;
34.             end
35.             `CAL_DIV: begin
36.                 oLO<=oDiv[31:0];
37.                 oHI<=oDiv[63:32];
38.                 busy<=busyDiv;
39.             end
40.             default: begin end
41.         endcase
42.     end
43.
44.     assign sum_finish=finish;
45.     always @(negedge ena or negedge busy) begin
46.         if(ena&!busy)
47.             finish<=1;
48.         else if (!ena)
49.             finish<=0;
50.     end
51.     always @(posedge ena or posedge busy) begin
52.         if(ena&!busy)
53.             start<=1;
54.         else if (busy)
55.             start<=0;
56.     end
57.
58.
59.
60.     MULT MULT_inst(
61.         .clk(clk),
62.         .reset(reset),    //active high
63.         .start(start&(calc==`CAL_MULT)),
64.         .a(a), //multiplicand
65.         .b(b), //multiplier
66.         .z(oMult),
67.         .busy(busyMult)
68.     );
69.     MULTU MULTU_inst(
70.         .clk(clk),
71.         .reset(reset),    //active high
72.         .start(start&(calc==`CAL_MULTU)),
73.         .a(a), //multiplicand
```

```
        .b(b), //multiplier
        .z(oMultu),
        .busy(busyMultu)
    );
    DIV DIV_inst(
        .dividend(a),
        .divisor(b),
        .start(start&(calc==`CAL_DIV)),
        .clock(clk),
        .reset(reset),    //active-high
        .q(oDiv[31:0]),
        .r(oDiv[63:32]),
        .busy(busyDiv)
    );
    DIVU DIVU_inst(
        .dividend(a),
        .divisor(b),
        .start(start&(calc==`CAL_DIVU)),
        .clock(clk),
        .reset(reset),    //active-high
        .q(oDivu[31:0]),
        .r(oDivu[63:32]),
        .busy(busyDivu)
    );
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,

    /*-----------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
```

```
29.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
30.                 //IF
31.                 mux_pc_sel<=`MUX_PC_NPC;
32.                 //ID
33.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
34.                 cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
35.                 //EX
36.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
37.                 dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
38.                 //WB
39.                 mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
40.                 hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
41.             end
42.             6'b000000: begin
43.                 case (inst[5:0])
44.                     6'b100000: begin    //add
45.                         //FLOW
46.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
47.                         //IF
48.                         mux_pc_sel<=`MUX_PC_NPC;
49.                         //ID
50.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
51.                         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
52.                         //EX
53.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
54.                         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
55.                         //WB
56.                         mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
57.                         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
58.                     end
59.                     6'b100001: begin    //addu
60.                         //FLOW
61.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
62.                         //IF
63.                         mux_pc_sel<=`MUX_PC_NPC;
64.                         //ID
65.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
66.                         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
67.                         //EX
68.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
69.                         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
70.                         //WB
71.                         mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
72.                         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
73.                     end
74.                     6'b100010: begin    //sub
75.                         //FLOW
```

```
76.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
77.                         //IF
78.                         mux_pc_sel<=`MUX_PC_NPC;
79.                         //ID
80.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
81.                         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
82.                         //EX
83.
    alu_sel<=`ALU_SUB;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
84.                         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
85.                         //WB
86.                         mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
87.                         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
88.                     end
89.                     6'b100011: begin    //subu
90.                         //FLOW
91.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
92.                         //IF
93.                         mux_pc_sel<=`MUX_PC_NPC;
94.                         //ID
95.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
96.                         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
97.                         //EX
98.
    alu_sel<=`ALU_SUBU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
99.                         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
100.                          //WB
101.                          mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
102.                          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
103.                      end
104.                      6'b100100: begin    //and
105.                          //FLOW
106.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
107.                          //IF
108.                          mux_pc_sel<=`MUX_PC_NPC;
109.                          //ID
110.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
111.                          cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
112.                          //EX
113.
    alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
114.                          dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
115.                          //WB
116.                          mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
117.                          hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
118.                      end
119.                      6'b100101: begin    //or
120.                          //FLOW
```

```
121.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
122.                        //IF
123.                        mux_pc_sel<=`MUX_PC_NPC;
124.                        //ID
125.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
126.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
127.                        //EX
128.
    alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
129.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
130.                        //WB
131.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
132.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
133.                    end
134.                    6'b100110: begin    //xor
135.                        //FLOW
136.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
137.                        //IF
138.                        mux_pc_sel<=`MUX_PC_NPC;
139.                        //ID
140.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
141.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
142.                        //EX
143.
    alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
144.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
145.                        //WB
146.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
147.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
148.                    end
149.                    6'b100111: begin    //nor
150.                        //FLOW
151.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
152.                        //IF
153.                        mux_pc_sel<=`MUX_PC_NPC;
154.                        //ID
155.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
156.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
157.                        //EX
158.
    alu_sel<=`ALU_NOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
159.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
160.                        //WB
161.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
162.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
163.                    end
164.                    6'b101010: begin    //slt
165.                        //FLOW
```

```
166.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
167.                        //IF
168.                        mux_pc_sel<=`MUX_PC_NPC;
169.                        //ID
170.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
171.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
172.                        //EX
173.
    alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
174.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
175.                        //WB
176.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
177.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
178.                    end
179.                    6'b101011: begin    //sltu
180.                        //FLOW
181.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
182.                        //IF
183.                        mux_pc_sel<=`MUX_PC_NPC;
184.                        //ID
185.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_S;
186.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
187.                        //EX
188.
    alu_sel<=`ALU_SLTU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
189.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
190.                        //WB
191.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
192.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
193.                    end
194.                    6'b000000: begin    //sll
195.                        //FLOW
196.
    raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
    LATION_REGFILE_HEAD,inst[15:11]};
197.                        //IF
198.                        mux_pc_sel<=`MUX_PC_NPC;
199.                        //ID
200.
    mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
201.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
202.                        //EX
203.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
204.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
205.                        //WB
206.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
207.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
208.                    end
209.                    6'b000010: begin    //srl
210.                        //FLOW
```

```
211.
    raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
    LATION_REGFILE_HEAD,inst[15:11]};
212.                        //IF
213.                        mux_pc_sel<=`MUX_PC_NPC;
214.                        //ID
215.
    mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
216.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
217.                        //EX
218.
    alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
219.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
220.                        //WB
221.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
222.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
223.                    end
224.                    6'b000011: begin    //sra
225.                        //FLOW
226.
    raddr1<=`VIOLATION_NON;raddr2<={`VIOLATION_REGFILE_HEAD,inst[20:16]};waddr<={`VIO
    LATION_REGFILE_HEAD,inst[15:11]};
227.                        //IF
228.                        mux_pc_sel<=`MUX_PC_NPC;
229.                        //ID
230.
    mux_ALUa_sel<=`MUX_ALUa_EXT;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
231.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
232.                        //EX
233.
    alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
234.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
235.                        //WB
236.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
237.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
238.                    end
239.                    6'b000100: begin    //sllv
240.                        //FLOW
241.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
242.                        //IF
243.                        mux_pc_sel<=`MUX_PC_NPC;
244.                        //ID
245.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
246.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
247.                        //EX
248.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
249.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
250.                        //WB
251.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
252.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
253.                    end
254.                    6'b000110: begin    //srlv
255.                        //FLOW
```

```
256.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
257.                        //IF
258.                        mux_pc_sel<=`MUX_PC_NPC;
259.                        //ID
260.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
261.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
262.                        //EX
263.
    alu_sel<=`ALU_SRL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
264.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
265.                        //WB
266.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
267.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
268.                    end
269.                    6'b000111: begin    //srav
270.                        //FLOW
271.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
272.                        //IF
273.                        mux_pc_sel<=`MUX_PC_NPC;
274.                        //ID
275.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
276.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
277.                        //EX
278.
    alu_sel<=`ALU_SRA;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
279.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
280.                        //WB
281.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
282.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
283.                    end
284.                    6'b001000: begin    //jr
285.                        //FLOW
286.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
    ATION_NON;
287.                        //IF
288.                        mux_pc_sel<=`MUX_PC_RS;
289.                        //ID
290.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
291.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
292.                        //EX
293.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
294.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
295.                        //WB
296.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
297.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
298.                    end
299.                    6'b011010: begin    //div
300.                        //FLOW
```

