计算机系统结构课程实验报告

——静态流水线设计与性能分析

同济大学
TONGJI UNIVERSITY

院　　系　电子与信息工程学院

专　　业　计算机科学与技术

姓　　名　庄镇华

学　　号　1853790

题　　目　静态流水线设计与性能分析

指导老师　陆有军

联系方式　17721295617

完成日期　2020.11.20

目录

一、实验环境部署与硬件配置说明

VIVADO 的安装

a.关闭防火墙软件，访问 https://www.xilinx.com/support/download.html，下载 vivado 在线安装文件（或者下载完整安装包），运行安装程序，注册（登录）Xilinx 账号。

b.打开在线安装程序，输入注册账号以及选择下载位置，将安装文件放在英文目录下，点击 next，看到软件下载信息，点击 download 开始下载。

c.下载完成打开下载目录，双击 xsetup.exe 进行安装。

d.选择 vivado HL Design Edition。

e.选择安装位置，注意不要使用中文以及带空格的目录。

f.点击 next，看到安装信息，开始安装。

g.证书的安装，选择左侧的 load license，然后点击 copy license，选择证书， 安装结束。

MODELSIM 的安装

a.访问 https://www.mentor.com/products/fpga/download/modelsim-pe-simulator- download 下载 modelsimPE 安装包。

b.关闭防火墙软件，执行安装程序，点击 NEXT。

c.进入安装界面，此处可更改默认的安装路径。

d.系统提示安装路径不存在，是否建立新的安装路径，点击 Yes。

e.安装 Key Driver，点击 Yes（win10 系统选择 No）。

f.安装完 License 之后重启。

g.建系统变量，变量名为 MGLS_LICENSE_FILE。

VIVADO 和MODELSIM 软件的关联

a.打开 Vivado 2016.2 Tcl Shell。

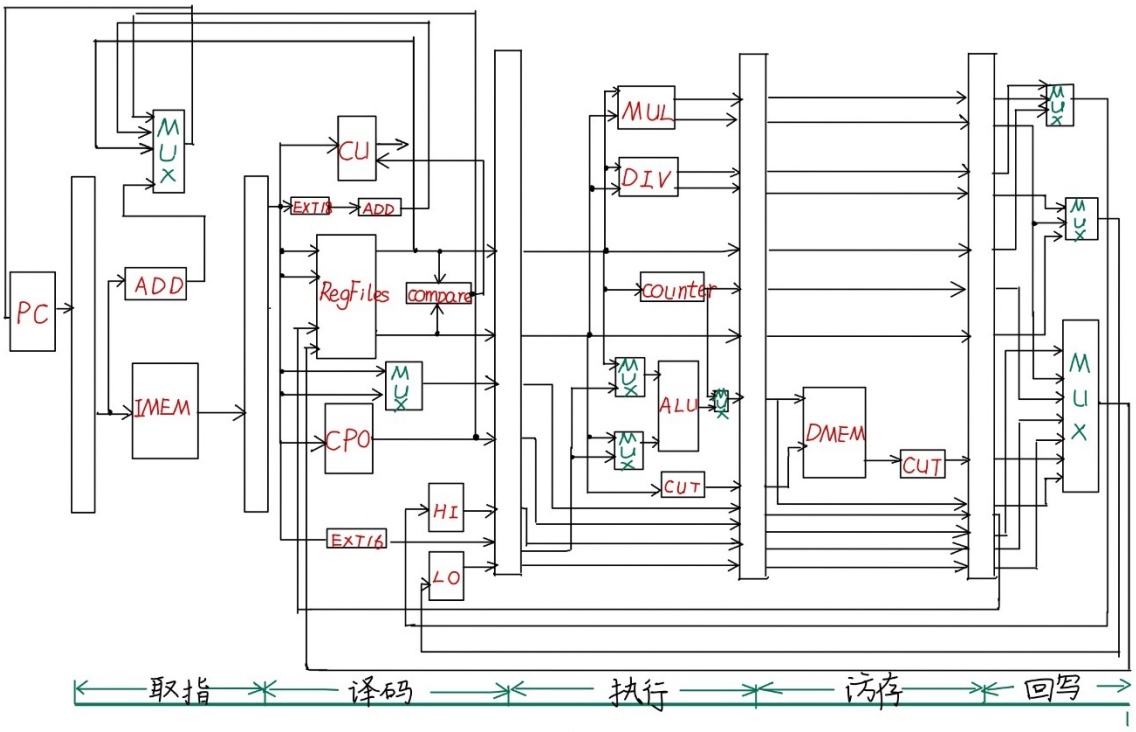
b.输入命令：compile_simlib -directory <保存编译生成的库文件的路径> -simulator modelsim -simulator_exec_path <Modelsim 可执行文件的路径>

c.编译结束，查看 D:/xilinx_sim_lib 文件夹，已经生成了库文件。

d.设置关联，打开 Vivado。Tools-->options-->General 选择 modelsim 的安装路径。在工程中对仿真工具进行配置，如下图。 选择 Simulation Setting-->Simulation-->将 Target simulation 设为 modelsim, Compiled library location 设为 D:/xilinx_sim_lib。

二、静态流水线的总体结构

多功能静态流水线的总体结构包括执行部件与段间的流水寄存器（流水锁存器），由执行部件与段间的流水寄存器形成相应的数据通路。



静态流水线数据流图

三、静态流水线总体结构部件的解释说明

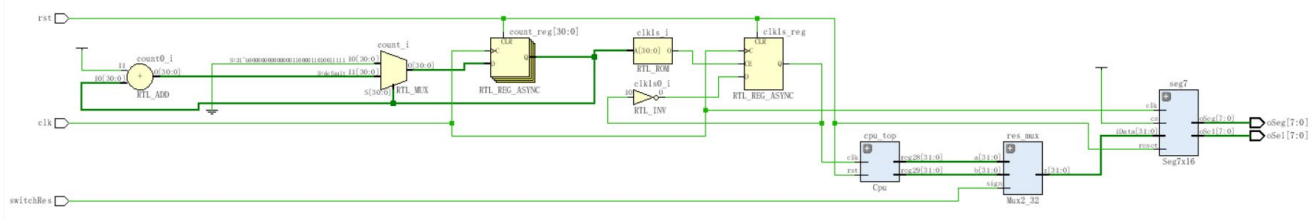
五级流水线各个阶段主要工作如下。

取指阶段：从指令存储器读出指令，同时确定下一条指令地址。

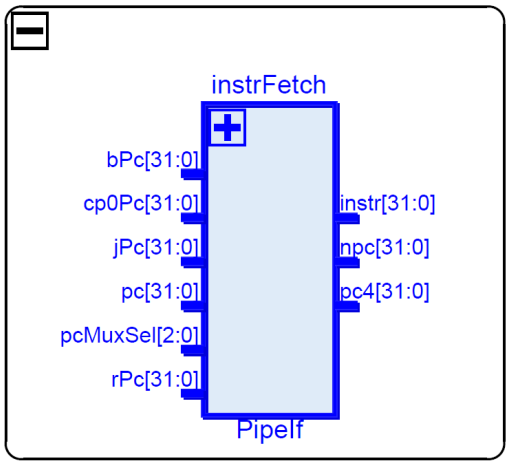
执行阶段：按照译码阶段给出的操作数、运算类型，进行计算，给出计算结果。如果是 Load/Store 指令，那么还会计算 Load/Store 的目标地址。

访存阶段：如果是 Load/Store 指令，那么在此阶段访问数据存储器，反之，只是将执行阶段的结果向下传递到回写阶段。

回写阶段：将运算结果保存到目标寄存器。



IF 级




```
    // 取指段接口定义
module PipeIf(
    input [31 : 0] pc,
    input [31 : 0] cp0Pc,  // 中断跳转地址
    input [31 : 0] bPc,    // beq bne bgez 跳转地址
    input [31 : 0] rPc,    // jr jalr 跳转地址
    input [31 : 0] jPc,    // j jal 跳转地址
    input [2 : 0] pcMuxSel,// 选择信号
    output [31 : 0] npc,   // next pc
    output [31 : 0] pc4,   // pc + 4
    output [31 : 0] instr  // 输出指令
    );
```

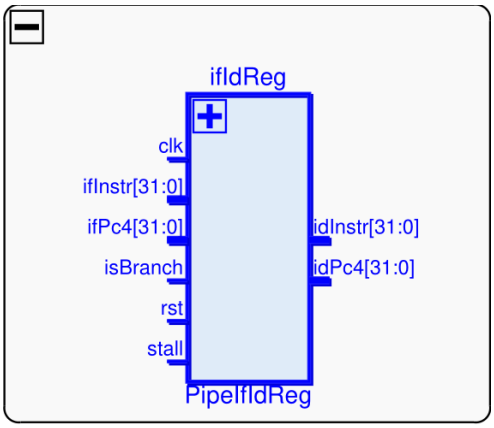
```
    //代码描述
    imem instr_mem(pc[12 : 2], instr);
    Add32 pc_plus4_adder(pc, 32'h4, pc4);
    Mux8_32 next_pc_mux(jPc, rPc, pc4, `EXCEPTION_ADDR, bPc, cp0Pc, 32'b0, 32'b0,
pcMuxSel, npc);
```

IF 级部件： 包含指令存储器、多路选择器和累加器。

输入：PC 的各种来源，PC 多路选择器的控制信号

输出：下一条 PC、PC + 4 以及当前指令

IF/ID 级间流水寄存器




```
    // 取指、译码寄存器接口定义
module PipeIfIdReg(
    input clk,
    input rst,
    input [31 : 0] ifPc4,
    input [31 : 0] ifInstr,
    input stall,
    input isBranch,
    output reg [31 : 0] idPc4,
    output reg [31 : 0] idInstr
    );
```

该模块用于存放取出的指令并执行 PC + 4 操作。

ID 级

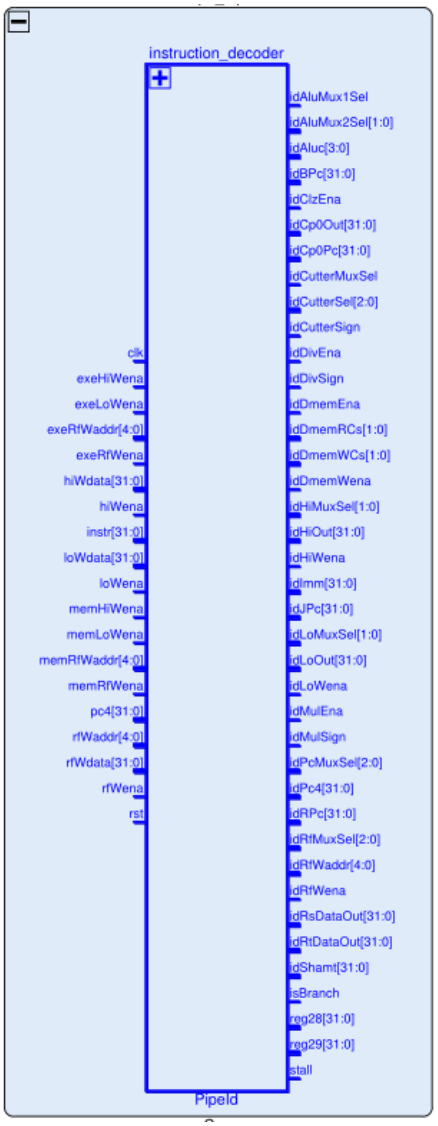
ID 级部件： 包含寄存器堆、控制单元、CP0 协处理器、Hi 寄存器、Lo 寄存器、立即数扩展器、计算转移地址的加法器和多路选择器。

输入：从 WB 级传回的写信号、写地址和写数据，IF 级传递的值

输出：各类控制信号、向 EXE 级传递的各类寄存器读出的值。

```
    // 译码段接口定义
module PipeId(
    input clk,
    input rst,
    input [31 : 0] pc4,
    input [31 : 0] instr,

    //寄存器堆数据
    input [31 : 0] rfWdata,    // 写入寄存器堆的数据
    input [4 : 0] rfWaddr,     // 写入寄存器堆的地址
    input rfWena,              // 寄存器堆写信号
    output [31 : 0] idRsDataOut,
    output [31 : 0] idRtDataOut,
    output [31 : 0] reg28,
    output [31 : 0] reg29

    //HI、LO 寄存器数据
    input [31 : 0] hiWdata,
    input [31 : 0] loWdata,
```

```
    input hiWena,               // HI 寄存器写信号
    input loWena,               // LO 寄存器写信号
    output [31 : 0] idHiOut,      // HI 寄存器输出数据
    output [31 : 0] idLoOut,      // LO 寄存器输出数据

    // EXE 数据
    input [4 : 0] exeRfWaddr,
    input exeRfWena,
    input exeHiWena,
    input exeLoWena,

    // MEM 数据
    input [4 : 0] memRfWaddr,
    input memRfWena,
    input memHiWena,
    input memLoWena,

    // CP0 数据
    output [31 : 0] idCp0Pc,  // 中断地址
    output [31 : 0] idCp0Out,     // CP0 输出数据

    output [31 : 0] idRPc,
    output [31 : 0] idBPc,     // 转移地址
    output [31 : 0] idJPc,     // 跳转地址
    output [31 : 0] idImm,     // 立即数
    output [31 : 0] idShamt,
    output [31 : 0] idPc4,
    output [4 : 0] idRfWaddr,

    // 控制器数据
    output [3 : 0] idAluc,
    output idMulSign,
    output idDivSign,
    output idCutterSign,
    output idClzEna,
    output idMulEna,
    output idDivEna,
    output idDmemEna,
    output idHiWena,
    output idLoWena,
    output idRfWena,
    output idDmemWena,
    output [1 : 0] idDmemWCs,
    output [1 : 0] idDmemRCs,
    output idCutterMuxSel,
    output idAluMux1Sel,
    output [1 : 0] idAluMux2Sel,
    output [1 : 0] idHiMuxSel,
    output [1 : 0] idLoMuxSel,
    output [2 : 0] idCutterSel,
    output [2 : 0] idRfMuxSel,
    output [2 : 0] idPcMuxSel,

    output stall,
    output isBranch,
);
```

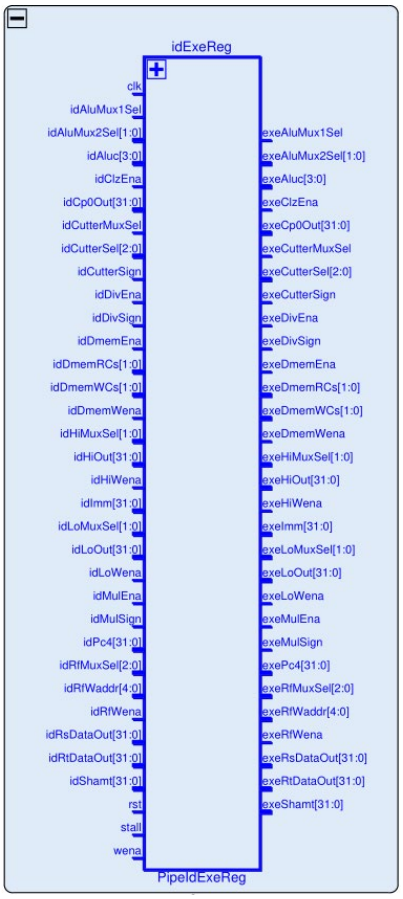


ID/EXE 级间流水寄存器

ID 级与 EXE 级间的流水寄存器：传递 ID 级输出的控制信号和读出的数据

```
    // 译码执行寄存器接口定义
module PipeIdExeReg(
    input clk,
    input rst,
    input wena,

    // ID 段输入
    input [31 : 0] idPc4,       // ID 阶段的 PC
    input [31 : 0] idRsDataOut,
    input [31 : 0] idRtDataOut,
    input [31 : 0] idImm,       // ID 阶段的立即数
    input [31 : 0] idShamt,
    input [31 : 0] idCp0Out,    // ID 阶段的 CP0 输出
    input [31 : 0] idHiOut,     // ID 阶段的 HI 寄存器输出
    input [31 : 0] idLoOut,     // ID 阶段的 LO 寄存器输出
    input [4 : 0] idRfWaddr,
```

```
    input idClzEna,
    input idMulEna,
    input idDivEna,
    input idDmemEna,
    input idMulSign,
    input idDivSign,
    input idCutterSign,
    input [3 : 0] idAluc,       // ID 阶段的 ALUC
    input idRfWena,
    input idHiWena,
    input idLoWena,
    input idDmemWena,
    input [1 : 0] idDmemWCs,
    input [1 : 0] idDmemRCs,
    input stall,
    input idCutterMuxSel,
    input idAluMux1Sel,
    input [1 : 0] idAluMux2Sel,
    input [1 : 0] idHiMuxSel,
    input [1 : 0] idLoMuxSel,
    input [2 : 0] idCutterSel,
    input [2 : 0] idRfMuxSel,

    // EXE 段输出
    output reg [31 : 0] exePc4,
    output reg [31 : 0] exeRsDataOut,
    output reg [31 : 0] exeRtDataOut,
    output reg [31 : 0] exeImm,
    output reg [31 : 0] exeShamt,
    output reg [31 : 0] exeCp0Out,
    output reg [31 : 0] exeHiOut,
    output reg [31 : 0] exeLoOut,
    output reg [4 : 0] exeRfWaddr,
    output reg exeClzEna,
    output reg exeMulEna,
    output reg exeDivEna,
    output reg exeDmemEna,
    output reg exeMulSign,
    output reg exeDivSign,
    output reg exeCutterSign,
    output reg [3 : 0] exeAluc,
    output reg exeRfWena,
    output reg exeHiWena,
    output reg exeLoWena,
    output reg exeDmemWena,
    output reg [1 : 0] exeDmemWCs,
    output reg [1 : 0] exeDmemRCs,
    output reg exeAluMux1Sel,
    output reg [1 : 0] exeAluMux2Sel,
    output reg exeCutterMuxSel,
    output reg [1 : 0] exeHiMuxSel,
    output reg [1 : 0] exeLoMuxSel,
    output reg [2 : 0] exeCutterSel,
    output reg [2 : 0] exeRfMuxSel
);
```



