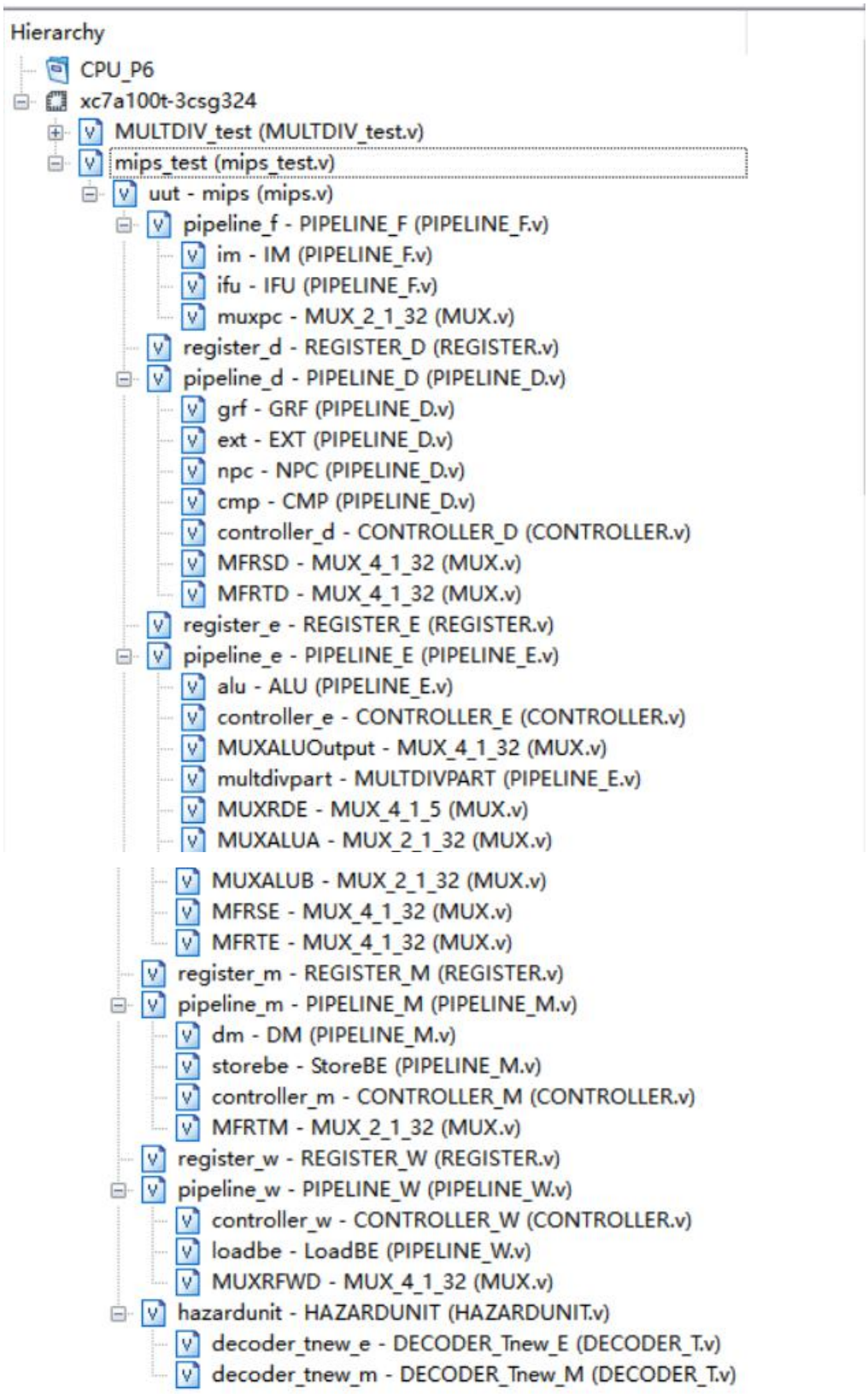
Project6 Verilog 完成流水线 CPU 开发

一、顶层设计



```
module mips(
    input clk,
    input reset
     );
     wire Stall, PCSel, RegWrite_W, RegWrite_D, RegWrite_E, RegWrite_M, ForwardRTM, Busy, Start;
     wire [1:0] ForwardRSD, ForwardRTD, ForwardRSE, ForwardRTE;
     wire [4:0] RD_W,RS_E, RT_E, RD_E, WriteRd_E,RD_M;
     wire [31:0] NPCOut, Instr_F, PCPlus4_F, Instr_D, PCPlus4_D, MUXRFWDOut, ALUOutput_M, EXTOut, RSV_D, RTV_D,
                 Instr_E, PCPlus4_E, RSV_E, RTV_E, EXTOut_E, ALUOutput, WriteData_E, Instr_M, RTV_M,
                 PCPlus4_M, MemOut, Instr_W, ReadData_W, PCPlus4_W, ALUOutput_W;
////////////////////////////////////////////////////////////////////////////////
//F级部件
//IM 指令存储单元                 //Bclass  cal_r  cal_i  Jclass  load  store  jal
//IFU 取指令单元                  //Bclass  cal_r  cal_i  Jclass  load  store  jal
//MUXPC 选择下一条PC值的多选器    //Bclass  Jclass  jal


    PIPELINE_F pipeline_f(
    .CLK(clk),                      //输入来自MIPS模块的时钟信号，控制IFU的时钟周期                    /**/
    .Reset(reset),                  //输入来自MIPS模块的复位信号，控制IFU是否清空                      /**/
    .Stall(Stall),                  //输入来自冲突单元的暂停信号，控制IFU是否暂停
    .PCSel(PCSel),                  //输入来自D级部件（D级控制器）的下一条PC的选择信号，选择IFU下一个值
    .NPCOut(NPCOut),                //输入来自D级部件（D级NPC）下一条PC可能的值，输入来自D级的跳转指令的IFU的下一个值

    .Instr_F(Instr_F),              //输出F级的指令         To: D级寄存器
    .PCPlus4_F(PCPlus4_F)           //输出F级的PC+4         To: D级寄存器
    );

//F级部件产生指令和PC+4

////////////////////////////////////////////////////////////////////////////////
//D级寄存器

    REGISTER_D register_d(
    .Instr_F(Instr_F),              //输入来自F级部件（IM）的指令
    .PCPlus4_F(PCPlus4_F),          //输入来自F级部件（IFU）的PC+4