```
301.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_HILO;
302.                        //IF
303.                        mux_pc_sel<=`MUX_PC_NPC;
304.                        //ID
305.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
306.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
307.                        //EX
308.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_DIV;cal_ena<=1'b1;use_overflow<=1'b0;
309.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
310.                        //WB
311.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
312.                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
313.                    end
314.                    6'b011011: begin    //divu
315.                        //FLOW
316.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_HILO;
317.                        //IF
318.                        mux_pc_sel<=`MUX_PC_NPC;
319.                        //ID
320.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
321.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
322.                        //EX
323.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_DIVU;cal_ena<=1'b1;use_overflow<=1'b0;
324.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
325.                        //WB
326.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
327.                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
328.                    end
329.                    6'b011001: begin    //multu
330.                        //FLOW
331.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_HILO;
332.                        //IF
333.                        mux_pc_sel<=`MUX_PC_NPC;
334.                        //ID
335.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
336.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
337.                        //EX
338.
    alu_sel<=`ALU_SLL;cal_sel<=`CAL_MULTU;cal_ena<=1'b1;use_overflow<=1'b0;
339.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
340.                        //WB
341.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_HI;
342.                        hi_w<=1'b1;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
343.                    end
344.                    6'b001001: begin    //jarl
345.                        //FLOW
```

```
346.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[15:11]};
347.                        //IF
348.                        mux_pc_sel<=`MUX_PC_RS;
349.                        //ID
350.
    mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
351.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
352.                        //EX
353.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
354.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
355.                        //WB
356.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
357.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
358.                    end
359.                    6'b001101: begin    //break
360.                        //FLOW
361.
    raddr1<={`VIOLATION_CP0REG_HEAD,5'd12};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_N
    ON;
362.                        //IF
363.                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
364.                        //ID
365.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
366.                        cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
367.                        //EX
368.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
369.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
370.                        //WB
371.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
372.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
373.                    end
374.                    6'b001100: begin    //syscall
375.                        //FLOW
376.
    raddr1<={`VIOLATION_CP0REG_HEAD,5'd12};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_N
    ON;
377.                        //IF
378.                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
379.                        //ID
380.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
381.
    cp0_exception<=1'b1;cp0_eret<=1'b0;cp0_cause<=`EXC_SYSCALL;
382.                        //EX
383.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
384.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
385.                        //WB
386.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
387.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
388.                    end
389.                    6'b010000: begin    //mfhi
390.                        //FLOW
```

```
391.
    raddr1<=`VIOLATION_HI;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst
    [15:11]};
392.                        //IF
393.                        mux_pc_sel<=`MUX_PC_NPC;
394.                        //ID
395.
    mux_ALUa_sel<=`MUX_ALUa_HI;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
396.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
397.                        //EX
398.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
399.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
400.                        //WB
401.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
402.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
403.                    end
404.                    6'b010010: begin    //mflo
405.                        //FLOW
406.
    raddr1<=`VIOLATION_LO;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,inst
    [15:11]};
407.                        //IF
408.                        mux_pc_sel<=`MUX_PC_NPC;
409.                        //ID
410.
    mux_ALUa_sel<=`MUX_ALUa_LO;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
411.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
412.                        //EX
413.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
414.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
415.                        //WB
416.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
417.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
418.                    end
419.                    6'b010001: begin    //mthi
420.                        //FLOW
421.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
    ATION_HI;
422.                        //IF
423.                        mux_pc_sel<=`MUX_PC_NPC;
424.                        //ID
425.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
426.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
427.                        //EX
428.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
429.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
430.                        //WB
431.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
432.                        hi_w<=1'b1;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
433.                    end
434.                    6'b010011: begin    //mtlo
435.                        //FLOW
```

```
436.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
    ATION_LO;
437.                        //IF
438.                        mux_pc_sel<=`MUX_PC_NPC;
439.                        //ID
440.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
441.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
442.                        //EX
443.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
444.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
445.                        //WB
446.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
447.                        hi_w<=1'b0;lo_w<=1'b1;regfile_w<=1'b0;cp0_w<=1'b0;
448.                    end
449.                    6'b110100: begin    //teq
450.                        //FLOW
451.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
452.                        //IF
453.                        mux_pc_sel<=`MUX_PC_INTR_ADDR;
454.                        //ID
455.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
456.
    cp0_exception<=judge_beq;cp0_eret<=1'b0;cp0_cause<=`EXC_TEQ;
457.                        //EX
458.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
459.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
460.                        //WB
461.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
462.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
463.                    end
464.                    default: begin    //do nothing
465.                        //FLOW
466.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
467.                        //IF
468.                        mux_pc_sel<=`MUX_PC_NPC;
469.                        //ID
470.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
471.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
472.                        //EX
473.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
474.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
475.                        //WB
476.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
477.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
478.                    end
479.                endcase
480.            end
481.            6'b001000: begin    //addi
```

```
482.                //FLOW
483.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
484.                //IF
485.                mux_pc_sel<=`MUX_PC_NPC;
486.                //ID
487.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
488.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
489.                //EX
490.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b1;
491.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
492.                //WB
493.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
494.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
495.            end
496.            6'b001001: begin    //addiu
497.                //FLOW
498.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
499.                //IF
500.                mux_pc_sel<=`MUX_PC_NPC;
501.                //ID
502.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
503.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
504.                //EX
505.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
506.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
507.                //WB
508.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
509.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
510.            end
511.            6'b001100: begin    //andi
512.                //FLOW
513.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
514.                //IF
515.                mux_pc_sel<=`MUX_PC_NPC;
516.                //ID
517.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
518.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
519.                //EX
520.
    alu_sel<=`ALU_AND;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
521.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
522.                //WB
523.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
524.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
525.            end
526.            6'b001101: begin    //ori
527.                //FLOW
```

```
528.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
529.                //IF
530.                mux_pc_sel<=`MUX_PC_NPC;
531.                //ID
532.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
533.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
534.                //EX
535.
    alu_sel<=`ALU_OR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
536.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
537.                //WB
538.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
539.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
540.            end
541.            6'b001110: begin    //xori
542.                //FLOW
543.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
544.                //IF
545.                mux_pc_sel<=`MUX_PC_NPC;
546.                //ID
547.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
548.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
549.                //EX
550.
    alu_sel<=`ALU_XOR;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
551.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
552.                //WB
553.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
554.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
555.            end
556.            6'b001111: begin    //lui
557.                //FLOW
558.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<={`VIOLATION_REGFILE_HEAD,ins
    t[20:16]};
559.                //IF
560.                mux_pc_sel<=`MUX_PC_NPC;
561.                //ID
562.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
563.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
564.                //EX
565.
    alu_sel<=`ALU_LUI;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
566.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
567.                //WB
568.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
569.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
570.            end
571.            6'b100011: begin    //lw
572.                //FLOW
```

```
573.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
574.                //IF
575.                mux_pc_sel<=`MUX_PC_NPC;
576.                //ID
577.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
578.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
579.                //EX
580.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
581.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
582.                //WB
583.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
584.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
585.            end
586.            6'b101011: begin    //sw
587.                //FLOW
588.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
589.                //IF
590.                mux_pc_sel<=`MUX_PC_NPC;
591.                //ID
592.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
593.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
594.                //EX
595.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
596.                dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_32;mem_sel<=`EXT32_NON;
597.                //WB
598.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
599.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
600.            end
601.            6'b000100: begin    //beq
602.                //FLOW
603.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
604.                //IF
605.                mux_pc_sel<=judge_beq?`MUX_PC_NPCEXT:`MUX_PC_NPC;
606.                //ID
607.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
608.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
609.                //EX
610.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
611.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
612.                //WB
613.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
614.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
615.            end
616.            6'b000101: begin    //bne
617.                //FLOW
```