EXE 级

EXE 级别部件： 包含了 ALU 模块、乘法器模块、除法器模块、计算前导零的计算模块和多路选择器。

输入：ID 级传递的控制信号以及各类源操作数值

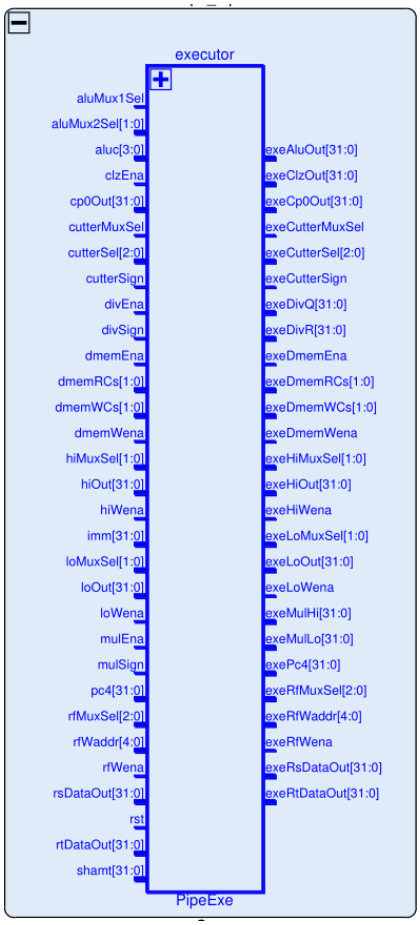
输出：向 MEM 级传递的控制信号以及计算的结果。

```
    // 执行段接口定义
module PipeExe(
    input rst,

    // 运算器输入 a
    input [31 : 0] rtDataOut,
    input [31 : 0] imm,

    // 运算器输入 b
    input [31 : 0] rsDataOut,
    input [31 : 0] shamt,

    output [31 : 0] exeRsDataOut,
    output [31 : 0] exeRtDataOut,

    // 运算器数据
    input [3 : 0] aluc,
    input aluMux1Sel,
    input [1 : 0] aluMux2Sel,
```

```
    output [31 : 0] exeAluOut,

    // 乘除法器
    input [31 : 0] hiOut,
    input [31 : 0] loOut,
    input mulSign,
    input divSign,
    input mulEna,
    input divEna,
    input [1 : 0] hiMuxSel,
    input [1 : 0] loMuxSel,
    input hiWena,
    input loWena,
    output [1 : 0] exeHiMuxSel,
    output [1 : 0] exeLoMuxSel,
    output [31 : 0] exeMulHi,
    output [31 : 0] exeMulLo,
    output [31 : 0] exeDivR,
    output [31 : 0] exeDivQ,
    output exeHiWena,
    output exeLoWena,
    output [31 : 0] exeHiOut,
    output [31 : 0] exeLoOut,

    //前导零计算
    input clzEna,
    output [31 : 0] exeClzOut,

    // 其他
    input [31 : 0] cp0Out,
    input [4 : 0] rfWaddr,
    input [31 : 0] pc4,
    input dmemWena,
    input [1 : 0] dmemWCs,
    input [1 : 0] dmemRCs,
    input cutterSign,
    input cutterMuxSel,
    input [2 : 0] cutterSel,
    input [2 : 0] rfMuxSel,
    input rfWena,
    input dmemEna,
    output [31 : 0] exePc4,
    output exeDmemWena,
    output [1 : 0] exeDmemWCs,
    output [1 : 0] exeDmemRCs,
    output exeCutterSign,
    output exeCutterMuxSel,
    output [2 : 0] exeCutterSel,
    output [2 : 0] exeRfMuxSel,
    output exeRfWena,
    output exeDmemEna,
    output [31 : 0] exeCp0Out,
    output [4 : 0] exeRfWaddr,
);
```

```
    //实现代码
    Div cpu_div(rst, divEna, divSign, rsDataOut, rtDataOut, exeDivQ, exeDivR);
```

```
    Mult cpu_mult(rst, mulEna, mulSign, rsDataOut, rtDataOut, exeMulHi,
exeMulLo);
    Counter cpu_counter(rsDataOut, clzEna, exeClzOut);
    Mux2_32 alu_mux1(shamt, rsDataOut, aluMux1Sel, aluIn1);
    Mux4_32 alu_mux2(rtDataOut, imm, 32'b0, 32'b0, aluMux2Sel, aluIn2);
    Alu cpu_alu(aluIn1, aluIn2, aluc, exeAluOut, zero, carry, negative,
overflow);
```

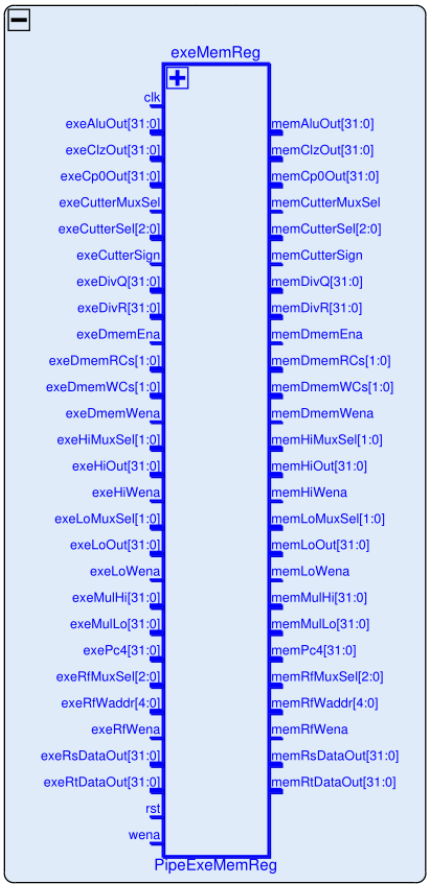


EXE/MEM 级间流水寄存器

EXE/MEM 级间流水寄存器： 负责存储 EXE 级输出的数据和控制信号。

```
    // 执行、访存寄存器接口定义
module PipeExeMemReg(
    // EXE 段输入
    input clk,
    input rst,
    input wena,
    input [31 : 0] exeMulHi,
    input [31 : 0] exeMulLo,
    input [31 : 0] exeDivR,
    input [31 : 0] exeDivQ,
    input [31 : 0] exeClzOut,
    input [31 : 0] exeAluOut,
    input [31 : 0] exePc4,
```

```
    input [31 : 0] exeRsDataOut,
    input [31 : 0] exeRtDataOut,
    input [31 : 0] exeCp0Out,
    input [31 : 0] exeHiOut,
    input [31 : 0] exeLoOut,
    input [4 : 0] exeRfWaddr,
    input exeDmemEna,
    input exeCutterSign,
    input exeRfWena,
    input exeHiWena,
    input exeLoWena,
    input exeDmemWena,
    input [1 : 0] exeDmemWCs,
    input [1 : 0] exeDmemRCs,
    input exeCutterMuxSel,
    input [1 : 0] exeHiMuxSel,
    input [1 : 0] exeLoMuxSel,
    input [2 : 0] exeCutterSel,
    input [2 : 0] exeRfMuxSel,

    // MEM 段输出
    output reg [31 : 0] memMulHi,
    output reg [31 : 0] memMulLo,
    output reg [31 : 0] memDivR,
    output reg [31 : 0] memDivQ,
    output reg [31 : 0] memClzOut,
    output reg [31 : 0] memAluOut,
    output reg [31 : 0] memPc4,
    output reg [31 : 0] memRsDataOut,
    output reg [31 : 0] memRtDataOut,
    output reg [31 : 0] memCp0Out,
    output reg [31 : 0] memHiOut,
    output reg [31 : 0] memLoOut,
    output reg [4 : 0] memRfWaddr,
    output reg memDmemEna,
    output reg memCutterSign,
    output reg memRfWena,
    output reg memHiWena,
    output reg memLoWena,
    output reg memDmemWena,
    output reg [1 : 0] memDmemWCs,
    output reg [1 : 0] memDmemRCs,
    output reg memCutterMuxSel,
    output reg [1 : 0] memHiMuxSel,
    output reg [1 : 0] memLoMuxSel,
    output reg [2 : 0] memCutterSel,
    output reg [2 : 0] memRfMuxSel
);
```



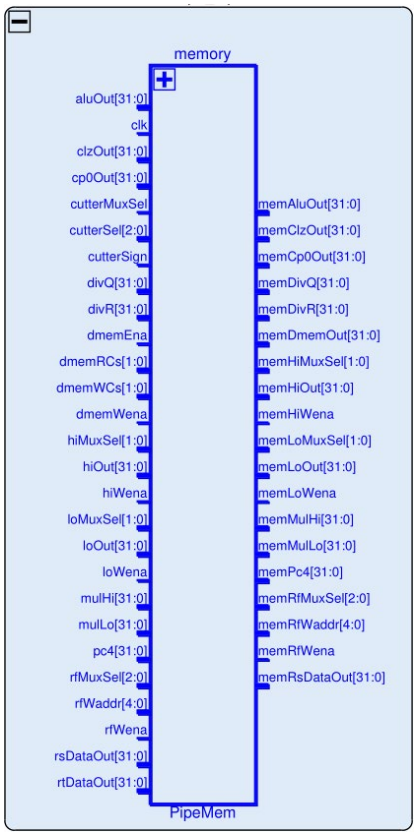
MEM 级

MEM 级部件： 包含了数据存储器模块、选择数据长度、符号扩展的模块和多路选择器。

输入：EXE 级的计算结果和传递的控制信号

输出：传递的控制信号与读出的结果。

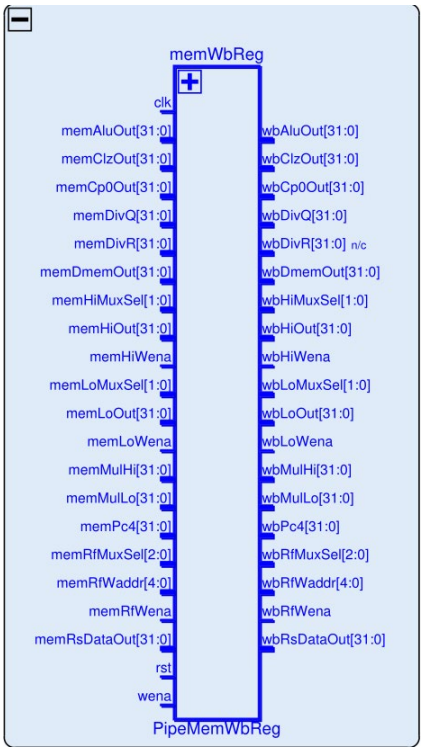
```
    // 访存段接口定义
module PipeMem(
    input clk,

    // 数据存储器
    input dmemEna,
    input dmemWena,
    input [1 : 0] dmemWCs,
    input [1 : 0] dmemRCs,
    input [31 : 0] aluOut,
    output [31 : 0] memDmemOut,

    input cutterSign,
    input cutterMuxSel,
    input [2 : 0] cutterSel,
    // 输入数据
    input [31 : 0] mulHi,
    input [31 : 0] mulLo,
    input [31 : 0] divR,
    input [31 : 0] divQ,
```




```
    input [31 : 0] clzOut,
    input [31 : 0] pc4,
    input [31 : 0] rsDataOut,
    input [31 : 0] rtDataOut,
    input [31 : 0] cp0Out,
    input [31 : 0] hiOut,
    input [31 : 0] loOut,
    input [4 : 0] rfWaddr,
    input rfWena,
    input hiWena,
    input loWena,
    input [1 : 0] hiMuxSel,
    input [1 : 0] loMuxSel,
    input [2 : 0] rfMuxSel,

    // 对应的输出数据
    output [31 : 0] memMulHi,
    output [31 : 0] memMulLo,
    output [31 : 0] memDivR,
    output [31 : 0] memDivQ,
    output [31 : 0] memClzOut,
    output [31 : 0] memAluOut,
    output [31 : 0] memPc4,
    output [31 : 0] memRsDataOut,
    output [31 : 0] memCp0Out,
    output [31 : 0] memHiOut,
    output [31 : 0] memLoOut,
    output [4 : 0] memRfWaddr,
    output memRfWena,
```

```
    output memHiWena,
    output memLoWena,
    output [1 : 0] memHiMuxSel,
    output [1 : 0] memLoMuxSel,
    output [2 : 0] memRfMuxSel
);
```

MEM/WB 级间流水线寄存器

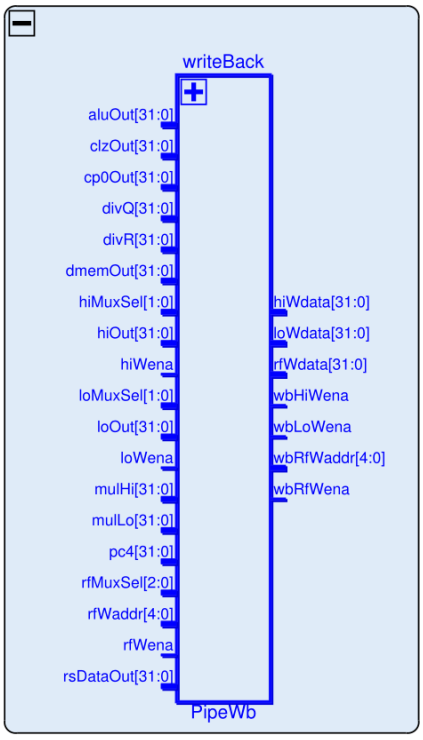


MEM/WB 级间流水线寄存器：存放控制信号和各类待写入的结果数值

```
    // 访存、写回寄存器定义
module PipeMemWbReg (
    input clk,
    input rst,
    input wena,
    // 输入数据
    input [31 : 0] memMulHi,
    input [31 : 0] memMulLo,
    input [31 : 0] memDivR,
    input [31 : 0] memDivQ,
    input [31 : 0] memClzOut,
    input [31 : 0] memAluOut,
    input [31 : 0] memDmemOut,
    input [31 : 0] memPc4,
    input [31 : 0] memRsDataOut,
    input [31 : 0] memCp0Out,
    input [31 : 0] memHiOut,
    input [31 : 0] memLoOut,
    input [4 : 0] memRfWaddr,
    input memRfWena,
    input memHiWena,
    input memLoWena,
```

```
    input [1 : 0] memHiMuxSel,
    input [1 : 0] memLoMuxSel,
    input [2 : 0] memRfMuxSel,

    //对应的输出数据
    output reg [31 : 0] wbMulHi,
    output reg [31 : 0] wbMulLo,
    output reg [31 : 0] wbDivR,
    output reg [31 : 0] wbDivQ,
    output reg [31 : 0] wbClzOut,
    output reg [31 : 0] wbAluOut,
    output reg [31 : 0] wbDmemOut,
    output reg [31 : 0] wbPc4,
    output reg [31 : 0] wbRsDataOut,
    output reg [31 : 0] wbCp0Out,
    output reg [31 : 0] wbHiOut,
    output reg [31 : 0] wbLoOut,
    output reg [4 : 0] wbRfWaddr,
    output reg wbRfWena,
    output reg wbHiWena,
    output reg wbLoWena,
    output reg [1 : 0] wbHiMuxSel,
    output reg [1 : 0] wbLoMuxSel,
    output reg [2 : 0] wbRfMuxSel
);
```