```
618.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
619.                //IF
620.                mux_pc_sel<=judge_beq?`MUX_PC_NPC:`MUX_PC_NPCEXT;
621.                //ID
622.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT16_SL2_S;
623.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
624.                //EX
625.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
626.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
627.                //WB
628.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
629.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
630.            end
631.            6'b001010: begin    //slti
632.                //FLOW
633.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
634.                //IF
635.                mux_pc_sel<=`MUX_PC_NPC;
636.                //ID
637.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
638.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
639.                //EX
640.
    alu_sel<=`ALU_SLT;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
641.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
642.                //WB
643.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
644.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
645.            end
646.            6'b001011: begin    //sltiu
647.                //FLOW
648.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
649.                //IF
650.                mux_pc_sel<=`MUX_PC_NPC;
651.                //ID
652.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_Z;
653.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
654.                //EX
655.
    alu_sel<=`ALU_SLTU;cal_sel<=`CAL_MULT;cal_ena<=1'b0;use_overflow<=1'b0;
656.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
657.                //WB
658.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
659.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
660.            end
661.            6'b000010: begin    //j
662.                //FLOW
```

```
663.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
664.                //IF
665.                mux_pc_sel<=`MUX_PC_CONNECT;
666.                //ID
667.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
668.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
669.                //EX
670.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
671.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
672.                //WB
673.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
674.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
675.            end
676.            6'b000011: begin    //jal
677.                //FLOW
678.
    raddr1<={`VIOLATION_REGFILE_HEAD,5'd31};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_
    NON;
679.                //IF
680.                mux_pc_sel<=`MUX_PC_CONNECT;
681.                //ID
682.
    mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
683.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
684.                //EX
685.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
686.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
687.                //WB
688.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
689.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
690.            end
691.            6'b000001: begin    //bgez
692.                //FLOW
693.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<=`VIOL
    ATION_NON;
694.                //IF
695.                mux_pc_sel<=judge_bgez?`MUX_PC_NPCEXT:`MUX_PC_NPC;
696.                //ID
697.
    mux_ALUa_sel<=`MUX_ALUa_NPC;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT16_SL2_S;
698.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
699.                //EX
700.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
701.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
702.                //WB
703.                mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
704.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
705.            end
706.            6'b100100: begin    //lbu
707.                //FLOW
```

```
708.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
709.                //IF
710.                mux_pc_sel<=`MUX_PC_NPC;
711.                //ID
712.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
713.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
714.                //EX
715.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
716.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_Z;
717.                //WB
718.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
719.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
720.            end
721.            6'b100101: begin    //lhu
722.                //FLOW
723.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
724.                //IF
725.                mux_pc_sel<=`MUX_PC_NPC;
726.                //ID
727.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
728.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
729.                //EX
730.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
731.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_Z;
732.                //WB
733.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
734.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
735.            end
736.            6'b100000: begin    //lb
737.                //FLOW
738.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
739.                //IF
740.                mux_pc_sel<=`MUX_PC_NPC;
741.                //ID
742.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
743.                cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
744.                //EX
745.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
746.                dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
747.                //WB
748.                mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
749.                hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
750.            end
751.            6'b100001: begin    //lh
752.                //FLOW
```

```
753.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[20:16]};
754.                 //IF
755.                 mux_pc_sel<=`MUX_PC_NPC;
756.                 //ID
757.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
758.                 cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
759.                 //EX
760.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
761.                 dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
762.                 //WB
763.                 mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
764.                 hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
765.             end
766.             6'b101000: begin    //sb
767.                 //FLOW
768.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
769.                 //IF
770.                 mux_pc_sel<=`MUX_PC_NPC;
771.                 //ID
772.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
773.                 cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
774.                 //EX
775.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
776.                 dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_8;mem_sel<=`EXT8_S;
777.                 //WB
778.                 mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
779.                 hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
780.             end
781.             6'b101001: begin    //sh
782.                 //FLOW
783.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<=`VIOLATION_NON;
784.                 //IF
785.                 mux_pc_sel<=`MUX_PC_NPC;
786.                 //ID
787.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_EXT;ext_sel<=`EXT16_S;
788.                 cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
789.                 //EX
790.
    alu_sel<=`ALU_ADD;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
791.                 dmem_w<=1'b1;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT16_S;
792.                 //WB
793.                 mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_MEM;
794.                 hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
795.             end
796.             6'b010000: begin
797.                 case (inst[25:21])
798.                     5'b10000: begin     //eret
```

```
799.                        //FLOW
800.
    raddr1<={`VIOLATION_CP0REG_HEAD,5'd14};raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_N
    ON;
801.                        //IF
802.                        mux_pc_sel<=`MUX_PC_EPC;
803.                        //ID
804.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
805.                        cp0_exception<=1'b0;cp0_eret<=1'b1;cp0_cause<=`EXC_BREAK;
806.                        //EX
807.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
808.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
809.                        //WB
810.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
811.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
812.                    end
813.                    5'b00000: begin     //mfc0
814.                        //FLOW
815.
    raddr1<={`VIOLATION_CP0REG_HEAD,inst[15:11]};raddr2<=`VIOLATION_NON;waddr<={`VIOL
    ATION_REGFILE_HEAD,inst[20:16]};
816.                        //IF
817.                        mux_pc_sel<=`MUX_PC_NPC;
818.                        //ID
819.
    mux_ALUa_sel<=`MUX_ALUa_CP0;mux_ALUb_sel<=`MUX_ALUb_IMM0;ext_sel<=`EXT5_Z;
820.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
821.                        //EX
822.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
823.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
824.                        //WB
825.                        mux_Rdc_sel<=`MUX_RDC_IR_20_16;mux_Rd_sel<=`MUX_RD_Z;
826.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
827.                    end
828.                    5'b00100: begin     //mtc0
829.                        //FLOW
830.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[20:16]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_CP0REG_HEAD,inst[15:11]};
831.                        //IF
832.                        mux_pc_sel<=`MUX_PC_NPC;
833.                        //ID
834.
    mux_ALUa_sel<=`MUX_ALUa_IMM0;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
835.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
836.                        //EX
837.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
838.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
839.                        //WB
840.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
841.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b1;
842.                    end
843.                    default: begin    //do nothing
844.                        //FLOW
```

```
845.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
846.                        //IF
847.                        mux_pc_sel<=`MUX_PC_NPC;
848.                        //ID
849.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
850.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
851.                        //EX
852.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
853.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
854.                        //WB
855.                        mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
856.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
857.                    end
858.                endcase
859.            end
860.            6'b011100: begin
861.                case (inst[5:0])
862.                    6'b000010: begin    //mul
863.                        //FLOW
864.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<={`VIOLATION_REGFILE_HEAD,in
    st[20:16]};waddr<={`VIOLATION_REGFILE_HEAD,inst[15:11]};
865.                        //IF
866.                        mux_pc_sel<=`MUX_PC_NPC;
867.                        //ID
868.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
869.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
870.                        //EX
871.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULT;cal_ena<=1'b1;use_overflow<=1'b0;
872.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
873.                        //WB
874.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_LO;
875.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
876.                    end
877.                    6'b100000: begin    //clz
878.                        //FLOW
879.
    raddr1<={`VIOLATION_REGFILE_HEAD,inst[25:21]};raddr2<=`VIOLATION_NON;waddr<={`VIO
    LATION_REGFILE_HEAD,inst[15:11]};
880.                        //IF
881.                        mux_pc_sel<=`MUX_PC_NPC;
882.                        //ID
883.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
884.                        cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
885.                        //EX
886.
    alu_sel<=`ALU_CLZ;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
887.                        dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
888.                        //WB
889.                        mux_Rdc_sel<=`MUX_RDC_IR_15_11;mux_Rd_sel<=`MUX_RD_Z;
890.                        hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b1;cp0_w<=1'b0;
891.                    end
```