WB 级



WB 级部件：包含了 Hi、Lo、RegFiles 输入数据的多路选择器。

输入：各类计算结果和控制信号

输出：写回 ID 级的写信号、写地址和写数据。

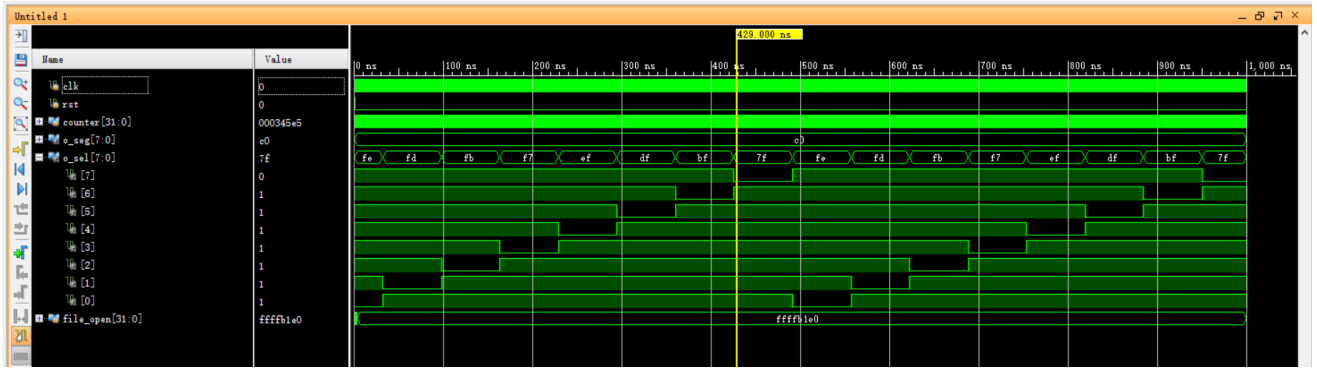
```
    // 写回段接口定义
module PipeWb(
    // 写回 HI 寄存器的数据
    input [31 : 0] divR,
    input [31 : 0] mulHi,
    input [1 : 0] hiMuxSel,
    output [31 : 0] hiWdata,

    // 写回 LO 寄存器的数据
    input [31 : 0] divQ,
    input [31 : 0] mulLo,
    input [1 : 0] loMuxSel,
    output [31 : 0] loWdata,
    input [31 : 0] rsDataOut,

    // 写回寄存器堆的数据
    input [31 : 0] loOut,
    input [31 : 0] pc4,
    input [31 : 0] clzOut,
    input [31 : 0] cp0Out,
    input [31 : 0] dmemOut,
    input [31 : 0] aluOut,
    input [31 : 0] hiOut,
    input [2 : 0] rfMuxSel,
    output [31 : 0] rfWdata,

    // 输入数据
    input [4 : 0] rfWaddr,
    input rfWena,
    input hiWena,
    input loWena,

    // 对应的输出数据
    output [4 : 0] wbRfWaddr,
    output wbRfWena,
    output wbHiWena,
    output wbLoWena
);
```

四、静态流水线的仿真过程

前仿真测试方法：指令存储器中调用 ip 核，将指令读入指令存储器，使用下图仿真文件 testbench.v 进行前仿真测试。

```
module testbench;
    reg clk, rst;
    reg [31 : 0] counter;
    wire [7 : 0] o_seg, o_sel;
    initial begin
        clk = 1'b0;
        rst = 1'b1;
        counter = 1;
        #1 rst = ~rst;
    end
    always begin
        #1 clk = ~clk;
    end
    pipe_top uut(.clk(clk), .rst(rst), .oSeg(o_seg), .oSel(o_sel));
endmodule
```

得到如下波形图：

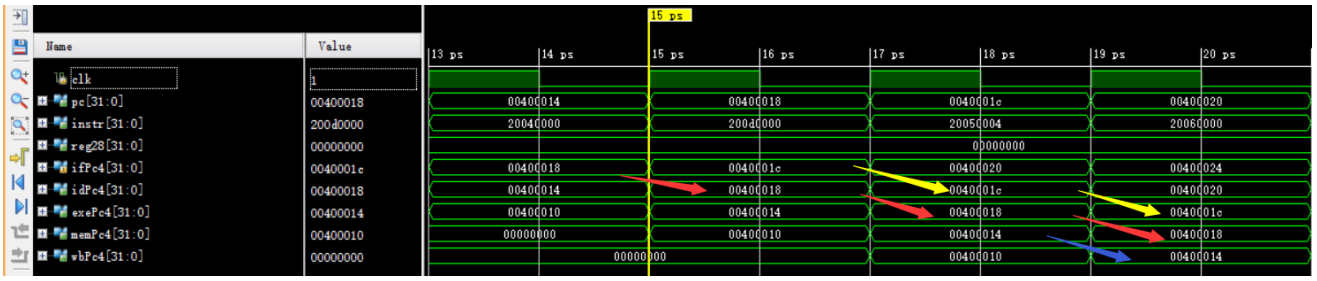


五、实验仿真的波形图及某时刻寄存器值的物理意义

静态流水线的波形图及某时刻寄存器值的物理意义

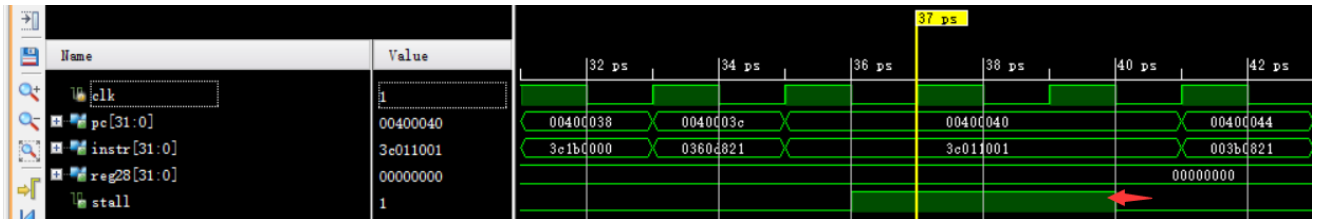
正常执行

横向看，分别是每一级的信号输出（只列出相关的写信号与数据）；纵向看，是每个时钟周期内每一级的当前输出。如下图所示，在 IF 级、ID 级、EXE 级、MEM 级的 PC + 4 信号都成功传递到下一周期的 ID 级、EXE 级、MEM 级和 WB 级。

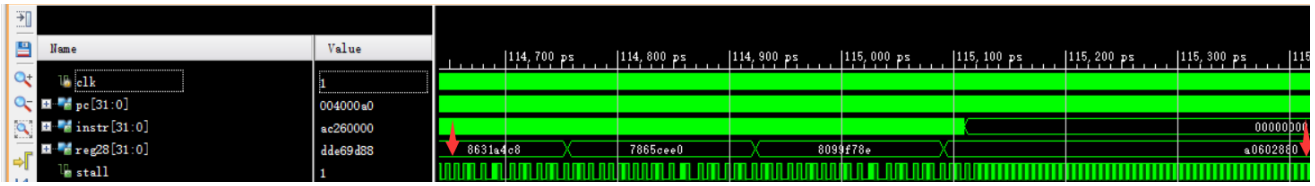


遇到数据冲突时的暂停情况

横向看，分别是每一级的信号输出（只列出相关的写信号与数据）；纵向看，是每个时钟周期内每一级的当前输出。模块使用寄存器 stall 的值来确定是否发生冲突，如果 stall == 1，代表发生冲突，解决方法是暂停周期，等待上一条指令冲突阶段结束。如下图所示，相继执行的两条指令为 0x3C1B0000（lui $27,0x0000）和 0x0360D821（addu $27,$27,$0），此时发生数据真相关。ID 级在指令 0x3C011001 检测到冲突，暂停了 2 个周期，直到第一条指令的写回级执行完，才继续执行下去。



寄存器值的物理意义



28 号寄存器保存的是不同时刻 E 数组的值，截图中是最后四次，与计算出来的值吻合。

0x8631a4c8	0x7865cee0	0x8099f78e	0xa0602880

六、实验验算数学模型及算法程序

算法流程

```
int a[m], b[m], c[m], d[m];
a[0] = 0;
b[0] = 1;
a[i] = a[i - 1] + i;
b[i] = b[i - 1] + 3 * i;
```

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

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

汇编代码

```
.data
    A:.space 240
    B:.space 240
    C:.space 240
    D:.space 240
    E:.space 240

.text
    j main
    exc:
    nop
    j exc

main:
    addi $2,$0,0 # a[i]
    addi $3,$0,1 # b[i]
    addi $4,$0,0 # c[i]
    addi $13,$0,0 # d[i]
    addi $5,$0,4 # counter
    addi $6,$0,0 # a[i - 1]
    addi $7,$0,1 # b[i - 1]
    addi $10,$0,0 # flag for i < 20 || i < 40
    addi $11,$0,240 # sum counts
    addi $14,$0,3

    addi $30,$0,0
    lui $27,0x0000
    addu $27,$27,$0
    sw $2,A($27)

    lui $27,0x0000
    addu $27,$27,$0
    sw $3,B($27)

    lui $27,0x0000
    addu $27,$27,$0
    sw $2,C($27)

    addi $31,$31,1
    lui $27,0x0000
    addu $27,$27,$0
    sw $3,D($27)
```

```
loop:
    srl $12,$5,2
    add $6,$6,$12

    lui $27,0x0000
    addu $27,$27,$5
    sw $6,A($27)

    mul $15,$14,$12

    add $7,$7,$15
    lui $27,0x0000
    addu $27,$27,$5
    sw $7,B($27)

    slti $10,$5,80
    bne $10,1,c1

    lui $27,0x0000 # i <= 19
    addu $27,$27,$5
    sw $7,D($27)

    addi $15,$6,0
    addi $16,$7,0
    j endc
c1:                 # 20 <= i <= 39
    slti $10,$5,160
    addi $27,$0,1
    bne $10,$27,c2

    add $15,$6,$7   # c[i] = a[i] + b[i]
    lui $27,0x0000
    addu $27,$27,$5
    sw $15,C($27)

    mul $16,$15,$6  # d[i] = a[i] * b[i]
    lui $27,0x0000
    addu $27,$27,$5
    sw $16,D($27)

    j endc

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

```
    mul $16,$15,$7  # d[i] = c[i] * b[i]
    lui $27,0x0000
    addu $27,$27,$5
    sw $16,D($27)
```

```
endc:
    add $28,$15,$16
    lui $27,0x0000
    addu $27,$27,$5
    sw $28,E($27)
    addi $5,$5,4
    bne $5,$11,loop
```

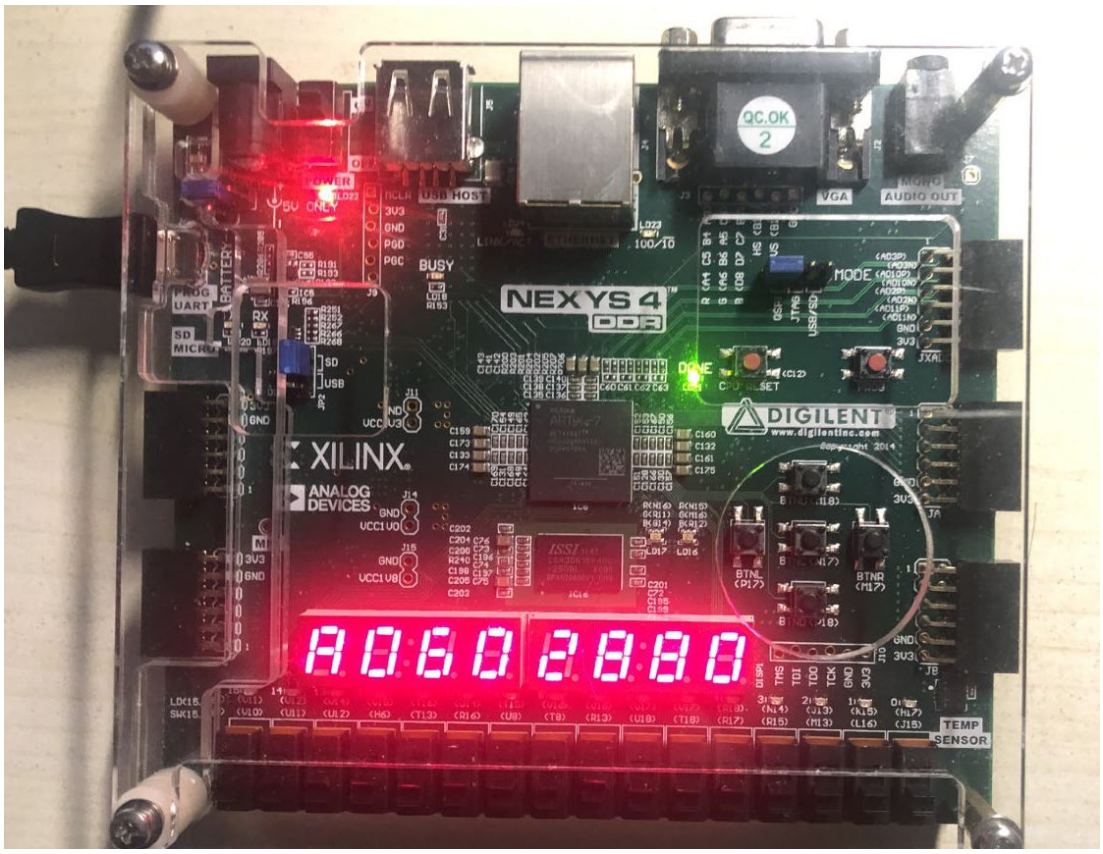
程序完成算法模型，并将 C 的结果存于 15 号寄存器中，D 的结果存于 16 号寄存器中，将 E（C + D）的结果存于 28 号寄存器中。

七、实验验算程序下板测试过程与实现

下板测试

- 时钟信号分频，降低频率，方便人眼识别。
- 将数码管模块与静态流水线 CPU 模块连接。
- 分析综合，编写管脚约束文件；
- 生成 bitstream，下板运行。

下图展示了运行停止后显示 C[59] + D[59]的值



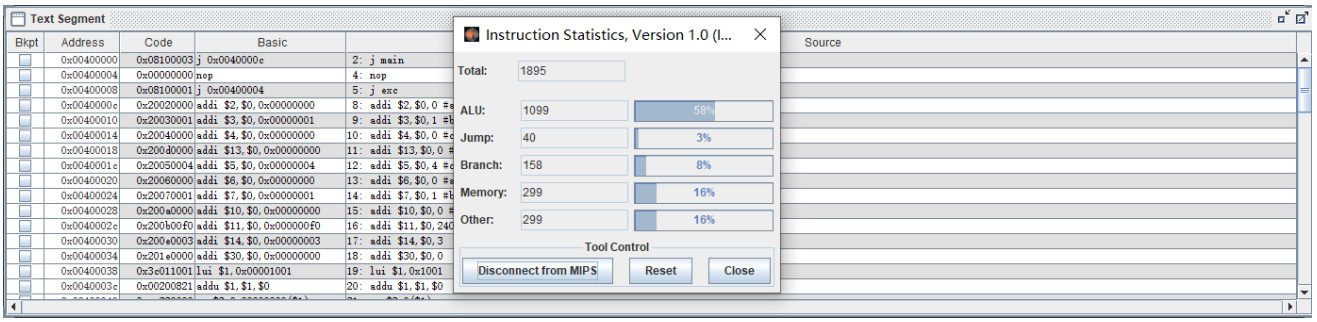
结果检验

利用高级程序语言计算结果，得到下表，与上述下板结果对比无误。

数组名	A	B	C	D	E
16 进制表示	0x000006ea	0x000014bf	0x008f7096	0x9fd0b7ea	0xa0602880

八、静态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）



如上图，易知程序执行的指令条数为 1895 条，根据波形图得到执行周期为 3052 个时钟周期。

吞吐率

吞吐率指在单位时间内流水线所完成的指令条数。本静态流水线 CPU 完成任务执行了 1896 条指令，一共使用了 3052 个时钟周期，因此吞吐率计算为：

$$\text{吞吐率} = \frac{1896}{3052\text{个时钟周期}} = 0.6212/\text{时钟周期}$$

加速比

不使用流水线和使用流水线所用的时间比称为流水线的加速比。对于本次测试，由于汇编程序里面没有 ld 指令，故 299 条 memory 访问指令全为 sw 指令，顺序执行需要 4 个时钟周期；分支指令 158 条，顺序执行需要 3 个时钟周期；ALU 指令 1099 条，顺序执行需要 4 个时钟周期；跳转指令 40 条，顺序执行需要 2 个时钟周期，其他指令 299 条，顺序执行认为需要 3 个时钟周期。因此加速比计算为：

$$\text{加速比} = \frac{299 \times 4 + 158 \times 3 + 1099 \times 4 + 40 \times 2 + 299 \times 3\text{个时钟周期}}{3052\text{个时钟周期}} = 2.308$$

效率

流水线的效率指流水线中各功能段的利用率，即流水线各段处于工作时间的时空区与流水线各段总的时空区的比值。对于本静态流水线 CPU，效率的计算如下：

$$\text{效率} = \frac{299 \times 4 + 158 \times 3 + 1099 \times 4 + 40 \times 2 + 299 \times 3}{3052 \times 5} \times 100\% = \frac{7043}{15260} = 46.15\%$$

冲突分析

数据相关

如果一条指令在 ID 阶段发现它使用到了现在正处于 EXE 或者 MEM 阶段的指令所要写的寄存器中的数据，则 stall 延迟信号有效，待前置指令全部进入 WB 阶段再继续执行。

```
//EXE阶段或MEM阶段对Hi寄存器访问冲突情况
if (stall == `RUN) begin
    if (op == `MFHI_OP && func == `MFHI_FUNC) begin
        if (exeHiWena == `WRITE_ENABLED) begin
            stall <= `STOP;
        end else if (memHiWena == `WRITE_ENABLED) begin
            stall <= `STOP;
        end
    end
end
```

控制相关

利用延迟槽和提前判断分支技术。在 ID 阶段进行转移条件的判断，如果满足转移条件，修改 PC 为转移目标地址，令处于延迟槽内的指令失效，即封锁处于 ID 阶段指令的任何写有效信号，如果延迟槽内也是分支指令，则禁止其对 PC 的修改。

如果不满足转移条件，延迟槽内的指令正常执行。

同时读写寄存器堆

如果需要同时读写同一单元，则将写入的值作为读到的值输出，便可解决冲突。

```
if((raddr == waddr) && (wena == `WRITE_ENABLED) && (rena == `READ_ENABLED))
    rdata2 <= wdata;
```

九、总结与体会

在本次的静态流水线实验中，我对 CPU 的运行有了更全面的认识。静态流水线 CPU 不仅比单周期、多周期 CPU 在效率上有很大提升，而且思路更加巧妙。

经过实践，我认为一个较好的流程是：先翻翻课本，初步认识流水线，知道流水线分段、暂停、延时槽机制，然后是画图和填表，弄清楚有哪些部件，部件之间如何连接，每条指令在每个阶段要进行什么操作。接着是流水线控制器设计，控制器主要有两个功能：产生控制信号、控制流水线暂停和运行。在设计之前，需要思考：指令在流水线中执行需要多个周期，如何保证控制器在一个周期发出的信号能够在正确的时间被相应的部件收到？如何判断需要暂停流水线的情况？如何使流水线暂停？事实上，控制器需要实时接收流水线各段执行的指令：对于 ID 段的指令，发出执行该条指令所需的所有控制信号，并通过段间流水寄存器传递到各段。对于 EX 段和 MEM 段的指令，判断是否与 ID 段的指令产生数据冲突。注意在静态流水线中，只会产生写后读冲突。

设计完控制器后，基本可以开始编写代码。模块主要有控制器、PC 寄存器、指令存储器和数据寄存器（IP 核）、寄存器堆、CP0 协处理器、HI、LO 寄存器、位拓展模块、乘除法器、ALU、DMEM 的数据位转换模块、各种多路选择器、段间流水寄存器。比较好的顺序是：编写除控制器外部件，编写控制器，连接各模块。

最后是测试与检验，根据实验报告模板提供的算法流程编写汇编语言，然后仿真、分析、综合，下板。感想就是硬件设计不能一蹴而就，检验耗费的时间、精力都是软件测试不能比的，所以需要更多地耐心与真诚。本次实验也有一些不足与缺陷，导致实现的静态流水线效率不是很高，后续的动态流水线实验和三级储存实验，我会更加的认真去完成。

十、附件（静态流水线的设计程序）

```
`define STATUS              12
`define CAUSE               13
`define EPC                 14

`define STOP                1'b1
`define RUN                 1'b0

`define ENABLED              1'b1
`define DISABLED             1'b0
`define RST_ENABLED          1'b1
`define RST_DISABLED         1'b0
`define READ_ENABLED         1'b1
`define READ_DISABLED        1'b0
`define WRITE_ENABLED        1'b1
`define WRITE_DISABLED       1'b0
`define CHIP_ENABLED         1'b1
`define CHIP_DISABLED        1'b0
`define SIGNED               1'b1
`define UNSIGNED             1'b0

`define SYSCALL             5'b01000
`define BREAK               5'b01001
`define TEQ                 5'b01101

`define ADDU_FUNC       6'b100001
`define AND_FUNC        6'b100100
`define JR_FUNC         6'b001000
`define XOR_FUNC         6'b100110
`define NOR_FUNC        6'b100111
`define OR_FUNC         6'b100101
`define SLL_FUNC        6'b000000
`define SLLV_FUNC       6'b000100
`define SLTU_FUNC       6'b101011
`define SRA_FUNC        6'b000011
`define SRL_FUNC        6'b000010
`define SUBU_FUNC       6'b100011
`define ADD_FUNC        6'b100000
`define SUB_FUNC        6'b100010
`define SLT_FUNC        6'b101010
`define SRLV_FUNC       6'b000110
`define SRAV_FUNC       6'b000111
`define CLZ_FUNC        6'b100000
`define DIVU_FUNC       6'b011011
`define ERET_FUNC       6'b011000
`define JALR_FUNC       6'b001001
`define MFHI_FUNC       6'b010000
`define MFLO_FUNC       6'b010010
`define MTHI_FUNC       6'b010001
`define MTLO_FUNC       6'b010011
`define MUL_FUNC        6'b000010
```

```
`define MULTU_FUNC      6'b011001
`define SYSCALL_FUNC    6'b001100
`define TEQ_FUNC        6'b110100
`define BREAK_FUNC      6'b001101
`define DIV_FUNC        6'b011010

`define ADDI_OP       6'b001000
`define ADDIU_OP      6'b001001
`define ANDI_OP       6'b001100
`define ORI_OP        6'b001101
`define SLTIU_OP      6'b001011
`define LUI_OP        6'b001111
`define XORI_OP       6'b001110
`define SLTI_OP       6'b001010
`define ADDU_OP       6'b000000
`define AND_OP        6'b000000
`define BEQ_OP        6'b000100
`define BNE_OP        6'b000101
`define JR_OP         6'b000000
`define LW_OP         6'b100011
`define XOR_OP        6'b000000
`define NOR_OP        6'b000000
`define OR_OP         6'b000000
`define SLL_OP        6'b000000
`define SLLV_OP       6'b000000
`define SLTU_OP       6'b000000
`define SRA_OP        6'b000000
`define SRL_OP        6'b000000
`define SUBU_OP       6'b000000
`define SW_OP         6'b101011
`define ADD_OP        6'b000000
`define SUB_OP        6'b000000
`define SLT_OP        6'b000000
`define SRLV_OP       6'b000000
`define SRAV_OP       6'b000000
`define CLZ_OP        6'b011100
`define DIVU_OP       6'b000000
`define ERET_OP       6'b010000
`define LHU_OP        6'b100101
`define SB_OP         6'b101000
`define SH_OP         6'b101001
`define LH_OP         6'b100001
`define MFHI_OP       6'b000000
`define MFLO_OP       6'b000000
`define MTHI_OP       6'b000000
`define MTLO_OP       6'b000000
`define MUL_OP        6'b011100
`define MULTU_OP      6'b000000
`define SYSCALL_OP    6'b000000
`define TEQ_OP        6'b000000
`define BGEZ_OP       6'b000001
`define BREAK_OP      6'b000000
`define DIV_OP        6'b000000
`define J_OP          6'b000010
`define JAL_OP        6'b000011
`define JALR_OP       6'b000000
`define LB_OP         6'b100000
`define LBU_OP        6'b100100

`define ZERO_32BIT            32'h00000000
```

```
`define EXCEPTION_ADDR        32'h00400004
```

```
`timescale 1ns / 1ps

module PcReg(
    input clk,
    input rst,
    input wena,
    input stall,  // 暂停
    input [31:0] data_in,    // 下一个 pc
    output [31:0] data_out    // 当前 pc
    );

    reg [31:0] pc;

    always @ (posedge clk or posedge rst)
    begin
        if(rst == `RST_ENABLED) begin
            pc <= 32'h00400000;
        end

        else if(stall == `RUN) begin
            if(wena == `WRITE_ENABLED) begin
                pc <= data_in;
            end
        end
    end

    assign data_out = pc;

endmodule



    //寄存器
module Reg(
    input clk,
    input rst,
    input wena,
    input [31 : 0] in,
    output reg [31 : 0] out
    );

    always @ (negedge clk or posedge rst)
    begin
        if(rst == 1) begin
            out <= 0;
        end else if(wena == 1) begin
            out <= in;
        end
    end

endmodule
```

```
`timescale 1ns / 1ps
```

```
module PipeIf(
    input [31 : 0] pc,
    input [31 : 0] cp0Pc,  // 中断跳转地址
    input [31 : 0] bPc,    // beq bne bgez 跳转地址
    input [31 : 0] rPc,    // jr jalr 跳转地址
    input [31 : 0] jPc,    // j jal 跳转地址
    input [2 : 0] pcMuxSel,
    output [31 : 0] npc,   // next pc
    output [31 : 0] pc4,   // pc + 4
    output [31 : 0] instr
    );

    imem instr_mem(pc[12 : 2], instr);
    Add32 pc_plus4_adder(pc, 32'h4, pc4);
    Mux8_32 next_pc_mux(jPc, rPc, pc4, `EXCEPTION_ADDR, bPc, cp0Pc, 32'b0, 32'b0,
pcMuxSel, npc);

endmodule

module PipeIfIdReg(
    input clk,
    input rst,
    input [31 : 0] ifPc4,
    input [31 : 0] ifInstr,
    input stall,
    input isBranch,
    output reg [31 : 0] idPc4,
    output reg [31 : 0] idInstr
    );

    always @ (posedge clk or posedge rst)
    begin
        if (rst == `RST_ENABLED) begin
            idPc4 <= 32'h0;
            idInstr <= 32'h0;
        end else if (isBranch == 1'b1 && stall == `RUN) begin
            idPc4 <= 32'h0;
            idInstr <= 32'h0;
        end else if (stall == `RUN) begin
            idPc4 <= ifPc4;
            idInstr <= ifInstr;
        end
    end

endmodule



module PipeId(
    input clk,
    input rst,
    input [31 : 0] pc4,
    input [31 : 0] instr,
    input [31 : 0] rfWdata,    //写入寄存器堆的数据
    input [31 : 0] hiWdata,
    input [31 : 0] loWdata,
    input [4 : 0] rfWaddr,     // 写入寄存器堆的地址
    input rfWena,              // 寄存器堆写信号
    input hiWena,              // HI 寄存器写信号
```

```
    input loWena,               // LO 寄存器写信号

    // EXE 数据
    input [4 : 0] exeRfWaddr,
    input exeRfWena,
    input exeHiWena,
    input exeLoWena,

    // MEM 数据
    input [4 : 0] memRfWaddr,
    input memRfWena,
    input memHiWena,
    input memLoWena,

    output [31 : 0] idCp0Pc,  // 中断地址
    output [31 : 0] idRPc,
    output [31 : 0] idBPc,    // 跳转地址
    output [31 : 0] idJPc,    // 条件转移地址
    output [31 : 0] idRsDataOut,
    output [31 : 0] idRtDataOut,
    output [31 : 0] idImm,    // 立即数
    output [31 : 0] idShamt,
    output [31 : 0] idPc4,
    output [31 : 0] idCp0Out,    // CP0 输出数据
    output [31 : 0] idHiOut,   // HI 寄存器输出数据
    output [31 : 0] idLoOut,   // LO 寄存器输出数据
    output [4 : 0] idRfWaddr,
    output [3 : 0] idAluc,
    output idMulSign,
    output idDivSign,
    output idCutterSign,
    output idClzEna,
    output idMulEna,
    output idDivEna,
    output idDmemEna,
    output idHiWena,
    output idLoWena,
    output idRfWena,
    output idDmemWena,
    output [1 : 0] idDmemWCs,
    output [1 : 0] idDmemRCs,
    output idCutterMuxSel,
    output idAluMux1Sel,
    output [1 : 0] idAluMux2Sel,
    output [1 : 0] idHiMuxSel,
    output [1 : 0] idLoMuxSel,
    output [2 : 0] idCutterSel,
    output [2 : 0] idRfMuxSel,
    output [2 : 0] idPcMuxSel,
    output stall,
    output isBranch,
    output [31 : 0] reg28,
    output [31 : 0] reg29
    );

    //协处理器
    wire mfc0;
    wire mtc0;
```

```
    wire eret;
    wire cp0Exception;
    wire [4 : 0] cp0Cause;
    wire [4 : 0] cp0Addr;
    wire [31 : 0] cp0Status;

    //寄存器堆
    wire [4 : 0] rd;
    wire [5 : 0] op;
    wire [4 : 0] rs;
    wire [4 : 0] rt;
    wire [5 : 0] func;
    wire rfRena1;
    wire rfRena2;

    //扩展器
    wire ext5MuxSel;
    wire [4 : 0] ext5MuxOut;
    wire ext16Sign;
    wire [15 : 0] ext16DataIn;
    wire [17 : 0] ext18DataIn;
    wire [31 : 0] ext16DataOut;
    wire [31 : 0] ext18DataOut;

    assign func = instr[5 : 0];
    assign rt = instr[20 : 16];
    assign rs = instr[25 : 21];
    assign op = instr[31 : 26];
    assign ext16DataIn = instr[15 : 0];
    assign ext18DataIn = {instr[15 : 0], 2'b00};

    assign idImm = ext16DataOut;
    assign idJPc = {pc4[31 : 28], instr[25 : 0], 2'b00};
    assign idRPc = idRsDataOut;
    assign idPc4 = pc4;
    assign idRfWaddr = rd;

    Add32 b_pc_adder(pc4, ext18DataOut, idBPc);

    Reg lo_reg(clk, rst, loWena, loWdata, idLoOut);

    Reg hi_reg(clk, rst, hiWena, hiWdata, idHiOut);

    Mux2_5 extend5_mux(instr[10 : 6], idRsDataOut[4 : 0], ext5MuxSel,
ext5MuxOut);

    IdStall stall_controller(clk, rst, op, func, rs, rt, rfRena1, rfRena2,
        exeRfWaddr, exeRfWena, exeHiWena, exeLoWena,
        memRfWaddr, memRfWena, memHiWena, memLoWena, stall);

    Extend #(.WIDTH(5)) sa_ext(ext5MuxOut, 1'b0, idShamt);

    Extend #(.WIDTH(16)) imm_ext(ext16DataIn, ext16Sign, ext16DataOut);

    Extend #(.WIDTH(18)) ext18_b_pc(ext18DataIn, 1'b1, ext18DataOut);

    BranchCompare compare(clk, rst, idRsDataOut, idRtDataOut, op, func,
cp0Exception, isBranch);
```

```
    RegFile cpu_regfile(clk, rst, rfWena, rs, rt, rfRena1, rfRena2, rfWaddr,
rfWdata, idRsDataOut, idRtDataOut, reg28, reg29);

    Cp0 cpu_cp0(clk, rst, mfc0, mtc0, pc4 - 4, cp0Addr, idRtDataOut,
cp0Exception, eret, cp0Cause, idCp0Out, cp0Status, idCp0Pc);

    Operation operation_unit(isBranch, instr, op, func, cp0Status, idRfWena,
idHiWena, idLoWena, idDmemWena,
        rfRena1, rfRena2, idClzEna, idMulEna, idDivEna, idDmemEna, idDmemWCs,
idDmemRCs, ext16Sign, idCutterSign,
        idMulSign, idDivSign, idAluc, rd, mfc0, mtc0, eret, cp0Exception,
cp0Addr, cp0Cause, ext5MuxSel, idCutterMuxSel,
        idAluMux1Sel, idAluMux2Sel, idHiMuxSel, idLoMuxSel, idCutterSel,
idRfMuxSel, idPcMuxSel);

endmodule



module PipeIdExeReg(
    input clk,
    input rst,
    input wena,
    input [31 : 0] idPc4,       // ID 阶段的 PC
    input [31 : 0] idRsDataOut,
    input [31 : 0] idRtDataOut,
    input [31 : 0] idImm,       // ID 阶段的立即数
    input [31 : 0] idShamt,
    input [31 : 0] idCp0Out,    // ID 阶段的 CP0 输出
    input [31 : 0] idHiOut,     // ID 阶段的 HI 寄存器输出
    input [31 : 0] idLoOut,     // ID 阶段的 LO 寄存器输出
    input [4 : 0] idRfWaddr,
    input idClzEna,
    input idMulEna,
    input idDivEna,
    input idDmemEna,
    input idMulSign,
    input idDivSign,
    input idCutterSign,
    input [3 : 0] idAluc,       // ID 阶段的 ALUC
    input idRfWena,
    input idHiWena,
    input idLoWena,
    input idDmemWena,
    input [1 : 0] idDmemWCs,
    input [1 : 0] idDmemRCs,
    input stall,
    input idCutterMuxSel,
    input idAluMux1Sel,
    input [1 : 0] idAluMux2Sel,
    input [1 : 0] idHiMuxSel,
    input [1 : 0] idLoMuxSel,
    input [2 : 0] idCutterSel,
    input [2 : 0] idRfMuxSel,
    output reg [31 : 0] exePc4,
    output reg [31 : 0] exeRsDataOut,
    output reg [31 : 0] exeRtDataOut,
    output reg [31 : 0] exeImm,
    output reg [31 : 0] exeShamt,
```