```
892.                     default: begin    //do nothing
893.                         //FLOW
894.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
895.                         //IF
896.                         mux_pc_sel<=`MUX_PC_NPC;
897.                         //ID
898.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
899.                         cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
900.                         //EX
901.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
902.                         dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
903.                         //WB
904.                         mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
905.                         hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
906.                     end
907.                 endcase
908.             end
909.             default: begin    //do nothing
910.                 //FLOW
911.
    raddr1<=`VIOLATION_NON;raddr2<=`VIOLATION_NON;waddr<=`VIOLATION_NON;
912.                 //IF
913.                 mux_pc_sel<=`MUX_PC_NPC;
914.                 //ID
915.
    mux_ALUa_sel<=`MUX_ALUa_RS;mux_ALUb_sel<=`MUX_ALUb_RT;ext_sel<=`EXT5_Z;
916.                 cp0_exception<=1'b0;cp0_eret<=1'b0;cp0_cause<=`EXC_BREAK;
917.                 //EX
918.
    alu_sel<=`ALU_ADDU;cal_sel<=`CAL_MULTU;cal_ena<=1'b0;use_overflow<=1'b0;
919.                 dmem_w<=1'b0;dmem_width<=`RAM_WIDTH_16;mem_sel<=`EXT8_Z;
920.                 //WB
921.                 mux_Rdc_sel<=`MUX_RDC_IMM31;mux_Rd_sel<=`MUX_RD_Z;
922.                 hi_w<=1'b0;lo_w<=1'b0;regfile_w<=1'b0;cp0_w<=1'b0;
923.             end
924.         endcase
925.     end
926. endmodule
```

5) cp0.v

```
1.  `include "define.vh"
2.
3.  module CP0(
4.      input clk,
5.      input rst,
6.      input mtc0, //cpu instruction mtc0,high-active
7.      input [31:0] pc,
8.
9.      input [4:0] waddr, //specifies CP0 reg to write
10.     input [31:0] wdata, //data from GP reg to place CP0 reg
11.     input exception,    //instruction syscall,break,teq,high-active
12.     input eret, //instruction eret,high-active
13.     input [4:0] cause,
14.
15.     input [4:0] raddr, //specifies CP0 reg to read
```

```
    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 test_egg program out
    output [31:0] reg_12,
    output [31:0] reg_13,
    output [31:0] reg_14
    );

    // (* DONT_TOUCH = "TRUE" *)
    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 test_egg program out
    assign reg_12=array_reg[12];
    assign reg_13=array_reg[13];
    assign reg_14=array_reg[14];
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 test_egg program out
    output [31:0] reg_12,
    output [31:0] reg_13,
    output [31:0] reg_14
    );

    wire [1:0] cond0,cond1,cond2,cond3,cond4;
    wire [6:0] flow_raddr1,flow_raddr2,flow_waddr1,flow_waddr2,flow_waddr3;

    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),

        //for test_egg program out
        .reg_12(reg_12),
        .reg_13(reg_13),
        .reg_14(reg_14)
    );

    wire [31:0] ex_HI,ex_LO,ex_Z,ex_Rt,ex_IR;
```

```
77.     wire mult_div_stall,cal_finish,overflow_stall;
78.
79.     execute ex_inst(
80.         .clk(clk),
81.         .reset(reset),
82.         .alua(id_ALUa),
83.         .alub(id_ALUb),
84.         .id_Rt(id_Rt),
85.         .id_IR(id_IR),
86.         .rHI(ex_HI),
87.         .rLO(ex_LO),
88.         .rZ(ex_Z),
89.         .rRt(ex_Rt),
90.         .rIR(ex_IR),
91.         .cond(cond2),
92.         .mult_div_stall(mult_div_stall),   //cause controller to stall 32 periods
93.         .cal_finish(cal_finish),
94.         .overflow_stall(overflow_stall),
95.         .flow_waddr(flow_waddr1)
96.     );
97.
98.     wire [31:0] me_HI,me_LO,me_Z,me_MEM,me_IR;
99.
100.      memory_access me_inst(
101.          .clk(clk),
102.          .reset(reset),
103.          .ex_HI(ex_HI),
104.          .ex_LO(ex_LO),
105.          .ex_Z(ex_Z),
106.          .ex_Rt(ex_Rt),
107.          .ex_IR(ex_IR),
108.          .rHI(me_HI),
109.          .rLO(me_LO),
110.          .rZ(me_Z),
111.          .rMEM(me_MEM),
112.          .rIR(me_IR),
113.          .cond(cond3),
114.          .flow_waddr(flow_waddr2)
115.      );
116.
117.      write_back wb_inst(
118.          .clk(clk),
119.          .reset(reset),
120.          .me_HI(me_HI),
121.          .me_LO(me_LO),
122.          .me_Z(me_Z),
123.          .me_MEM(me_MEM),
124.          .me_IR(me_IR),
125.          .mux_Rdc_out(regfile_Rdc),
126.          .mux_Rd_out(regfile_Rd),
127.          .Rd_out_for_LO(Rd_out_for_LO),
128.          .cond(cond4),
129.          .hi_w(hi_w),
130.          .lo_w(lo_w),
131.          .cp0_w(cp0_w),
132.          .regfile_w(regfile_w),
133.          .flow_waddr(flow_waddr3)
134.      );
```

```
135.
136.      flow_control flow_control_inst(
137.          .clk(clk),.reset(reset),
138.          .raddr1(flow_raddr1),.raddr2(flow_raddr2),.waddr1(flow_waddr1),.waddr2(
    flow_waddr2),.waddr3(flow_waddr3),
139.          .mult_div_stall(mult_div_stall),.mult_div_over(cal_finish),.overflow_st
    all(overflow_stall),
140.          /*-----------flow_control-----------*/
141.          .cond0(cond0),.cond1(cond1),.cond2(cond2),.cond3(cond3),.cond4(cond4)
142.      );
143.  endmodule
```

8) DIV.v

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

    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);
```

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

10) extend.v

```
1.  `include "define.vh"
2.
3.  module ext(
4.      input [2:0] ext_switch,
5.
6.      input [4:0] iData_len5,
7.      input [7:0] iData_len8,
8.      input [15:0] iData_len16,
9.      input [31:0] iData_len32,
10.     output reg [31:0] oData
11. );
```

```
12.
13. always @(*) begin
14.     case (ext_switch)
15.         `EXT5_Z: oData<={27'b0,iData_len5};
16.         `EXT16_SL2_S:
    oData=iData_len16[15]?{14'h3fff,iData_len16,2'b0}:{14'b0,iData_len16,2'b0};
17.         `EXT16_Z: oData<={16'b0,iData_len16};
18.         `EXT16_S: oData<={{16{iData_len16[15]}},iData_len16};
19.         `EXT8_Z: oData<={24'b0,iData_len8};
20.         `EXT8_S: oData<={{24{iData_len8[7]}},iData_len8};
21.         `EXT32_NON: oData<=iData_len32;
22.         default: oData<=32'b0;
23.     endcase
24. end
25. endmodule
```