```
    output reg [31 : 0] exeCp0Out,
    output reg [31 : 0] exeHiOut,
    output reg [31 : 0] exeLoOut,
    output reg [4 : 0] exeRfWaddr,
    output reg exeClzEna,
    output reg exeMulEna,
    output reg exeDivEna,
    output reg exeDmemEna,
    output reg exeMulSign,
    output reg exeDivSign,
    output reg exeCutterSign,
    output reg [3 : 0] exeAluc,
    output reg exeRfWena,
    output reg exeHiWena,
    output reg exeLoWena,
    output reg exeDmemWena,
    output reg [1 : 0] exeDmemWCs,
    output reg [1 : 0] exeDmemRCs,
    output reg exeAluMux1Sel,
    output reg [1 : 0] exeAluMux2Sel,
    output reg exeCutterMuxSel,
    output reg [1 : 0] exeHiMuxSel,
    output reg [1 : 0] exeLoMuxSel,
    output reg [2 : 0] exeCutterSel,
    output reg [2 : 0] exeRfMuxSel
    );

    always @ (posedge clk or posedge rst) begin
        if(rst == `RST_ENABLED || stall == `STOP) begin
            exeImm <= 32'b0;
            exeShamt <= 32'b0;
            exeCp0Out <= 32'b0;
            exeHiOut <= 32'b0;
            exeLoOut <= 32'b0;
            exeRfWaddr <= 5'b0;
            exeClzEna <= 1'b0;
            exeMulEna <= 1'b0;
            exeDivEna <= 1'b0;
            exeDmemEna <= 1'b0;
            exeMulSign <= 1'b0;
            exeDivSign <= 1'b0;
            exeCutterSign <= 1'b0;
            exeAluc <= 4'b0;
            exeRfWena <= 1'b0;
            exeHiWena <= 1'b0;
            exeLoWena <= 1'b0;
            exeDmemWena <= 1'b0;
            exeDmemWCs <= 2'b0;
            exeDmemRCs <= 2'b0;
            exeAluMux1Sel <= 1'b0;
            exeAluMux2Sel <= 1'b0;
            exeCutterMuxSel <= 1'b0;
            exeHiMuxSel <= 2'b0;
            exeLoMuxSel <= 2'b0;
            exeCutterSel <= 3'b0;
            exeRfMuxSel <= 3'b0;
            exePc4 <= 32'b0;
            exeRsDataOut <= 32'b0;
            exeRtDataOut <= 32'b0;
        end
```

```
        else if(wena == `WRITE_ENABLED) begin
            exeImm <= idImm;
            exeShamt <= idShamt;
            exeCp0Out <= idCp0Out;
            exeHiOut <= idHiOut;
            exeLoOut <= idLoOut;
            exeRfWaddr <= idRfWaddr;
            exeDmemEna <= idDmemEna;
            exeMulSign <= idMulSign;
            exeDivSign <= idDivSign;
            exeCutterSign <= idCutterSign;
            exeDmemWena <= idDmemWena;
            exeDmemWCs <= idDmemWCs;
            exeDmemRCs <= idDmemRCs;
            exeAluMux1Sel <= idAluMux1Sel;
            exeAluMux2Sel <= idAluMux2Sel;
            exeCutterMuxSel <= idCutterMuxSel;
            exeHiMuxSel <= idHiMuxSel;
            exeLoMuxSel <= idLoMuxSel;
            exeCutterSel <= idCutterSel;
            exeRfMuxSel <= idRfMuxSel;
            exePc4 <= idPc4;
            exeRsDataOut <= idRsDataOut;
            exeRtDataOut <= idRtDataOut;
            exeAluc <= idAluc;
            exeRfWena <= idRfWena;
            exeHiWena <= idHiWena;
            exeLoWena <= idLoWena;
            exeClzEna <= idClzEna;
            exeMulEna <= idMulEna;
            exeDivEna <= idDivEna;
        end
    end

endmodule


module PipeExe(
    input rst,
    input [31 : 0] pc4,
    input [31 : 0] rsDataOut,
    input [31 : 0] rtDataOut,
    input [31 : 0] imm,
    input [31 : 0] shamt,
    input [31 : 0] cp0Out,
    input [31 : 0] hiOut,
    input [31 : 0] loOut,
    input [4 : 0] rfWaddr,
    input [3 : 0] aluc,
    input mulEna,
    input divEna,
    input clzEna,
    input dmemEna,
    input mulSign,
    input divSign,
    input cutterSign,
    input rfWena,
    input hiWena,
```

```
    input loWena,
    input dmemWena,
    input [1 : 0] dmemWCs,
    input [1 : 0] dmemRCs,
    input cutterMuxSel,
    input aluMux1Sel,
    input [1 : 0] aluMux2Sel,
    input [1 : 0] hiMuxSel,
    input [1 : 0] loMuxSel,
    input [2 : 0] cutterSel,
    input [2 : 0] rfMuxSel,
    output [31 : 0] exeMulHi,
    output [31 : 0] exeMulLo,
    output [31 : 0] exeDivR,
    output [31 : 0] exeDivQ,
    output [31 : 0] exeClzOut,
    output [31 : 0] exeAluOut,
    output [31 : 0] exePc4,
    output [31 : 0] exeRsDataOut,
    output [31 : 0] exeRtDataOut,
    output [31 : 0] exeCp0Out,
    output [31 : 0] exeHiOut,
    output [31 : 0] exeLoOut,
    output [4 : 0] exeRfWaddr,
    output exeDmemEna,
    output exeRfWena,
    output exeHiWena,
    output exeLoWena,
    output exeDmemWena,
    output [1 : 0] exeDmemWCs,
    output [1 : 0] exeDmemRCs,
    output exeCutterSign,
    output exeCutterMuxSel,
    output [2 : 0] exeCutterSel,
    output [1 : 0] exeHiMuxSel,
    output [1 : 0] exeLoMuxSel,
    output [2 : 0] exeRfMuxSel
    );

    wire zero;
    wire carry;
    wire negative;
    wire overflow;
    wire [31 : 0] aluIn1;
    wire [31 : 0] aluIn2;

    assign exeDmemRCs = dmemRCs;
    assign exeDmemWCs = dmemWCs;
    assign exePc4 = pc4;
    assign exeRsDataOut = rsDataOut;
    assign exeRtDataOut = rtDataOut;
    assign exeHiOut = hiOut;
    assign exeLoOut = loOut;
    assign exeHiMuxSel = hiMuxSel;
    assign exeLoMuxSel = loMuxSel;
    assign exeRfMuxSel = rfMuxSel;
    assign exeRfWaddr = rfWaddr;
    assign exeCp0Out = cp0Out;
    assign exeDmemEna = dmemEna;
    assign exeRfWena = rfWena;
```

```
    assign exeHiWena = hiWena;
    assign exeLoWena = loWena;
    assign exeDmemWena = dmemWena;
    assign exeCutterSign = cutterSign;
    assign exeCutterMuxSel = cutterMuxSel;
    assign exeCutterSel = cutterSel;

    Div cpu_div(rst, divEna, divSign, rsDataOut, rtDataOut, exeDivQ, exeDivR);
    Mult cpu_mult(rst, mulEna, mulSign, rsDataOut, rtDataOut, exeMulHi,
exeMulLo);
    Counter cpu_counter(rsDataOut, clzEna, exeClzOut);
    Mux2_32 alu_mux1(shamt, rsDataOut, aluMux1Sel, aluIn1);
    Mux4_32 alu_mux2(rtDataOut, imm, 32'b0, 32'b0, aluMux2Sel, aluIn2);
    Alu cpu_alu(aluIn1, aluIn2, aluc, exeAluOut, zero, carry, negative,
overflow);

endmodule



module PipeExeMemReg(
    input clk,
    input rst,
    input wena,
    input [31 : 0] exeMulHi,
    input [31 : 0] exeMulLo,
    input [31 : 0] exeDivR,
    input [31 : 0] exeDivQ,
    input [31 : 0] exeClzOut,
    input [31 : 0] exeAluOut,
    input [31 : 0] exePc4,
    input [31 : 0] exeRsDataOut,
    input [31 : 0] exeRtDataOut,
    input [31 : 0] exeCp0Out,
    input [31 : 0] exeHiOut,
    input [31 : 0] exeLoOut,
    input [4 : 0] exeRfWaddr,
    input exeDmemEna,
    input exeCutterSign,
    input exeRfWena,
    input exeHiWena,
    input exeLoWena,
    input exeDmemWena,
    input [1 : 0] exeDmemWCs,
    input [1 : 0] exeDmemRCs,
    input exeCutterMuxSel,
    input [1 : 0] exeHiMuxSel,
    input [1 : 0] exeLoMuxSel,
    input [2 : 0] exeCutterSel,
    input [2 : 0] exeRfMuxSel,
    output reg [31 : 0] memMulHi,
    output reg [31 : 0] memMulLo,
    output reg [31 : 0] memDivR,
    output reg [31 : 0] memDivQ,
    output reg [31 : 0] memClzOut,
    output reg [31 : 0] memAluOut,
    output reg [31 : 0] memPc4,
    output reg [31 : 0] memRsDataOut,
    output reg [31 : 0] memRtDataOut,
    output reg [31 : 0] memCp0Out,
```

```
    output reg [31 : 0] memHiOut,
    output reg [31 : 0] memLoOut,
    output reg [4 : 0] memRfWaddr,
    output reg memDmemEna,
    output reg memCutterSign,
    output reg memRfWena,
    output reg memHiWena,
    output reg memLoWena,
    output reg memDmemWena,
    output reg [1 : 0] memDmemWCs,
    output reg [1 : 0] memDmemRCs,
    output reg memCutterMuxSel,
    output reg [1 : 0] memHiMuxSel,
    output reg [1 : 0] memLoMuxSel,
    output reg [2 : 0] memCutterSel,
    output reg [2 : 0] memRfMuxSel
    );

    always @ (posedge clk or posedge rst)
    begin
        if (rst == `RST_ENABLED) begin
            memHiMuxSel <= 0;
            memLoMuxSel <= 0;
            memCutterSel <= 0;
            memRfMuxSel <= 0;
            memHiOut <= 0;
            memLoOut <= 0;
            memRfWaddr <= 0;
            memDmemEna <= 0;
            memCutterSign <= 0;
            memMulHi <= 0;
            memMulLo <= 0;
            memDivR <= 0;
            memDivQ <= 0;
            memClzOut <= 0;
            memAluOut <= 0;
            memPc4 <= 0;
            memRsDataOut <= 0;
            memRtDataOut <= 0;
            memCp0Out <= 0;
            memRfWena <= 0;
            memHiWena <= 0;
            memLoWena <= 0;
            memDmemWena <= 0;
            memDmemWCs <= 0;
            memDmemRCs <= 0;
            memCutterMuxSel <= 0;
        end else if (wena == `WRITE_ENABLED) begin
            memHiOut <= exeHiOut;
            memLoOut <= exeLoOut;
            memRfWaddr <= exeRfWaddr;
            memMulHi <= exeMulHi;
            memMulLo <= exeMulLo;
            memDivR <= exeDivR;
            memDivQ <= exeDivQ;
            memClzOut <= exeClzOut;
            memAluOut <= exeAluOut;
            memPc4 <= exePc4;
            memRsDataOut <= exeRsDataOut;
            memRtDataOut <= exeRtDataOut;
```

```
            memCp0Out <= exeCp0Out;
            memHiMuxSel <= exeHiMuxSel;
            memLoMuxSel <= exeLoMuxSel;
            memCutterSel <= exeCutterSel;
            memDmemWCs <= exeDmemWCs;
            memDmemRCs <= exeDmemRCs;
            memCutterMuxSel <= exeCutterMuxSel;
            memRfMuxSel <= exeRfMuxSel;
            memDmemEna <= exeDmemEna;
            memCutterSign <= exeCutterSign;
            memRfWena <= exeRfWena;
            memHiWena <= exeHiWena;
            memLoWena <= exeLoWena;
            memDmemWena <= exeDmemWena;
        end
    end

endmodule


module PipeMem(
    input clk,
    input [31 : 0] mulHi,
    input [31 : 0] mulLo,
    input [31 : 0] divR,
    input [31 : 0] divQ,
    input [31 : 0] clzOut,
    input [31 : 0] aluOut,
    input [31 : 0] pc4,
    input [31 : 0] rsDataOut,
    input [31 : 0] rtDataOut,
    input [31 : 0] cp0Out,
    input [31 : 0] hiOut,
    input [31 : 0] loOut,
    input [4 : 0] rfWaddr,
    input dmemEna,
    input rfWena,
    input hiWena,
    input loWena,
    input dmemWena,
    input cutterSign,
    input [1 : 0] dmemWCs,
    input [1 : 0] dmemRCs,
    input cutterMuxSel,
    input [2 : 0] cutterSel,
    input [1 : 0] hiMuxSel,
    input [1 : 0] loMuxSel,
    input [2 : 0] rfMuxSel,
    output [31 : 0] memMulHi,
    output [31 : 0] memMulLo,
    output [31 : 0] memDivR,
    output [31 : 0] memDivQ,
    output [31 : 0] memClzOut,
    output [31 : 0] memAluOut,
    output [31 : 0] memDmemOut,
    output [31 : 0] memPc4,
    output [31 : 0] memRsDataOut,
    output [31 : 0] memCp0Out,
    output [31 : 0] memHiOut,
```

```
    output [31 : 0] memLoOut,
    output [4 : 0] memRfWaddr,
    output memRfWena,
    output memHiWena,
    output memLoWena,
    output [1 : 0] memHiMuxSel,
    output [1 : 0] memLoMuxSel,
    output [2 : 0] memRfMuxSel
    );

    wire [31 : 0] cutterMuxOut;
    wire [31 : 0] dmemOut;

    assign memDivQ = divQ;
    assign memDivR = divR;
    assign memClzOut = clzOut;
    assign memAluOut = aluOut;
    assign memPc4 = pc4;
    assign memRsDataOut = rsDataOut;
    assign memHiOut = hiOut;
    assign memLoOut = loOut;
    assign memCp0Out = cp0Out;
    assign memRfWaddr = rfWaddr;
    assign memRfWena = rfWena;
    assign memHiWena = hiWena;
    assign memLoWena = loWena;
    assign memHiMuxSel = hiMuxSel;
    assign memLoMuxSel = loMuxSel;
    assign memRfMuxSel = rfMuxSel;
    assign memMulHi = mulHi;
    assign memMulLo = mulLo;

    Cut cpuCutter(cutterMuxOut, cutterSel, cutterSign, memDmemOut);
    Mux2_32 cutterMux(rtDataOut, dmemOut, cutterMuxSel, cutterMuxOut);
    Dmem cpuDmem(clk, dmemEna, dmemWena, dmemWCs, dmemRCs, memDmemOut, aluOut,
dmemOut);

endmodule


module PipeMemWbReg (
    input clk,
    input rst,
    input wena,
    input [31 : 0] memMulHi,
    input [31 : 0] memMulLo,
    input [31 : 0] memDivR,
    input [31 : 0] memDivQ,
    input [31 : 0] memClzOut,
    input [31 : 0] memAluOut,
    input [31 : 0] memDmemOut,
    input [31 : 0] memPc4,
    input [31 : 0] memRsDataOut,
    input [31 : 0] memCp0Out,
    input [31 : 0] memHiOut,
    input [31 : 0] memLoOut,
    input [4 : 0] memRfWaddr,
    input memRfWena,
    input memHiWena,
```

```
    input memLoWena,
    input [1 : 0] memHiMuxSel,
    input [1 : 0] memLoMuxSel,
    input [2 : 0] memRfMuxSel,
    output reg [31 : 0] wbMulHi,
    output reg [31 : 0] wbMulLo,
    output reg [31 : 0] wbDivR,
    output reg [31 : 0] wbDivQ,
    output reg [31 : 0] wbClzOut,
    output reg [31 : 0] wbAluOut,
    output reg [31 : 0] wbDmemOut,
    output reg [31 : 0] wbPc4,
    output reg [31 : 0] wbRsDataOut,
    output reg [31 : 0] wbCp0Out,
    output reg [31 : 0] wbHiOut,
    output reg [31 : 0] wbLoOut,
    output reg [4 : 0] wbRfWaddr,
    output reg wbRfWena,
    output reg wbHiWena,
    output reg wbLoWena,
    output reg [1 : 0] wbHiMuxSel,
    output reg [1 : 0] wbLoMuxSel,
    output reg [2 : 0] wbRfMuxSel
    );

    always @ (posedge clk or posedge rst)
    begin
        if(rst == `RST_ENABLED) begin
            wbPc4 <= 0;
            wbRsDataOut <= 0;
            wbCp0Out <= 0;
            wbHiOut <= 0;
            wbLoOut <= 0;
            wbRfWaddr <= 0;
            wbRfWena <= 0;
            wbHiWena <= 0;
            wbLoWena <= 0;
            wbHiMuxSel <= 0;
            wbLoMuxSel <= 0;
            wbRfMuxSel <= 0;
            wbMulHi <= 0;
            wbMulLo <= 0;
            wbDivR <= 0;
            wbDivQ <= 0;
            wbClzOut <= 0;
            wbAluOut <= 0;
            wbDmemOut <= 0;
        end else if(wena == `WRITE_ENABLED) begin
            wbDivR <= memDivR;
            wbDivQ <= memDivQ;
            wbPc4 <= memPc4;
            wbRsDataOut <= memRsDataOut;
            wbCp0Out <= memCp0Out;
            wbHiOut <= memHiOut;
            wbLoOut <= memLoOut;
            wbRfWaddr <= memRfWaddr;
            wbRfWena <= memRfWena;
            wbHiWena <= memHiWena;
            wbLoWena <= memLoWena;
            wbMulHi <= memMulHi;
```

```
            wbMulLo <= memMulLo;
            wbClzOut <= memClzOut;
            wbAluOut <= memAluOut;
            wbDmemOut <= memDmemOut;
            wbHiMuxSel <= memHiMuxSel;
            wbLoMuxSel <= memLoMuxSel;
            wbRfMuxSel <= memRfMuxSel;
        end
    end

endmodule



module PipeWb(
    input [31 : 0] mulHi,
    input [31 : 0] mulLo,
    input [31 : 0] divR,
    input [31 : 0] divQ,
    input [31 : 0] clzOut,
    input [31 : 0] aluOut,
    input [31 : 0] dmemOut,
    input [31 : 0] pc4,
    input [31 : 0] rsDataOut,
    input [31 : 0] cp0Out,
    input [31 : 0] hiOut,
    input [31 : 0] loOut,
    input [4 : 0] rfWaddr,
    input rfWena,
    input hiWena,
    input loWena,
    input [1 : 0] hiMuxSel,
    input [1 : 0] loMuxSel,
    input [2 : 0] rfMuxSel,
    output [31 : 0] hiWdata,
    output [31 : 0] loWdata,
    output [31 : 0] rfWdata,
    output [4 : 0] wbRfWaddr,
    output wbRfWena,
    output wbHiWena,
    output wbLoWena
    );

    assign wbRfWaddr = rfWaddr;
    assign wbLoWena = loWena;
    assign wbRfWena = rfWena;
    assign wbHiWena = hiWena;

    Mux4_32 mux_lo(divQ, mulLo, rsDataOut, 32'b0, loMuxSel, loWdata);
    Mux4_32 mux_hi(divR, mulHi, rsDataOut, 32'h0, hiMuxSel, hiWdata);
    Mux8_32 mux_rf(loOut, pc4, clzOut, cp0Out, dmemOut, aluOut, hiOut, mulLo,
rfMuxSel, rfWdata);

endmodule
```

```
`timescale 1ns / 1ps

module pipe_top(
```

```
    input clk,
    input rst,
    input switchRes,
    output [7:0] oSeg,
    output [7:0] oSel
    );

    reg [30:0] count;
    reg clk1s;

    wire [31:0] instruction;
    wire [31:0] pc;
    wire [31:0] reg28;
    wire [31:0] reg29;
    wire [31:0] seg7In;

    always @ (posedge clk or posedge rst)
    begin
        if(rst) begin
            clk1s <= 0;
            count <= 0;
        end else if(count == 99999) begin
            count <= 0;
            clk1s <= ~clk1s;
        end else
            count <= count + 1;
    end

    Cpu cpu(clk, rst, instruction, pc, reg28, reg29);
    Mux2_32 resMux(reg28, reg29, switchRes, seg7In);
    Seg7x16 seg7(clk, rst, 1, seg7In, oSeg, oSel);

endmodule
```

```
`timescale 1ns / 1ps

module Seg7x16 (
    input clk,
    input reset,
    input cs,
    input [31 : 0] iData,
    output [7 : 0] oSeg,
    output [7 : 0] oSel
    );

    reg [14 : 0] cnt;
    always @ (posedge clk, posedge reset)
      if (reset)
        cnt <= 0;
      else
        cnt <= cnt + 1'b1;
    wire seg7Clk = cnt[14];

    reg [2 : 0] seg7Addr;
    always @ (posedge seg7Clk, posedge reset)
        if (reset)
          seg7Addr <= 0;
        else
```

```
        seg7Addr <= seg7Addr + 1'b1;
    reg [7 : 0] oSelR;
    always @ (*)
    case(seg7Addr)
        7 : oSelR = 8'b01111111;
        6 : oSelR = 8'b10111111;
        5 : oSelR = 8'b11011111;
        4 : oSelR = 8'b11101111;
        3 : oSelR = 8'b11110111;
        2 : oSelR = 8'b11111011;
        1 : oSelR = 8'b11111101;
        0 : oSelR = 8'b11111110;
    endcase

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

    reg [7 : 0] segDataR;
    always @ (*)
    case(seg7Addr)
        0 : segDataR = iDataStore[3 : 0];
        1 : segDataR = iDataStore[7 : 4];
        2 : segDataR = iDataStore[11 : 8];
        3 : segDataR = iDataStore[15 : 12];
        4 : segDataR = iDataStore[19 : 16];
        5 : segDataR = iDataStore[23 : 20];
        6 : segDataR = iDataStore[27 : 24];
        7 : segDataR = iDataStore[31 : 28];
    endcase

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

    assign oSel = oSelR;
    assign oSeg = oSegR;
```

```
endmodule
```

```
`timescale 1ns / 1ps

module Add32 (
    input [31 : 0] a,
    input [31 : 0] b,
    output [31 : 0] z
    );
    assign z = a + b;
endmodule
```

```
`timescale 1ns / 1ps

module Alu (
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output [31:0] r,
    output zero,
    output carry,
    output negative,
    output overflow
    );

    reg [32 : 0] result;
    reg same_signal;
    reg flag;

    always @ (*) begin
        case(aluc)
            4'b0000: result = a + b;
            4'b0010:
            begin
                result = $signed(a) + $signed(b);
                same_signal = ~(a[31] ^ b[31]);
                flag = (same_signal && result[31] != a[31]) ? 1 : 0;
            end
            4'b0001: result = a - b;
            4'b0011:
            begin
                result = $signed(a) - $signed(b);
                same_signal = ~(a[31] ^ b[31]);
                flag = (~same_signal && result[31] != a[31]) ? 1 : 0;
            end

            4'b0100: result = a & b;
            4'b0101: result = a | b;
            4'b0110: result = a ^ b;
            4'b0111: result = ~(a | b);

            4'b1000: result = {b[15 : 0], 16'b0};
            4'b1001: result = {b[15 : 0], 16'b0};

            4'b1011: result = ($signed(a) < $signed(b)) ? 1 : 0;
```

```
            4'b1010: result =  (a < b) ? 1 : 0;

            4'b1100: result = $signed(b) >>> a;
            4'b1110: result = b << a;
            4'b1111: result = b << a;
            4'b1101: result = b >> a;
        endcase
    end

    assign r = result[31 : 0];
    assign carry = (aluc == 4'b0000 | aluc == 4'b0001 | aluc == 4'b1010 | aluc == 
4'b1100 | aluc == 4'b1101 | aluc == 4'b1110) ? result[32] : 1'bz;
    assign zero = (r == 32'b0) ? 1 : 0;
    assign negative = result[31];
    assign overflow = (aluc == 4'b0010 | aluc == 4'b0011) ? flag : 1'bz;

endmodule
```

```
`timescale 1ns / 1ps

module BranchCompare(
    input clk,
    input rst,
    input [31 : 0] in1,
    input [31 : 0] in2,
    input [5 : 0] op,
    input [5 : 0] func,
    input exception,
    output reg isBranch
    );

    always @ (*) begin
        if (rst == `RST_ENABLED)
            isBranch <= 1'b0;
        else if (op == `BGEZ_OP)
            isBranch <= (in1 >= 0) ? 1'b1 : 1'b0;
        else if (op == `BEQ_OP)
            isBranch <= (in1 == in2) ? 1'b1 : 1'b0;
        else if (op == `TEQ_OP && func == `TEQ_FUNC)
            isBranch <= (in1 == in2) ? 1'b1 : 1'b0;
        else if (op == `BNE_OP)
            isBranch <= (in1 != in2) ? 1'b1 : 1'b0;
        else if (op == `J_OP)
            isBranch <= 1'b1;
        else if (op == `JR_OP && func == `JR_FUNC)
            isBranch <= 1'b1;
        else if (op == `JAL_OP)
            isBranch <= 1'b1;
        else if (op == `JALR_OP && func == `JALR_FUNC)
            isBranch <= 1'b1;
        else if (exception)
            isBranch <= 1'b1;
        else
            isBranch <= 1'b0;
    end

   endmodule
```

```
`timescale 1ns / 1ps

module Operation (
    input isBranch,
    input [31:0] instruction,
    input [5:0] Op,
    input [5:0] Func,
    input [31:0] status,
    output rfWena,
    output hiWena,
    output loWena,
    output dmemWena,
    output rfRena1,
    output rfRena2,
    output clzEna,
    output mulEna,
    output divEna,
    output dmemEna,
    output [1:0] dmemWCs,
    output [1:0] dmemRCs,
    output ext16Sign,
    output cutterSign,
    output mulSign,
    output divSign,
    output [3:0] aluc,
    output [4:0] rd,
    output mfc0,
    output mtc0,
    output eret,
    output exception,
    output [4:0] cp0Addr,
    output [4:0] cause,
    output ext5MuxSel,
    output cutterMuxSel,
    output aluMux1Sel,
    output [1:0] aluMux2Sel,
    output [1:0] hiMuxSel,
    output [1:0] loMuxSel,
    output [2:0] cutterSel,
    output [2:0] rfMuxSel,
    output [2:0] pcMuxSel
    );

    wire Addi = (Op == 6'b001000);
    wire Addiu = (Op == 6'b001001);
    wire Andi = (Op == 6'b001100);
    wire Ori = (Op == 6'b001101);
    wire Sltiu = (Op == 6'b001011);
    wire Div = (Op == 6'b000000 && Func == 6'b011010);
    wire Lui = (Op == 6'b001111);
    wire Xori = (Op == 6'b001110);
    wire Slti = (Op == 6'b001010);
    wire Addu = (Op == 6'b000000 && Func==6'b100001);
    wire And = (Op == 6'b000000 && Func == 6'b100100);
    wire Break = (Op == 6'b000000 && Func == 6'b001101);
    wire Beq = (Op == 6'b000100);
```

```
    wire Bne = (Op == 6'b000101);
    wire J = (Op == 6'b000010);
    wire Jal = (Op == 6'b000011);
    wire Bgez = (Op == 6'b000001);
    wire Jr = (Op == 6'b000000 && Func == 6'b001000);
    wire Lw = (Op == 6'b100011);
    wire Xor = (Op == 6'b000000 && Func == 6'b100110);
    wire Nor = (Op == 6'b000000 && Func == 6'b100111);
    wire Or = (Op == 6'b000000 && Func == 6'b100101);
    wire Teq = (Op == 6'b000000 && Func == 6'b110100);
    wire Sll = (Op == 6'b000000 && Func == 6'b000000);
    wire Sllv = (Op == 6'b000000 && Func == 6'b000100);
    wire Sltu = (Op == 6'b000000 && Func == 6'b101011);
    wire Sra = (Op == 6'b000000 && Func == 6'b000011);
    wire Srl = (Op == 6'b000000 && Func == 6'b000010);
    wire Subu = (Op == 6'b000000 && Func == 6'b100011);
    wire Syscall = (Op == 6'b000000 && Func== 6'b001100);
    wire Sw = (Op == 6'b101011);
    wire Add = (Op == 6'b000000 && Func == 6'b100000);
    wire Sub = (Op == 6'b000000 && Func == 6'b100010);
    wire Slt = (Op == 6'b000000 && Func == 6'b101010);
    wire Srlv = (Op == 6'b000000 && Func == 6'b000110);
    wire Srav = (Op == 6'b000000 && Func == 6'b000111);
    wire Multu = (Op == 6'b000000 && Func == 6'b011001);
    wire Clz = (Op == 6'b011100 && Func == 6'b100000);
    wire Divu = (Op == 6'b000000 && Func == 6'b011011);
    wire Eret = (Op == 6'b010000 && Func == 6'b011000);
    wire Jalr = (Op == 6'b000000 && Func == 6'b001001);
    wire Mul = (Op == 6'b011100 && Func == 6'b000010);
    wire Lb = (Op == 6'b100000);
    wire Lbu = (Op == 6'b100100);
    wire Lhu = (Op == 6'b100101);
    wire Sb = (Op == 6'b101000);
    wire Mtlo = (Op == 6'b000000 && Func == 6'b010011);
    wire Sh = (Op == 6'b101001);
    wire Lh = (Op == 6'b100001);
    wire Mfc0 = (instruction[31:21] == 11'b01000000000 &&
instruction[10:3]==8'b00000000);
    wire Mfhi = (Op == 6'b000000 && Func == 6'b010000);
    wire Mthi = (Op == 6'b000000 && Func == 6'b010001);
    wire Mflo = (Op == 6'b000000 && Func == 6'b010010);
    wire Mtc0 = (instruction[31:21] == 11'b01000000100 &&
instruction[10:3]==8'b00000000);

    // 使能信号
    assign rfRena1 = Addi + Addiu + Andi + Ori + Sltiu + Xori + Slti + Addu + And
+ Beq + Bne + Jr + Lw + Xor + Nor + Or + Sllv + Sltu + Subu + Sw + Add + Sub +
Slt + Srlv + Srav + Clz + Divu + Jalr + Lb + Lbu + Lhu + Sb + Sh + Lh + Mul +
Multu + Teq + Div;
    assign rfRena2 = Addu + And + Beq + Bne + Xor + Nor + Or + Sll + Sllv + Sltu
+ Sra + Srl + Subu + Sw + Add + Sub + Slt + Srlv + Srav + Divu + Sb + Sh + Mtc0 +
Mul + Multu + Teq + Div;
    assign hiWena = Div + Divu + Multu + Mthi + Mul;
    assign loWena = Div + Divu + Multu + Mtlo + Mul;
    assign rfWena = Addi + Addiu + Andi + Ori + Sltiu + Lui + Xori + Slti + Addu
+ And + Xor + Nor + Or + Sll + Sllv + Sltu + Sra + Srl + Subu + Add + Sub + Slt +
Srlv + Srav + Lb + Lbu + Lh + Lhu + Lw + Mfc0 + Clz + Jal + Jalr + Mfhi + Mflo +
Mul;
    assign clzEna = Clz;
```

```
    assign mulEna = Mul + Multu;
    assign divEna = Div + Divu;

    // 数据寄存器
    assign dmemWena = Sb + Sh + Sw;
    assign dmemEna = Lw + Sw + Sb + Sh + Lb + Lh + Lhu + Lbu;
    assign dmemWCs[1] = Sh + Sb;
    assign dmemWCs[0] = Sw + Sb;
    assign dmemRCs[1] = Lh + Lb + Lhu + Lbu;
    assign dmemRCs[0] = Lw + Lb + Lbu;

    // 符号信号
    assign cutterSign = Lb + Lh;
    assign mulSign = Mul;
    assign divSign = Div;
    assign ext16Sign = Addi + Addiu + Sltiu + Slti;

    // ext5 选择信号
    assign ext5MuxSel = Sllv + Srav + Srlv;

    // alu 选择信号
    assign aluMux1Sel = ~(Sll + Srl + Sra + Div + Divu + Mul + Multu + J + Jr +
Jal + Jalr + Mfc0 + Mtc0 + Mfhi + Mflo + Mthi + Mtlo + Clz + Eret + Syscall +
Break);
    assign aluMux2Sel[1] = Bgez;
    assign aluMux2Sel[0] = Slti + Sltiu + Addi + Addiu + Andi + Ori + Xori + Lb +
Lbu + Lh + Lhu + Lw + Sb + Sh + Sw + Lui;

    // 运算器
    assign aluc[3] = Slt + Sltu + Sllv + Srlv + Srav + Lui + Srl + Sra + Slti +
Sltiu + Sll;
    assign aluc[2] = And + Or + Xor + Nor + Sll + Srl + Sra + Sllv + Srlv + Srav
+ Andi + Ori + Xori;
    assign aluc[1] = Add + Sub + Xor + Nor + Slt + Sltu + Sll + Sllv + Addi +
Xori + Beq + Bne + Slti + Sltiu + Bgez + Teq;
    assign aluc[0] = Subu + Sub + Or + Nor + Slt + Sllv + Srlv + Sll + Srl + Slti
+ Ori + Beq + Bne + Bgez + Teq;

    // cutter 选择信号
    assign cutterMuxSel = ~(Sb + Sh + Sw);
    assign cutterSel[2] = Sh;
    assign cutterSel[1] = Lb + Lbu + Sb;
    assign cutterSel[0] = Lh + Lhu + Sb;

    // regfiles 选择信号
    assign rfMuxSel[2] = ~(Beq + Bne + Bgez + Div + Divu + Sb + Multu + Sh + Sw +
J + Jr + Jal + Jalr + Mfc0 + Mtc0 + Mflo + Mthi + Mtlo + Clz + Eret + Syscall +
Teq + Break);
    assign rfMuxSel[1] = Mul + Mfc0 + Mtc0 + Clz + Mfhi;
    assign rfMuxSel[0] = ~(Beq + Bne + Bgez + Div + Divu + Multu + Lb + Lbu + Lh
+ Lhu + Lw + Sb + Sh + Sw + J + Mtc0 + Mfhi + Mflo + Mthi + Mtlo + Clz + Eret +
Syscall + Teq + Break);

    // pc 选择信号
    assign pcMuxSel[2] = Eret + (Beq&&isBranch) + (Bne&&isBranch) +
(Bgez&&isBranch);
    assign pcMuxSel[1] = ~(J + Jr + Jal + Jalr + pcMuxSel[2]);
    assign pcMuxSel[0] = Eret + exception + Jr + Jalr;
```

```
    // hi 选择信号
    assign hiMuxSel[1] = Mthi;
    assign hiMuxSel[0] = Mul + Multu;