11) flow_control.v

```
1.  `include "define.vh"
2.
3.  module flow_control(
4.      input clk,input reset,
5.      input [6:0] raddr1,input [6:0] raddr2,input [6:0] waddr1,input [6:0]
    waddr2,input [6:0] waddr3,
6.      input mult_div_stall,input mult_div_over,input overflow_stall,
7.
8.      /*-----------flow_control-----------*/
9.      output [1:0] cond0,output [1:0] cond1,output [1:0] cond2,output [1:0]
    cond3,output [1:0] cond4
10.     );
11.
12.     reg [1:0] cond[0:4];
13.     assign cond0=cond[0];
14.     assign cond1=cond[1];
15.     assign cond2=cond[2];
16.     assign cond3=cond[3];
17.     assign cond4=cond[4];
18.
19.     reg [1:0] state,nextstate;reg [4:0] cnt;
20.     wire
    violation1=(waddr1!=`VIOLATION_NON)&&(raddr1==waddr1||raddr2==waddr1||(waddr1==`V
    IOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_
    HI||raddr2==`VIOLATION_LO)));
21.     wire
    violation2=(waddr2!=`VIOLATION_NON)&&(raddr1==waddr2||raddr2==waddr2||(waddr2==`V
    IOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_
    HI||raddr2==`VIOLATION_LO)));
22.     wire
    violation3=(waddr3!=`VIOLATION_NON)&&(raddr1==waddr3||raddr2==waddr3||(waddr3==`V
    IOLATION_HILO&&(raddr1==`VIOLATION_HI||raddr1==`VIOLATION_LO||raddr2==`VIOLATION_
    HI||raddr2==`VIOLATION_LO)));
23.     wire violation=violation1|violation2|violation3;
24.
25.     always @(posedge clk or posedge reset) begin
26.         if(reset)
27.             state<=`FLOW_NORMAL;
28.         else
29.             state<=nextstate;
30.     end
```

```
31.
32.     always @(*) begin
33.         case (state)
34.             `FLOW_NORMAL: begin
35.                 if (overflow_stall) begin
36.                     if (violation) begin
37.                         cond[0]<=`PARTS_COND_FLOW;  //IF
38.                         cond[1]<=`PARTS_COND_ZERO;  //ID
39.                         cond[2]<=`PARTS_COND_ZERO;  //EX
40.                         cond[3]<=`PARTS_COND_FLOW;  //ME
41.                         cond[4]<=`PARTS_COND_FLOW;  //WB
42.                         nextstate<=`FLOW_NORMAL;
43.                     end else begin
44.                         cond[0]<=`PARTS_COND_FLOW;  //IF
45.                         cond[1]<=`PARTS_COND_FLOW;  //ID
46.                         cond[2]<=`PARTS_COND_ZERO;  //EX
47.                         cond[3]<=`PARTS_COND_FLOW;  //ME
48.                         cond[4]<=`PARTS_COND_FLOW;  //WB
49.                         nextstate<=`FLOW_NORMAL;
50.                     end
51.                 end
52.                 else if (!mult_div_over&mult_div_stall) begin
53.                     cond[0]<=`PARTS_COND_STALL;  //IF
54.                     cond[1]<=`PARTS_COND_STALL;  //ID
55.                     cond[2]<=`PARTS_COND_STALL;  //EX
56.                     cond[3]<=`PARTS_COND_STALL;  //ME
57.                     cond[4]<=`PARTS_COND_STALL;  //WB
58.                     nextstate<=`FLOW_MULDIV;
59.                 end
60.                 else if (violation) begin
61.                     cond[0]<=`PARTS_COND_STALL;  //IF
62.                     cond[1]<=`PARTS_COND_ZERO;  //ID
63.                     cond[2]<=`PARTS_COND_FLOW;  //EX
64.                     cond[3]<=`PARTS_COND_FLOW;  //ME
65.                     cond[4]<=`PARTS_COND_FLOW;  //WB
66.                     nextstate<=`FLOW_NORMAL;
67.                 end else begin
68.                     cond[0]<=`PARTS_COND_FLOW;  //IF
69.                     cond[1]<=`PARTS_COND_FLOW;  //ID
70.                     cond[2]<=`PARTS_COND_FLOW;  //EX
71.                     cond[3]<=`PARTS_COND_FLOW;  //ME
72.                     cond[4]<=`PARTS_COND_FLOW;  //WB
73.                     nextstate<=`FLOW_NORMAL;
74.                 end
75.             end
76.             `FLOW_MULDIV: begin
77.                 if (mult_div_over) begin
78.                     if (violation) begin
79.                         cond[0]<=`PARTS_COND_STALL;  //IF
80.                         cond[1]<=`PARTS_COND_ZERO;  //ID
81.                         cond[2]<=`PARTS_COND_FLOW;  //EX
82.                         cond[3]<=`PARTS_COND_FLOW;  //ME
83.                         cond[4]<=`PARTS_COND_FLOW;  //WB
84.                         nextstate<=`FLOW_NORMAL;
85.                     end else begin
86.                         cond[0]<=`PARTS_COND_FLOW;  //IF
87.                         cond[1]<=`PARTS_COND_FLOW;  //ID
88.                         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
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
    );

    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'b
0));
    always@(posedge clk or posedge reset)
    begin
        if(reset) begin
            cnt<=0;
            multa<=0;
            multb<=0;
            multpart<=0;
            busy<=0;
        end
```

```
        else begin
            if(start) begin
                cnt<=1;
                multa<={a[31],a};
                multb<={b,1'b0};
                multpart<=0;
                busy<=1;
            end else if(busy) 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;
                    end
                    32: begin
                        {multpart,multb}<={add,multb};
                        cnt<=cnt+1;
                        busy<=0;
                    end
                    default: begin end
                endcase
            end
        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
    );

    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;
        end
        else begin
            if(start) begin
```

```
                cnt<=1;
                multa<=a;
                multb<=b;
                multpart<=0;
                busy<=1;
            end else if(busy) 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;
                    end
                    32: begin
                        {multpart,multb,shiftr}<={cf,add,multb};
                        cnt<=cnt+1;
                        busy<=0;
                    end
                    default: begin end
                endcase
            end
        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(
```

```
33.     input [1:0] sel,
34.     input [31:0] iData_1,
35.     input [31:0] iData_2,
36.     input [31:0] iData_3,
37.     input [31:0] iData_4,
38.     output reg [31:0] oData
39.     );
40. 
41.     //reg oData;  //wire in implementation
42. 
43.     always @(*) begin
44.         case(sel)
45.             2'b00: oData<=iData_1;
46.             2'b01: oData<=iData_2;
47.             2'b10: oData<=iData_3;
48.             2'b11: oData<=iData_4;
49.             default: begin end
50.         endcase
51.     end
52. endmodule
53. 
54. // module mux_len32_sel2(
55. //     input sel,
56. //     input [31:0] iData_1,
57. //     input [31:0] iData_2,
58. //     output reg [31:0] oData
59. //     );
60. 
61. //     //reg oData;  //wire in implementation
62. 
63. //     always @(*) begin
64. //         case(sel)
65. //             1'b0: oData<=iData_1;
66. //             1'b1: oData<=iData_2;
67. //             default: begin end
68. //         endcase
69. //     end
70. // endmodule
71. 
72. 
73. module mux_len5_sel4(
74.     input [1:0] sel,
75.     input [4:0] iData_1,
76.     input [4:0] iData_2,
77.     input [4:0] iData_3,
78.     input [4:0] iData_4,
79.     output reg [4:0] oData
80.     );
81. 
82.     //reg oData;  //wire in implementation
83. 
84.     always @(*) begin
85.         case(sel)
86.             2'b00: oData<=iData_1;
87.             2'b01: oData<=iData_2;
88.             2'b10: oData<=iData_3;
89.             2'b11: oData<=iData_4;
90.             default: begin end
```

```
        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)
```

```
54.                 `PARTS_COND_FLOW: rPC<=mux_pc_out;
55.                 `PARTS_COND_STALL: rPC<=rPC;
56.                 `PARTS_COND_ZERO: rPC<=`PC_ADDR_INIT;
57.                 default: begin end
58.             endcase
59.         end
60.     end
61. 
62.     always @(negedge clk or posedge reset) begin    //flow
63.         if (reset) begin
64.             rIR<=`IR_NON;
65.         end else begin
66.             case (cond)
67.                 `PARTS_COND_FLOW: rIR<=imem_out;
68.                 `PARTS_COND_STALL: rIR<=rIR;
69.                 `PARTS_COND_ZERO: rIR<=`IR_NON;
70.                 default: begin end
71.             endcase
72.         end
73.     end
74. 
75. endmodule
76. 
77. module instruction_decode(
78.     input clk,
79.     input reset,
80. 
81.     input [31:0] if_IR, //to read
82.     input [31:0] if_NPC,
83.     input [4:0] regfile_Rdc,    //also cp0
84.     input [31:0] regfile_Rd,    //also cp0,hi,lo
85.     input [31:0] Rd_out_for_LO, //add for lo *only when hi also write
86. 
87.     output reg [31:0] rALUa,
88.     output reg [31:0] rALUb,
89.     output reg [31:0] rRt,
90.     output reg [31:0] rIR,
91.     output [31:0] cp0_EPC,
92.     output [31:0] cp0_intr_addr,
93.     output [31:0] ext_out,
94.     output [31:0] connect,
95.     output [31:0] regfile_Rs,
96.     output [31:0] regfile_Rt,
97. 
98.     //control signal
99.     input [1:0] cond,
100.      output [2:0] mux_pc_sel,
101.      input hi_w,
102.      input lo_w,
103.      input regfile_w,
104.      input cp0_w,
105.      output [6:0] flow_raddr1,
106.      output [6:0] flow_raddr2,
107. 
108.      //for test_egg program out
109.      output [31:0] reg_12,
110.      output [31:0] reg_13,
111.      output [31:0] reg_14
```

```
112.      );
113.
114.      assign connect={if_NPC[31:28],if_IR[25:0],2'b0};
115.
116.      reg [31:0] rHI,rLO;
117.
118.      wire mux_lo_sel,mux_hi_sel;
119.      wire [31:0] cal_lo_out,cal_hi_out,mux_lo_out,mux_hi_out;
120.      wire [31:0] alua,alub;
121.      wire [31:0] cp0_out;wire [4:0] cp0_cause;
122.
123.      wire [2:0] mux_ALUa_sel;wire [1:0] mux_ALUb_sel;wire [2:0] ext_sel;
124.      wire cp0_eret,cp0_exception;
125.
126.      wire judge_beq,judge_bgez;
127.      assign judge_beq=(regfile_Rs==regfile_Rt);
128.      assign judge_bgez=(regfile_Rs[31]==1'b0);
129.
130.      controller controller_id(
131.          .inst(if_IR),
132.          .judge_beq(judge_beq), //judge BEQ BNE condition to jump
133.          .judge_bgez(judge_bgez), //judge BGEZ condition to jump
134.
135.          /*-----------flow_control-----------*/
136.          .raddr1(flow_raddr1),.raddr2(flow_raddr2),
137.          /*-----------control-----------*/
138.          //ID
139.          .mux_pc_sel(mux_pc_sel),
140.          .mux_ALUa_sel(mux_ALUa_sel),.mux_ALUb_sel(mux_ALUb_sel),.ext_sel(ext_se
    l),
141.          .cp0_exception(cp0_exception),.cp0_eret(cp0_eret),.cp0_cause(cp0_cause)
142.      );
143.
144.      mux_len32_sel8 mux_ALUa(
145.          .sel(mux_ALUa_sel),
146.          .iData_1(rHI),
147.          .iData_2(rLO),
148.          .iData_3(if_NPC),
149.          .iData_4(regfile_Rs),
150.          .iData_5(ext_out),
151.          .iData_6(cp0_out),
152.          .iData_7(32'b0),
153.          .oData(alua)
154.      );
155.      mux_len32_sel4 mux_ALUb(
156.          .sel(mux_ALUb_sel),
157.          .iData_1(32'd0),
158.          .iData_2(regfile_Rt),
159.          .iData_3(ext_out),
160.          .oData(alub)
161.      );
162.      ext ext_id(
163.          .ext_switch(ext_sel),
164.          .iData_len5(if_IR[10:6]),
165.          .iData_len8(),
166.          .iData_len16(if_IR[15:0]),
167.          .iData_len32(),
168.          .oData(ext_out)
```

```
169.      );
170.
171.
172.      regfile cpu_ref(
173.          .clk(clk),  //posedge write-active
174.          .rst(reset),  //active-high asynchronous
175.          .we(regfile_w),   //high:write low:read
176.          .raddr1({27'b0,if_IR[25:21]}),
177.          .raddr2({27'b0,if_IR[20:16]}),
178.          .rdata1(regfile_Rs),
179.          .rdata2(regfile_Rt),
180.          .waddr({27'b0,regfile_Rdc}),
181.          .wdata(regfile_Rd),
182.
183.          //for test_egg program out
184.          .reg_12(reg_12),
185.          .reg_13(reg_13),
186.          .reg_14(reg_14)
187.      );
188.      CP0 cp0_inst(
189.          .clk(clk),
190.          .rst(reset),
191.          .mtc0(cp0_w), //cpu instruction mtc0,high-active
192.          .pc(if_NPC),
193.          .waddr(regfile_Rdc), //specifies CP0 reg to write
194.          .wdata(regfile_Rd), //data from GP reg to place CP0 reg
195.          .exception(cp0_exception&&(cond==`PARTS_COND_FLOW)),    //instruction
    syscall,break,teq,high-active
196.          .eret(cp0_eret&&(cond==`PARTS_COND_FLOW)), //instruction eret,high-
    active
197.          .cause(cp0_cause),
198.          .raddr(if_IR[15:11]), //specifies CP0 reg to read
199.          .rdata(cp0_out),    //data from CP0 reg for GP reg
200.          .exc_addr(cp0_EPC),  //address for PC at the beginning of an exception
201.          .intr_addr(cp0_intr_addr)
202.      );
203.
204.      always @(posedge clk or posedge reset) begin    //execute
205.          if (reset) begin
206.              rALUa<=32'b0;
207.              rALUb<=32'b0;
208.              rRt<=32'b0;
209.              rHI<=32'b0;
210.              rLO<=32'b0;
211.          end else begin
212.              case (cond)
213.                  `PARTS_COND_FLOW: begin
214.                      rALUa<=alua;
215.                      rALUb<=alub;
216.                      rRt<=regfile_Rt;
217.                  end
218.                  `PARTS_COND_STALL: begin
219.                      rALUa<=rALUa;
220.                      rALUb<=rALUb;
221.                      rRt<=rRt;
222.                  end
223.                  `PARTS_COND_ZERO: begin
224.                      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
    );

    wire [31:0] cal_lo,cal_hi;
    wire [1:0] cal_sel;wire cal_ena;
    wire [31:0] alu_z;
    wire [3:0] alu_sel;

    wire use_overflow;
    assign mult_div_stall=cal_ena;
    assign overflow_stall=use_overflow&alu_overflow;

    controller controller_ex(
        .inst(id_IR),
```

```
283.          .judge_beq(1'b0), //judge BEQ BNE condition to jump
284.          .judge_bgez(1'b0), //judge BGEZ condition to jump
285.          /*-----------flow_control-----------*/
286.          .waddr(flow_waddr),
287.          /*-----------control-----------*/
288.          //EX
289.          .alu_sel(alu_sel),.cal_sel(cal_sel),.cal_ena(cal_ena)/*also for
    stall*/,.use_overflow(use_overflow)
290.      );
291.
292.      calculator calculator_inst(
293.          .clk(!clk), //negedge calculate
294.          .a(alua), //multiplicand/dividend
295.          .b(alub), //multiplier/divisor
296.          .calc(cal_sel),
297.          .reset(reset),      //high-active,at the beginning of test
298.          .ena(cal_ena),  //high-active
299.          .oLO(cal_lo),
300.          .oHI(cal_hi),
301.          .sum_finish(cal_finish)
302.      );
303.
304.      alu alu_inst(
305.          .a(alua),
306.          .b(alub),
307.          .aluc(alu_sel),
308.          .r(alu_z),
309.          .overflow(alu_overflow)
310.      );
311.
312.      always @(posedge clk or posedge reset) begin    //execute
313.          if (reset) begin
314.              rHI<=32'b0;
315.              rLO<=32'b0;
316.              rZ<=32'b0;
317.              rRt<=32'b0;
318.          end else begin
319.              case (cond)
320.                  `PARTS_COND_FLOW: begin
321.                      rHI<=cal_hi;
322.                      rLO<=cal_lo;
323.                      rZ<=alu_z;
324.                      rRt<=id_Rt;
325.                  end
326.                  `PARTS_COND_STALL: begin
327.                      rHI<=rHI;
328.                      rLO<=rLO;
329.                      rZ<=rZ;
330.                      rRt<=rRt;
331.                  end
332.                  `PARTS_COND_ZERO: begin
333.                      rHI<=32'b0;
334.                      rLO<=32'b0;
335.                      rZ<=32'b0;
336.                      rRt<=32'b0;
337.                  end
338.                  default: begin end
339.              endcase
```

```
340.         end
341.     end
342.
343.     always @(negedge clk or posedge reset) begin    //flow
344.         if (reset) begin
345.             rIR<=`IR_NON;
346.         end else begin
347.             case (cond)
348.                 `PARTS_COND_FLOW: rIR<=id_IR;
349.                 `PARTS_COND_STALL: rIR<=rIR;
350.                 `PARTS_COND_ZERO: rIR<=`IR_NON;
351.                 default: begin end
352.             endcase
353.         end
354.     end
355. endmodule
356.
357. module memory_access(
358.     input clk,
359.     input reset,
360.
361.     input [31:0] ex_HI,
362.     input [31:0] ex_LO,
363.     input [31:0] ex_Z,
364.     input [31:0] ex_Rt,
365.     input [31:0] ex_IR,
366.
367.     output reg [31:0] rHI,
368.     output reg [31:0] rLO,
369.     output reg [31:0] rZ,
370.     output reg [31:0] rMEM,
371.     output reg [31:0] rIR,
372.
373.     //control signal
374.     input [1:0] cond,
375.     output [6:0] flow_waddr
376.     );
377.
378.     wire dmem_w;wire [1:0] dmem_width;wire [31:0] dmem_out,ext_out;wire [2:0]
    mem_sel;
379.
380.     controller controller_me(
381.         .inst(ex_IR),
382.         .judge_beq(1'b0), //judge BEQ BNE condition to jump
383.         .judge_bgez(1'b0), //judge BGEZ condition to jump
384.         /*-----------flow_control-----------*/
385.         .waddr(flow_waddr),
386.         /*-----------control-----------*/
387.         //ME
388.         .dmem_w(dmem_w),.dmem_width(dmem_width),.mem_sel(mem_sel)
389.     );
390.
391.     ram dmem_inst(
392.         .clk(clk),  //posedge read/write-active
393.         .wena(dmem_w), //high:write low:read
394.         .width(dmem_width),  //0:word,1:hword,2:byte
395.         .addr(ex_Z-32'h10010000),
396.         .data_in(ex_Rt),   //use zero side
```

```
397.          .data_out(dmem_out)   //use zero side
398.      );
399.
400.      ext ext_me(
401.          .ext_switch(mem_sel),
402.          .iData_len5(),
403.          .iData_len8(dmem_out[7:0]),
404.          .iData_len16(dmem_out[15:0]),
405.          .iData_len32(dmem_out),
406.          .oData(ext_out)
407.      );
408.
409.      always @(posedge clk or posedge reset) begin    //execute
410.          if (reset) begin
411.              rHI<=32'b0;
412.              rLO<=32'b0;
413.              rZ<=32'b0;
414.              rMEM<=32'b0;
415.          end else begin
416.              case (cond)
417.                  `PARTS_COND_FLOW: begin
418.                      rHI<=ex_HI;
419.                      rLO<=ex_LO;
420.                      rZ<=ex_Z;
421.                      rMEM<=ext_out;
422.                  end
423.                  `PARTS_COND_STALL: begin
424.                      rHI<=rHI;
425.                      rLO<=rLO;
426.                      rZ<=rZ;
427.                      rMEM<=rMEM;
428.                  end
429.                  `PARTS_COND_ZERO: begin
430.                      rHI<=32'b0;
431.                      rLO<=32'b0;
432.                      rZ<=32'b0;
433.                      rMEM<=32'b0;
434.                  end
435.                  default: begin end
436.              endcase
437.          end
438.      end
439.
440.      always @(negedge clk or posedge reset) begin    //flow
441.          if (reset) begin
442.              rIR<=`IR_NON;
443.          end else begin
444.              case (cond)
445.                  `PARTS_COND_FLOW: rIR<=ex_IR;
446.                  `PARTS_COND_STALL: rIR<=rIR;
447.                  `PARTS_COND_ZERO: rIR<=`IR_NON;
448.                  default: begin end
449.              endcase
450.          end
451.      end
452.  endmodule
453.
454.  module write_back(
```

```
455.      input clk,
456.      input reset,
457.
458.      input [31:0] me_HI,
459.      input [31:0] me_LO,
460.      input [31:0] me_Z,
461.      input [31:0] me_MEM,
462.      input [31:0] me_IR,
463.
464.      output [4:0] mux_Rdc_out,
465.      output [31:0] mux_Rd_out,
466.      output [31:0] Rd_out_for_LO,
467.
468.      //control signal
469.      input [1:0] cond,
470.      output hi_w,
471.      output lo_w,
472.      output cp0_w,
473.      output regfile_w,
474.      output [6:0] flow_waddr
475.      );
476.      assign Rd_out_for_LO=me_LO;
477.
478.      wire [1:0] mux_Rdc_sel,mux_Rd_sel;
479.      wire hi_w_in,lo_w_in,cp0_w_in,regfile_w_in;
480.      wire write_ena=(cond==`PARTS_COND_FLOW);
481.      assign hi_w=write_ena&hi_w_in;
482.      assign lo_w=write_ena&lo_w_in;
483.      assign cp0_w=write_ena&cp0_w_in;
484.      assign regfile_w=write_ena®file_w_in;
485.
486.      controller controller_wb(
487.          .inst(me_IR),
488.          .judge_beq(1'b0), //judge BEQ BNE condition to jump
489.          .judge_bgez(1'b0), //judge BGEZ condition to jump
490.
491.          /*-----------flow_control-----------*/
492.          .waddr(flow_waddr),
493.          /*-----------control-----------*/
494.          //WB
495.          .mux_Rdc_sel(mux_Rdc_sel),.mux_Rd_sel(mux_Rd_sel),
496.          .hi_w(hi_w_in),.lo_w(lo_w_in),.regfile_w(regfile_w_in),.cp0_w(cp0_w_in)
497.      );
498.
499.      mux_len5_sel4 mux_Rdc(
500.          .sel(mux_Rdc_sel),
501.          .iData_1(me_IR[15:11]),
502.          .iData_2(me_IR[20:16]),
503.          .iData_3(5'd31),
504.          .oData(mux_Rdc_out)
505.      );
506.      mux_len32_sel4 mux_Rd(
507.          .sel(mux_Rd_sel),
508.          .iData_1(me_Z),
509.          .iData_2(me_MEM),
510.          .iData_3(me_HI),
511.          .iData_4(me_LO),
512.          .oData(mux_Rd_out)
```

```
513.      );
514.  endmodule
```