    // lo 选择信号
    assign loMuxSel[1] = Mtlo;
    assign loMuxSel[0] = Mul + Multu;

    // rd 信号
    assign rd = (Add + Addu + Sub + Subu + And + Or + Xor + Nor + Slt + Sltu +
Sll + Srl + Sra + Sllv + Srlv + Srav + Clz + Jalr + Mfhi + Mflo + Mul) ?
                    instruction[15:11] : (( Addi + Addiu + Andi + Ori + Xori + Lb
+ Lbu + Lh + Lhu + Lw + Slti + Sltiu + Lui + Mfc0) ? instruction[20:16] : (Jal ?
5'd31:5'b0));

    assign mfc0 = Mfc0;
    assign mtc0 = Mtc0;
    assign cp0Addr = instruction[15:11];
    assign exception = status[0] && ((Syscall && status[1]) || (Break &&
status[2]) || (Teq && status[3]));
    assign eret = Eret;
    assign cause = Break ? 5'b01001 : (Syscall ? 5'b01000 : (Teq ? 5'b01101 :
5'b00000));

endmodule
```

```
`timescale 1ns / 1ps

module Cp0(
    input clk,
    input rst,
    input mfc0,
    input mtc0,
    input [31 : 0] pc,
    input [4 : 0] addr,
    input [31 : 0] wdata,
    input exception,
    input eret,
    input [4 : 0] cause,
    output [31 : 0] rdata,
    output [31 : 0] status,
    output [31 : 0] epc_out
    );

    reg [31 : 0] cp0_register [31 : 0];
    reg [31 : 0] temp;
    integer i;

    always @ (posedge clk or posedge rst)
    begin
        if (rst == `RST_ENABLED) begin
            for(i = 0; i < 32; i = i + 1) begin
                if(i == `STATUS)
                    cp0_register[i] <= 32'h0000000f;
                else
```

```
                    cp0_register[i] <= 0;
            end
            temp <= 0;
        end else begin if(mtc0)
                cp0_register[addr] <= wdata;
            else if(exception) begin
                cp0_register[`EPC] = pc;
                temp = cp0_register[`STATUS];
                cp0_register[`STATUS] = cp0_register[`STATUS] << 5;
                cp0_register[`CAUSE] = {25'b0, cause, 2'b0};
            end else if(eret)
                cp0_register[`STATUS] = temp;
        end
    end

    assign status = cp0_register[`STATUS];
    assign epc_out = eret ? cp0_register[`EPC] : `EXCEPTION_ADDR;
    assign rdata = mfc0 ? cp0_register[addr] : 32'h0;

endmodule
```

```
`timescale 1ns / 1ps

module Cpu(
    input clk,
    input rst,
    output [31 : 0] instr,
    output [31 : 0] pc,
    output [31 : 0] reg28,
    output [31 : 0] reg29
    );

    // IF 输入
    wire [31 : 0] ifPcI;
    wire [31 : 0] ifCp0PcI;
    wire [31 : 0] ifBPcI;
    wire [31 : 0] ifRPcI;
    wire [31 : 0] ifJPcI;
    wire [2 : 0] ifPcMuxSelI;

    //IF 输出
    wire [31 : 0] ifNpcO;
    wire [31 : 0] ifPc4O;
    wire [31 : 0] ifInstructionO;

    wire stall;
    wire isBranch;
    wire pcWena;

    //ID 输入
    wire [31 : 0] idPc4I;
    wire [31 : 0] idInstructionI;
    wire [31 : 0] idRfWdataI;
    wire [31 : 0] idHiWdataI;
    wire [31 : 0] idLoWdataI;
    wire [4 : 0] idRfWaddrI;
    wire idRfWenaI;
```

```
    wire idHiWenaI;
    wire idLoWenaI;

    // ID 输出
    wire [31 : 0] idCp0PcO;
    wire [31 : 0] idRPcO;
    wire [31 : 0] idBPcO;
    wire [31 : 0] idJPcO;
    wire [31 : 0] idRsDataOutO;
    wire [31 : 0] idRtDataOutO;
    wire [31 : 0] idImmO;
    wire [31 : 0] idShamtO;
    wire [31 : 0] idPc4O;
    wire [31 : 0] idCp0OutO;
    wire [31 : 0] idHiOutO;
    wire [31 : 0] idLoOutO;
    wire [4 : 0]  idRfWaddrO;
    wire [3 : 0] idAlucO;
    wire idMulSignO;
    wire idDivSignO;
    wire idCutterSignO;
    wire idClzEnaO;
    wire idMulEnaO;
    wire idDivEnaO;
    wire idDmemEnaO;
    wire idHiWenaO;
    wire idLoWenaO;
    wire idRfWenaO;
    wire idDmemWenaO;
    wire [1 : 0] idDmemWCsO;
    wire [1 : 0] idDmemRCsO;
    wire idCutterMuxSelO;
    wire idAluMux1SelO;
    wire [1 : 0] idAluMux2SelO;
    wire [1 : 0] idHiMuxSelO;
    wire [1 : 0] idLoMuxSelO;
    wire [2 : 0] idCutterSelO;
    wire [2 : 0] idRfMuxSelO;
    wire [2 : 0] idPcMuxSelO;

    // EXE 输入
    wire [31 : 0] exePc4I;
    wire [31 : 0] exeImmI;
    wire [31 : 0] exeShamtI;
    wire [31 : 0] exeRsDataOutI;
    wire [31 : 0] exeRtDataOutI;
    wire [31 : 0] exeCp0OutI;
    wire [31 : 0] exeHiOutI;
    wire [31 : 0] exeLoOutI;
    wire [4 : 0] exeRfWaddrI;
    wire [3 : 0] exeAlucI;
    wire exeMulEnaI;
    wire exeDivEnaI;
    wire exeClzEnaI;
    wire exeDmemEnaI;
    wire [1 : 0] exeDmemWCsI;
    wire [1 : 0] exeDmemRCsI;
    wire exeMulSignI;
    wire exeDivSignI;
```

```
    wire exeCutterSignI;
    wire exeRfWenaI;
    wire exeHiWenaI;
    wire exeLoWenaI;
    wire exeDmemWenaI;
    wire exeCutterMuxSelI;
    wire exeAluMux1SelI;
    wire [1 : 0] exeAluMux2SelI;
    wire [2 : 0] exeCutterSelI;
    wire [1 : 0] exeHiMuxSelI;
    wire [1 : 0] exeLoMuxSelI;
    wire [2 : 0] exeRfMuxSelI;

    // EXE 输出
    wire [31 : 0] exeMulHiO;
    wire [31 : 0] exeMulLoO;
    wire [31 : 0] exeDivRO;
    wire [31 : 0] exeDivQO;
    wire [31 : 0] exeClzOutO;
    wire [31 : 0] exeAluOutO;
    wire [31 : 0] exePc4O;
    wire [31 : 0] exeRsDataOutO;
    wire [31 : 0] exeRtDataOutO;
    wire [31 : 0] exeCp0OutO;
    wire [31 : 0] exeHiOutO;
    wire [31 : 0] exeLoOutO;
    wire [4 : 0] exeRfWaddrO;
    wire exeDmemEnaO;
    wire exeCutterSignO;
    wire exeRfWenaO;
    wire exeHiWenaO;
    wire exeLoWenaO;
    wire exeDmemWenaO;
    wire [1 : 0] exeDmemWCsO;
    wire [1 : 0] exeDmemRCsO;
    wire exeCutterMuxSelO;
    wire [2 : 0] exeCutterSelO;
    wire [1 : 0] exeHiMuxSelO;
    wire [1 : 0] exeLoMuxSelO;
    wire [2 : 0] exeRfMuxSelO;

    // MEM 输入
    wire [31 : 0] memMulHiI;
    wire [31 : 0] memMulLoI;
    wire [31 : 0] memDivRI;
    wire [31 : 0] memDivQI;
    wire [31 : 0] memClzOutI;
    wire [31 : 0] memAluOutI;
    wire [31 : 0] memPc4I;
    wire [31 : 0] memRsDataOutI;
    wire [31 : 0] memRtDataOutI;
    wire [31 : 0] memCp0OutI;
    wire [31 : 0] memHiOutI;
    wire [31 : 0] memLoOutI;
    wire [4 : 0] memRfWaddrI;
    wire memDmemEnaI;
    wire memRfWenaI;
    wire memHiWenaI;
    wire memLoWenaI;
```

```
    wire memDmemWenaI;
    wire memCutterSignI;
    wire [1 : 0] memDmemWCsI;
    wire [1 : 0] memDmemRCsI;
    wire memCutterMuxSelI;
    wire [2 : 0] memCutterSelI;
    wire [1 : 0] memHiMuxSelI;
    wire [1 : 0] memLoMuxSelI;
    wire [2 : 0] memRfMuxSelI;

    // MEM 输出
    wire [31 : 0] memMulHiO;
    wire [31 : 0] memMulLoO;
    wire [31 : 0] memDivRO;
    wire [31 : 0] memDivQO;
    wire [31 : 0] memClzOutO;
    wire [31 : 0] memAluOutO;
    wire [31 : 0] memDmemOutO;
    wire [31 : 0] memPc4O;
    wire [31 : 0] memRsDataOutO;
    wire [31 : 0] memCp0OutO;
    wire [31 : 0] memHiOutO;
    wire [31 : 0] memLoOutO;
    wire [4 : 0] memRfWaddrO;
    wire memRfWenaO;
    wire memHiWenaO;
    wire memLoWenaO;
    wire [1 : 0] memHiMuxSelO;
    wire [1 : 0] memLoMuxSelO;
    wire [2 : 0] memRfMuxSelO;

    // WB 输入
    wire [31 : 0] wbMulHiI;
    wire [31 : 0] wbMulLoI;
    wire [31 : 0] wbDivRI;
    wire [31 : 0] wbDivQI;
    wire [31 : 0] wbClzOutI;
    wire [31 : 0] wbAluOutI;
    wire [31 : 0] wbDmemOutI;
    wire [31 : 0] wbPc4I;
    wire [31 : 0] wbRsDataOutI;
    wire [31 : 0] wbCp0OutI;
    wire [31 : 0] wbHiOutI;
    wire [31 : 0] wbLoOutI;
    wire [4 : 0] wbRfWaddrI;
    wire wbRfWenaI;
    wire wbHiWenaI;
    wire wbLoWenaI;
    wire [1 :0] wbHiMuxSelI;
    wire [1 : 0] wbLoMuxSelI;
    wire [2 : 0] wbRfMuxSelI;

    // WB 输出
    wire [31 : 0] wbHiWdataO;
    wire [31 : 0] wbLoWdataO;
    wire [31 : 0] wbRfWdataO;
    wire [4 : 0] wbRfWaddrO;
    wire wbRfWenaO;
    wire wbHiWenaO;
```

```
    wire wbLoWenaO;

    wire idExeWena;
    wire exeMemWena;
    wire memWbWena;

    assign instr = ifInstructionO;
    assign pc = ifPcI;
    assign pcWena = 1'b1;
    assign idExeWena = 1'b1;
    assign exeMemWena = 1'b1;
    assign memWbWena = 1'b1;

    assign ifPcMuxSelI = idPcMuxSelO;
    assign ifCp0PcI = idCp0PcO;
    assign ifBPcI = idBPcO;
    assign ifRPcI = idRPcO;
    assign ifJPcI = idJPcO;

    assign idRfWdataI = wbRfWdataO;
    assign idHiWdataI = wbHiWdataO;
    assign idLoWdataI = wbLoWdataO;
    assign idRfWaddrI = wbRfWaddrO;
    assign idRfWenaI = wbRfWenaO;
    assign idHiWenaI = wbHiWenaO;
    assign idLoWenaI = wbLoWenaO;

    PcReg programCounter(clk, rst, pcWena, stall, ifNpcO, ifPcI);

    PipeIf instrFetch(ifPcI, ifCp0PcI, ifBPcI, ifRPcI, ifJPcI, ifPcMuxSelI,
ifNpcO, ifPc4O, ifInstructionO);

    PipeIfIdReg ifIdReg(clk, rst, ifPc4O, ifInstructionO, stall, isBranch,
idPc4I, idInstructionI);

    PipeId instruction_decoder(clk, rst, idPc4I, idInstructionI, idRfWdataI,
idHiWdataI, idLoWdataI, idRfWaddrI, idRfWenaI, idHiWenaI, idLoWenaI,
        exeRfWaddrO, exeRfWenaO, exeHiWenaO, exeLoWenaO,
        memRfWaddrO, memRfWenaO, memHiWenaO, memLoWenaO,
        idCp0PcO, idRPcO, idBPcO, idJPcO, idRsDataOutO, idRtDataOutO, idImmO,
idShamtO, idPc4O,
        idCp0OutO, idHiOutO, idLoOutO, idRfWaddrO, idAlucO, idMulSignO,
idDivSignO, idCutterSignO, idClzEnaO,
        idMulEnaO, idDivEnaO, idDmemEnaO, idHiWenaO, idLoWenaO, idRfWenaO,
idDmemWenaO, idDmemWCsO, idDmemRCsO,
        idCutterMuxSelO, idAluMux1SelO, idAluMux2SelO, idHiMuxSelO, idLoMuxSelO,
idCutterSelO, idRfMuxSelO, idPcMuxSelO,
        stall, isBranch, reg28, reg29);

    PipeIdExeReg idExeReg(clk, rst, idExeWena, idPc4O, idRsDataOutO,
idRtDataOutO, idImmO, idShamtO, idCp0OutO, idHiOutO,
        idLoOutO, idRfWaddrO, idClzEnaO, idMulEnaO, idDivEnaO, idDmemEnaO,
idMulSignO, idDivSignO, idCutterSignO,
        idAlucO, idRfWenaO, idHiWenaO, idLoWenaO, idDmemWenaO, idDmemWCsO,
idDmemRCsO, stall,
        idCutterMuxSelO, idAluMux1SelO, idAluMux2SelO, idHiMuxSelO, idLoMuxSelO,
idCutterSelO, idRfMuxSelO,
        exePc4I, exeRsDataOutI, exeRtDataOutI, exeImmI, exeShamtI, exeCp0OutI,
exeHiOutI, exeLoOutI, exeRfWaddrI,
        exeClzEnaI, exeMulEnaI, exeDivEnaI, exeDmemEnaI, exeMulSignI,
```

```
exeDivSignI, exeCutterSignI, exeAlucI,
        exeRfWenaI, exeHiWenaI, exeLoWenaI, exeDmemWenaI, exeDmemWCsI,
exeDmemRCsI, exeAluMux1SelI, exeAluMux2SelI,
        exeCutterMuxSelI, exeHiMuxSelI, exeLoMuxSelI, exeCutterSelI,
exeRfMuxSelI);

    //57
    PipeExe executor(rst, exePc4I, exeRsDataOutI, exeRtDataOutI, exeImmI,
exeShamtI, exeCp0OutI, exeHiOutI, exeLoOutI,
        exeRfWaddrI, exeAlucI, exeMulEnaI, exeDivEnaI, exeClzEnaI, exeDmemEnaI,
exeMulSignI, exeDivSignI, exeCutterSignI,
        exeRfWenaI, exeHiWenaI, exeLoWenaI, exeDmemWenaI, exeDmemWCsI,
exeDmemRCsI, exeCutterMuxSelI, exeAluMux1SelI,
        exeAluMux2SelI, exeHiMuxSelI, exeLoMuxSelI, exeCutterSelI, exeRfMuxSelI,
        exeMulHiO, exeMulLoO, exeDivRO, exeDivQO, exeClzOutO, exeAluOutO,
exePc4O, exeRsDataOutO, exeRtDataOutO,
        exeCp0OutO, exeHiOutO, exeLoOutO, exeRfWaddrO,
        exeDmemEnaO, exeRfWenaO, exeHiWenaO, exeLoWenaO, exeDmemWenaO,
exeDmemWCsO, exeDmemRCsO, exeCutterSignO,
        exeCutterMuxSelO, exeCutterSelO, exeHiMuxSelO, exeLoMuxSelO,
exeRfMuxSelO);

    //55
    PipeExeMemReg exeMemReg(clk, rst, exeMemWena, exeMulHiO, exeMulLoO, exeDivRO,
exeDivQO, exeClzOutO, exeAluOutO,
        exePc4O, exeRsDataOutO, exeRtDataOutO, exeCp0OutO, exeHiOutO, exeLoOutO,
exeRfWaddrO, exeDmemEnaO, exeCutterSignO,
        exeRfWenaO, exeHiWenaO, exeLoWenaO, exeDmemWenaO, exeDmemWCsO,
exeDmemRCsO, exeCutterMuxSelO, exeHiMuxSelO,
        exeLoMuxSelO, exeCutterSelO, exeRfMuxSelO, memMulHiI, memMulLoI,
memDivRI, memDivQI, memClzOutI, memAluOutI,
        memPc4I, memRsDataOutI, memRtDataOutI, memCp0OutI, memHiOutI, memLoOutI,
memRfWaddrI, memDmemEnaI, memCutterSignI,
        memRfWenaI, memHiWenaI, memLoWenaI, memDmemWenaI, memDmemWCsI,
memDmemRCsI, memCutterMuxSelI, memHiMuxSelI,
        memLoMuxSelI, memCutterSelI, memRfMuxSelI);