16) ram.v

```
1.  `include "define.vh"
2.
3.  module ram(
4.      input clk,  //posedge read/write-active
5.      input wena, //high:write low:read
6.
7.      input [1:0] width,  //0:word,1:hword,2:byte
8.      input [31:0] addr,
9.      input [31:0] data_in,   //use zero side
10.     output reg [31:0] data_out   //use zero side
11.     );
12.
13.     reg [31:0] memory [0:4095]; //16KB
14.     wire [31:0] ram_addr;
15.     assign ram_addr=addr>>2;
16.
17.     always @(posedge clk) begin
18.         if(wena) begin
19.             case (width)
20.                 `RAM_WIDTH_32: memory[ram_addr]<=data_in;   //addr[1:0]=00
21.                 `RAM_WIDTH_16: begin     //addr[0]=0
22.                     if(addr[1]==1'b0)
23.                         memory[ram_addr][15:0]<=data_in[15:0];
24.                     else
25.                         memory[ram_addr][31:16]<=data_in[15:0];
26.                 end
27.                 `RAM_WIDTH_8: begin
28.                     case (addr[1:0])
29.                         2'b00: memory[ram_addr][7:0]<=data_in[7:0];
30.                         2'b01: memory[ram_addr][15:8]<=data_in[7:0];
31.                         2'b10: memory[ram_addr][23:16]<=data_in[7:0];
32.                         2'b11: memory[ram_addr][31:24]<=data_in[7:0];
33.                         default: begin end
34.                     endcase
35.                 end
36.                 default: begin end
37.             endcase
38.         end
39.     end
40.
41.     always @(*) begin
42.         case (width)
43.             `RAM_WIDTH_32: data_out[31:0]<=memory[ram_addr];   //addr[1:0]=00
44.             `RAM_WIDTH_16: begin      //addr[0]=0
45.                 if(addr[1]==1'b0)
46.                     data_out<={16'b0,memory[ram_addr][15:0]};
47.                 else
48.                     data_out<={16'b0,memory[ram_addr][31:16]};
49.             end
50.             `RAM_WIDTH_8: begin
51.                 case (addr[1:0])
52.                     2'b00: data_out<={24'b0,memory[ram_addr][7:0]};
53.                     2'b01: data_out<={24'b0,memory[ram_addr][15:8]};
54.                     2'b10: data_out<={24'b0,memory[ram_addr][23:16]};
```

```
                    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;

```

```
48.    reg [7:0] seg_data_r;
49.    always @ (*)
50.      case(seg7_addr)
51.               0 : seg_data_r = i_data_store[3:0];
52.               1 : seg_data_r = i_data_store[7:4];
53.               2 : seg_data_r = i_data_store[11:8];
54.               3 : seg_data_r = i_data_store[15:12];
55.               4 : seg_data_r = i_data_store[19:16];
56.               5 : seg_data_r = i_data_store[23:20];
57.               6 : seg_data_r = i_data_store[27:24];
58.               7 : seg_data_r = i_data_store[31:28];
59.             endcase
60.
61.    reg [7:0] o_seg_r;
62.    always @ (posedge clk, posedge reset)
63.      if(reset)
64.               o_seg_r <= 8'hff;
65.             else
66.               case(seg_data_r)
67.                 4'h0 : o_seg_r <= 8'hC0;
68.          4'h1 : o_seg_r <= 8'hF9;
69.          4'h2 : o_seg_r <= 8'hA4;
70.          4'h3 : o_seg_r <= 8'hB0;
71.          4'h4 : o_seg_r <= 8'h99;
72.          4'h5 : o_seg_r <= 8'h92;
73.          4'h6 : o_seg_r <= 8'h82;
74.          4'h7 : o_seg_r <= 8'hF8;
75.          4'h8 : o_seg_r <= 8'h80;
76.          4'h9 : o_seg_r <= 8'h90;
77.          4'hA : o_seg_r <= 8'h88;
78.          4'hB : o_seg_r <= 8'h83;
79.          4'hC : o_seg_r <= 8'hC6;
80.          4'hD : o_seg_r <= 8'hA1;
81.          4'hE : o_seg_r <= 8'h86;
82.          4'hF : o_seg_r <= 8'h8E;
83.               endcase
84.
85.    assign o_sel = o_sel_r;
86.    assign o_seg = o_seg_r;
87.
88. endmodule
```