    //46
    PipeMem memory(clk, memMulHiI, memMulLoI, memDivRI, memDivQI, memClzOutI,
memAluOutI, memPc4I, memRsDataOutI,
        memRtDataOutI, memCp0OutI, memHiOutI, memLoOutI, memRfWaddrI,
memDmemEnaI, memRfWenaI, memHiWenaI, memLoWenaI,
        memDmemWenaI, memCutterSignI, memDmemWCsI, memDmemRCsI, memCutterMuxSelI,
memCutterSelI, memHiMuxSelI,
        memLoMuxSelI, memRfMuxSelI, memMulHiO, memMulLoO, memDivRO, memDivQO,
memClzOutO, memAluOutO, memDmemOutO,
        memPc4O, memRsDataOutO, memCp0OutO, memHiOutO, memLoOutO, memRfWaddrO,
memRfWenaO, memHiWenaO, memLoWenaO,
        memHiMuxSelO, memLoMuxSelO, memRfMuxSelO);

    //41
    PipeMemWbReg memWbReg(clk, rst, memWbWena, memMulHiO, memMulLoO, memDivRO,
memDivQO, memClzOutO,
        memAluOutO, memDmemOutO, memPc4O, memRsDataOutO, memCp0OutO, memHiOutO,
memLoOutO, memRfWaddrO,
        memRfWenaO, memHiWenaO, memLoWenaO, memHiMuxSelO, memLoMuxSelO,
memRfMuxSelO, wbMulHiI, wbMulLoI,
        wbDivRI, wbDivQI, wbClzOutI, wbAluOutI, wbDmemOutI, wbPc4I, wbRsDataOutI,
wbCp0OutI, wbHiOutI,
        wbLoOutI, wbRfWaddrI, wbRfWenaI, wbHiWenaI, wbLoWenaI, wbHiMuxSelI,
```

```
wbLoMuxSelI, wbRfMuxSelI);

    //26
    PipeWb writeBack(wbMulHiI, wbMulLoI, wbDivQI, wbDivQI, wbClzOutI, wbAluOutI,
wbDmemOutI, wbPc4I,
        wbRsDataOutI, wbCp0OutI, wbHiOutI, wbLoOutI, wbRfWaddrI, wbRfWenaI,
wbHiWenaI, wbLoWenaI, wbHiMuxSelI,
        wbLoMuxSelI, wbRfMuxSelI, wbHiWdataO, wbLoWdataO, wbRfWdataO, wbRfWaddrO,
wbRfWenaO, wbHiWenaO, wbLoWenaO);

endmodule
```

```
`timescale 1ns / 1ps

module cutter(
    input [31:0] data_in,
    input [2:0] sel,
    input sign,
    output [31:0] data_out
    );

    reg [31:0] temp;

    always @ (*) begin
        case(sel)
            3'b001: temp <= {(sign && data_in[15]) ? 16'hffff : 16'h0000,
data_in[15:0]};
            3'b010: temp <= {(sign && data_in[7]) ? 24'hffffff : 24'h000000,
data_in[7:0]};
            3'b011: temp <= {24'h000000, data_in[7:0]};
            3'b100: temp <= {16'h0000, data_in[15:0]};
            default: temp <= data_in;
        endcase
    end

    assign data_out = temp;

endmodule



//数据存储器
module Dmem (
    input clk,
    input ena,
    input wena,
    input [1 : 0] w_cs,
    input [1 : 0] r_cs,
    input [31 : 0] data_in,
    input [31 : 0] addr,
    output reg [31 : 0] data_out
    );

    reg [31 : 0] Buffer[2047 : 0];

    wire [1 : 0] low_addr;
    wire [9 : 0] high_addr;
```

```
    assign low_addr = (addr - 32'h10010000) % 4;
    assign high_addr = (addr - 32'h10010000) / 4;

    //写操作
    always @ (posedge clk) begin
        if (ena == `ENABLED) begin
            if (wena == `WRITE_ENABLED) begin
            case (w_cs)
                2'b01 : Buffer[high_addr] <= data_in;
                2'b10 :
                    begin
                        if (low_addr == 2'b00)
                            Buffer[high_addr][15 : 0] <= data_in[15 : 0];
                        else if (low_addr == 2'b10)
                            Buffer[high_addr][31 : 16] <= data_in[15 : 0];
                    end
                2'b11 :
                    begin
                        if (low_addr == 2'b00)
                            Buffer[high_addr][7 : 0] <= data_in[7 : 0];
                        else if (low_addr == 2'b01)
                            Buffer[high_addr][15 : 8] <= data_in[7 : 0];
                        else if (low_addr == 2'b10)
                            Buffer[high_addr][23 : 16] <= data_in[7 : 0];
                        else if (low_addr == 2'b11)
                            Buffer[high_addr][31 : 24] <= data_in[7 : 0];
                    end
                default : ;
            endcase
            end
        end
    end

    //读操作
    always @ (*) begin
        if (ena == `ENABLED && wena == `WRITE_DISABLED) begin
        case (r_cs)
            2'b01 : data_out <= Buffer[high_addr];
            2'b10 :
                begin
                    case(low_addr)
                        2'b00 : data_out <= Buffer[high_addr][15 : 0];
                        2'b10 : data_out <= Buffer[high_addr][31 : 16];
                        default : data_out <= 32'bx;
                    endcase
                end
            2'b11 :
                begin
                    case(low_addr)
                        2'b00 : data_out <= Buffer[high_addr][7 : 0];
                        2'b01 : data_out <= Buffer[high_addr][15 : 8];
                        2'b10 : data_out <= Buffer[high_addr][23 : 16];
                        2'b11 : data_out <= Buffer[high_addr][31 : 24];
                        default : data_out <= 32'bx;
                    endcase
                end
        endcase
        end
    end
```

```
endmodule

//改变前缀
module Cut (
    input [31 : 0] in,
    input [2 : 0] sel,
    input sign,
    output reg [31 : 0] out
    );

    always @ (*) begin
        case (sel)
            3'b010: out <= {(sign && in[7]) ? 24'hffffff : 24'h000000, in[7 :
0]};
            3'b011: out <= {24'h000000, in[7 : 0]};
            3'b001: out <= {(sign && in[15]) ? 16'hffff : 16'h0000, in[15 : 0]};
            3'b100: out <= {16'h0000, in[15 : 0]};
            default: out <= in;
        endcase
    end

endmodule
```

```
`timescale 1ns / 1ps

module Mult (
    input reset,
    input ena,
    input sign,
    input [31 : 0] a,
    input [31 : 0] b,
    output [31 : 0] Hi,
    output [31 : 0] Lo
    );

    reg [63 : 0] stored;
    reg [63 : 0] temp;
    reg [31 : 0] tempA;
    reg [31 : 0] tempB;
    reg isMinus;
    integer i;

    assign Lo = (ena == `ENABLED) ? stored[31 : 0] : 32'b0;
    assign Hi = (ena == `ENABLED) ? stored[63 : 32] : 32'b0;

    always @ (*)
    begin
        if (reset == `RST_ENABLED) begin
            tempA <= 0;
            tempB <= 0;
            stored <= 0;
            isMinus <= 0;
        end else if(ena == `ENABLED) begin
            if (a == 0 || b == 0) begin
                stored <= 0;
            end else if(sign == `UNSIGNED) begin
                stored = 0;
                for (i = 0; i < 32; i = i + 1) begin
```

```
                    temp = b[i] ? ({32'b0, a} << i) : 64'b0;
                    stored = stored + temp;
                end
            end else begin
                stored = 0;
                isMinus = a[31] ^ b[31];
                tempA = a;
                tempB = b;
                if (a[31] == 1) begin
                    tempA = a ^ 32'hffffffff;
                    tempA = tempA + 1;
                end
                if (b[31] == 1) begin
                    tempB = b ^ 32'hffffffff;
                    tempB = tempB + 1;
                end
                for (i = 0; i < 32; i = i + 1) begin
                    temp = tempB[i] ? ({32'b0, tempA} << i) : 64'b0;
                    stored = stored + temp;
                end
                if (isMinus == 1) begin
                    stored = stored ^ 64'hffffffffffffffff;
                    stored = stored + 1;
                end
            end
        end
    end

    endmodule
```

```
`timescale 1ns / 1ps

module Div(
    input reset,
    input ena,
    input sign,
    input [31 : 0] dividend,
    input [31 : 0] divisor,
    output [31 : 0] q,
    output [31 : 0] r
    );

    reg isMinus;
    reg isDivMinus;
    reg [63 : 0] tempDividend;
    reg [63 : 0] tempDivisor;
    integer counter;

    assign q = (ena == `CHIP_ENABLED) ? tempDividend[31 : 0] : 32'b0;
    assign r = (ena == `CHIP_ENABLED) ? tempDividend[63 : 32]: 32'b0;

    always @ (*)
    begin
        if (reset == `RST_ENABLED) begin
            tempDividend <= 0;
            tempDivisor <= 0;
            isMinus <= 0;
            isDivMinus <= 0;
```

```
        end else if(ena == `ENABLED) begin
            if (sign == `UNSIGNED) begin
                tempDividend = dividend;
                tempDivisor = {divisor, 32'b0};
                for (counter = 0; counter < 32; counter = counter + 1) begin
                    tempDividend = tempDividend << 1;
                    if (tempDividend >= tempDivisor) begin
                        tempDividend = tempDividend - tempDivisor;
                        tempDividend = tempDividend + 1;
                    end
                end
                counter = 0;
            end else begin
                tempDividend = dividend;
                tempDivisor = {divisor, 32'b0};
                isMinus = dividend[31] ^ divisor[31];
                isDivMinus = dividend[31];
                if (dividend[31] == 1) begin
                    tempDividend = dividend ^ 32'hffffffff;
                    tempDividend = tempDividend + 1;
                end
                if (divisor[31] == 1) begin
                    tempDivisor = {divisor ^ 32'hffffffff, 32'b0};
                    tempDivisor = tempDivisor + 64'h0000000100000000;
                end
                for (counter = 0; counter < 32; counter = counter + 1) begin
                    tempDividend = tempDividend << 1;
                    if (tempDividend >= tempDivisor) begin
                        tempDividend = tempDividend - tempDivisor;
                        tempDividend = tempDividend + 1;
                    end
                end
                if (isDivMinus == 1) begin
                    tempDividend = tempDividend ^ 64'hffffffff00000000;
                    tempDividend = tempDividend + 64'h0000000100000000;
                end
                if (isMinus == 1) begin
                    tempDividend = tempDividend ^ 64'h00000000ffffffff;
                    tempDividend = tempDividend + 64'h0000000000000001;
                    if (tempDividend[31:0] == 32'b0) begin
                        tempDividend = tempDividend - 64'h0000000100000000;
                    end
                end
            end
        end
    end

    endmodule
```

```
`timescale 1ns / 1ps

//扩展
module Extend #(parameter WIDTH = 5)(
    input [WIDTH - 1 : 0] a,
    input sign, //1 有符号 0 无符号
    output [31 : 0] z
    );
```

```
    assign z = sign ? ({{(32 - WIDTH){a[WIDTH - 1]}}, a}) : ({{(32 -
WIDTH){1'b0}}, a});
endmodule
```

```
`timescale 1ns / 1ps

module Mux2_5 (
    input [4 : 0] a,
    input [4 : 0] b,
    input sign,
    output reg [4 : 0] z
    );

    always @ (*) begin
        case (sign)
            1'b0: z <= a;
            1'b1: z <= b;
        endcase
    end

endmodule

module Mux2_32 (
    input [31 : 0] a,
    input [31 : 0] b,
    input sign,
    output reg [31 : 0] z
    );

    always @ (*) begin
        case (sign)
            1'b0: z <= a;
            1'b1: z <= b;
        endcase
    end

endmodule

module Mux4_32 (
    input [31 : 0] a,
    input [31 : 0] b,
    input [31 : 0] c,
    input [31 : 0] d,
    input [1 : 0] sign,
    output reg [31:0] z
    );

    always @ (*) begin
        case (sign)
            2'b00: z <= a;
            2'b01: z <= b;
            2'b10: z <= c;
            2'b11: z <= d;
            default: z <= 31'bz;
        endcase
   end
```

```
endmodule

module Mux8_32 (
    input [31 : 0] a,
    input [31 : 0] b,
    input [31 : 0] c,
    input [31 : 0] d,
    input [31 : 0] e,
    input [31 : 0] f,
    input [31 : 0] g,
    input [31 : 0] h,
    input [2 : 0]  sign,
    output reg [31 : 0] z
    );

    always @ (*) begin
        case (sign)
            3'b000: z <= a;
            3'b001: z <= b;
            3'b010: z <= c;
            3'b011: z <= d;
            3'b100: z <= e;
            3'b101: z <= f;
            3'b110: z <= g;
            3'b111: z <= h;
           default: z <= 32'bz;
        endcase
    end

endmodule
```

```
`timescale 1ns / 1ps

module IdStall(
    input clk,
    input rst,
    input [5 : 0] op,
    input [5 : 0] func,
    input [4 : 0] rs,
    input [4 : 0] rt,
    input rfRena1,
    input rfRena2,

    // EXE 数据
    input [4 : 0] exeRfWaddr,
    input exeRfWena,
    input exeHiWena,
    input exeLoWena,

    // MEM 数据
    input [4 : 0] memRfWaddr,
    input memRfWena,
    input memHiWena,
    input memLoWena,

    output reg stall
    );
```

```
    reg count;

    // ID、EXE、MEM 数据冲突检测
    always @ (negedge clk or posedge rst)
    begin
        if (rst == `RST_ENABLED) begin
            stall <= `RUN;
            count <= 0;
        end else if (count >= 1) begin
            count <= count - 1;
        end else if (stall == `STOP) begin
            stall <= `RUN;
        end else if (stall == `RUN) begin
            if (op == `MFHI_OP && func == `MFHI_FUNC) begin
                if (exeHiWena == `WRITE_ENABLED) begin
                    count <= 1;
                    stall <= `STOP;
                end else if (memHiWena == `WRITE_ENABLED) begin
                    stall <= `STOP;
                end
            end else if (op == `MFLO_OP && func == `MFLO_FUNC) begin
                if (exeLoWena == `WRITE_ENABLED) begin
                    count <= 1;
                    stall <= `STOP;
                end else if (memLoWena == `WRITE_ENABLED) begin
                    stall <= `STOP;
                end
            end else begin
                if (exeRfWena == `WRITE_ENABLED && rfRena1 == `READ_ENABLED &&
exeRfWaddr == rs) begin
                    count <= 1'b1;
                    stall <= `STOP;
                end else if (memRfWena == `WRITE_ENABLED && rfRena1 ==
`READ_ENABLED && memRfWaddr == rs) begin
                    stall <= `STOP;
                end
                if (exeRfWena == `WRITE_ENABLED && rfRena2 == `READ_ENABLED &&
exeRfWaddr == rt) begin
                    count <= 1'b1;
                    stall <= `STOP;
                end else if(memRfWena == `WRITE_ENABLED && rfRena2 ==
`READ_ENABLED && memRfWaddr == rt) begin
                    stall <= `STOP;
                end
            end
        end
    end

endmodule
```

```
`timescale 1ns / 1ps

module Counter(
    input [31 : 0] in,
    input ena,
    output reg [31 : 0] out
    );
```

```
    always @ (*) begin
        if (ena == `ENABLED) begin
            if (in[31])
                out <= 32'd0;
            else if (in[30])
                out <= 32'd1;
            else if (in[29])
                out <= 32'd2;
            else if (in[28])
                out <= 32'd3;
            else if (in[27])
                out <= 32'd4;
            else if (in[26])
                out <= 32'd5;
            else if (in[25])
                out <= 32'd6;
            else if (in[24])
                out <= 32'd7;
            else if (in[23])
                out <= 32'd8;
            else if (in[22])
                out <= 32'd9;
            else if (in[21])
                out <= 32'd10;
            else if (in[20])
                out <= 32'd11;
            else if (in[19])
                out <= 32'd12;
            else if (in[18])
                out <= 32'd13;
            else if (in[17])
                out <= 32'd14;
            else if (in[16])
                out <= 32'd15;
            else if (in[15])
                out <= 32'd16;
            else if (in[14])
                out <= 32'd17;
            else if (in[13])
                out <= 32'd18;
            else if (in[12])
                out <= 32'd19;
            else if (in[11])
                out <= 32'd20;
            else if (in[10])
                out <= 32'd21;
            else if (in[9])
                out <= 32'd22;
            else if (in[8])
                out <= 32'd23;
            else if (in[7])
                out <= 32'd24;
            else if (in[6])
                out <= 32'd25;
            else if (in[5])
                out <= 32'd26;
            else if (in[4])
                out <= 32'd27;
            else if (in[3])
```

```
                out <= 32'd28;
            else if (in[2])
                out <= 32'd29;
            else if (in[1])
                out <= 32'd30;
            else if (in[0])
                out <= 32'd31;
            else if (in[0] == 1'b0)
                out <= 32'd32;
            else
                out <= 32'bx;
        end
    end

endmodule
```