18) seg7_top.v

```
1.  `define CLK_PERIOD 3
2.
3.  module seg7_top(
4.      input clk_in,
5.    input reset,
6.    output [7:0] o_seg,
7.    output [7:0] o_sel,
8.     output egg_break
9.
10.    // output [31:0] test_pc,
11.    // output [31:0] test_ir
12.    );
13.
14.    wire [31:0] test_pc,test_ir;
15.
```

```
    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-3];
    assign clk_seg7=counter[`CLK_PERIOD-1];

    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_12,reg_13,reg_14;
    assign seg7_idata={reg_12[15:0],reg_13[15:0]};
    assign egg_break=reg_14[0];

    seg7x16 seg7x16_inst(
        .clk(clk_cpu),
      .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 test_egg program out
        .reg_12(reg_12),
        .reg_13(reg_13),
        .reg_14(reg_14)
    );
endmodule
```

3、 静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_12,reg_13,reg_14;

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

```

```
13.         reg_12,
14.         reg_13,
15.         reg_14
16.     );
17.
18.
19.     integer file_output;
20.     integer counter=0;
21.
22.     initial begin
23.         file_output=$fopen("output.txt");
24.
    //$readmemh("C:/DigitalLogic/DLProject/project_51/test.hex",cpu_tb.uut.mem.memory
    );
25.         pc_history[0]=32'h0000_0000;
26.         clk=0;
27.         reset=1;
28.         #275;
29.         reset=0;
30.     end
31.
32.     always begin
33.         #4;
34.         clk=~clk;
35.         #1;
36.         if(clk==1'b0&&reset==0) begin
37.
    if(counter==1000/*||(inst_history[3]===32'h00000000&&pc_history[3]!=32'h00400000)
    */) begin
38.                 $fclose(file_output);
39.                 $finish;
40.             end
41.             else if(pc_history[0]!=pc) begin
42.                 pc_history[4]=pc_history[3];
43.                 pc_history[3]=pc_history[2];
44.                 pc_history[2]=pc_history[1];
45.                 pc_history[1]=pc_history[0];
46.                 pc_history[0]=pc;
47.
48.                 inst_history[4]=inst_history[3];
49.                 inst_history[3]=inst_history[2];
50.                 inst_history[2]=inst_history[1];
51.                 inst_history[1]=inst_history[0];
52.                 inst_history[0]=inst;
53.
54.                 if (pc_history[4]!=32'h0000_0000) begin
55.                     counter=counter+1;
56.                     $fdisplay(file_output,"pc: %h",pc_history[4]);
57.                     $fdisplay(file_output,"instr: %h",inst_history[4]);
58.
    $fdisplay(file_output,"regfile0: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[0]);
59.
    $fdisplay(file_output,"regfile1: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[1]);
60.
    $fdisplay(file_output,"regfile2: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[2]);
61.
    $fdisplay(file_output,"regfile3: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[3]);
```

```
62.
    $fdisplay(file_output,"regfile4: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[4]);
63.
    $fdisplay(file_output,"regfile5: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[5]);
64.
    $fdisplay(file_output,"regfile6: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[6]);
65.
    $fdisplay(file_output,"regfile7: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[7]);
66.
    $fdisplay(file_output,"regfile8: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[8]);
67.
    $fdisplay(file_output,"regfile9: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[9]);
68.
    $fdisplay(file_output,"regfile10: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[10]);
69.
    $fdisplay(file_output,"regfile11: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[11]);
70.
    $fdisplay(file_output,"regfile12: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[12]);
71.
    $fdisplay(file_output,"regfile13: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[13]);
72.
    $fdisplay(file_output,"regfile14: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[14]);
73.
    $fdisplay(file_output,"regfile15: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[15]);
74.
    $fdisplay(file_output,"regfile16: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[16]);
75.
    $fdisplay(file_output,"regfile17: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[17]);
76.
    $fdisplay(file_output,"regfile18: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[18]);
77.
    $fdisplay(file_output,"regfile19: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[19]);
78.
    $fdisplay(file_output,"regfile20: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[20]);
79.
    $fdisplay(file_output,"regfile21: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[21]);
80.
    $fdisplay(file_output,"regfile22: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[22]);
81.
    $fdisplay(file_output,"regfile23: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[23]);
82.
    $fdisplay(file_output,"regfile24: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[24]);
83.
    $fdisplay(file_output,"regfile25: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[25]);
84.
    $fdisplay(file_output,"regfile26: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[26]);
85.
    $fdisplay(file_output,"regfile27: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[27]);
86.
    $fdisplay(file_output,"regfile28: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[28]);
87.
    $fdisplay(file_output,"regfile29: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[29]);
88.
    $fdisplay(file_output,"regfile30: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[30]);
89.
    $fdisplay(file_output,"regfile31: %h",cpu_tb.uut.id_inst.cpu_ref.array_reg[31]);
90.                 end
91.             end
```

```
        end
    end
endmodule
```

2) cpu_synthesis_implementation_tb.v

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

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

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

```