    .CLK(clk),                      //输入来自MIPS模块的时钟信号                                       /**/
    .Stall(Stall),                  //输入来自冲突单元的暂停信号
    .Reset(reset),                  //输入来自MIPS模块的复位信号                                       /**/

    .Instr_D(Instr_D),              //输出D级的指令      To:  D级部件 E级寄存器 冲突单元
    .PCPlus4_D(PCPlus4_D)           //输出D级的PC+4      To:  D级部件 E级寄存器
    );
//D级寄存器产生D级的指令和PC+4

////////////////////////////////////////////////////////////////////////////////
//D级部件
//GRF 寄存器堆                        //Bclass  cal_r  cal_i  load  store  jal
//EXT 符号扩展器                      //cal_i  load  store
//NPC PC计算选择器                    //Bclass  Jclass  jal
//CMP 比较器                          //Bclass
//CONTROLLER_D D级控制器              //Bclass  cal_r  cal_i  Jclass  load  store  jal
//MFRSD  转发指令在D级要读取的GRFRD1的值   //Bclass  cal_r  cal_i  load  store
//MFRTD  转发指令在D级要读取的GRFRD2的值   //Bclass  cal_r  load  store

    PIPELINE_D pipeline_d(
    .CLK(clk),                      //输入来自MIPS模块的时钟信号                                         /**/
    .Reset(reset),                  //输入来自MIPS模块的复位信号                                         /**/
    .RegWrite_W(RegWrite_W),        //输入来自W级寄存器的写入信号，控制是否写入寄存器堆
    .RD_W(RD_W),                    //输入来自W级寄存器的写入地址，决定写入的地址
    .Instr_D(Instr_D),              //输入来自D级寄存器的指令
    .MUXRFWDOut(MUXRFWDOut),        //输入来自W级部件（MUXRFWD）的写入数据，暨转发所需
    .PCPlus4_D(PCPlus4_D),          //输入来自D级寄存器的PC+4
    .ALUOutput_M(ALUOutput_M),      //输入来自M级寄存器的ALU输出，转发所需
    .ForwardRSD(ForwardRSD),        //输入来自冲突单元的RS转发控制信号
    .ForwardRTD(ForwardRTD),        //输入来自冲突单元的RT转发控制信号

    .PCPlus4_W(PCPlus4_W),          //输入来自W级寄存器的PC+4，是用来输出display的

    .EXTOut(EXTOut),                //输出立即数扩展结果          To:  E级寄存器
    .NPCOut(NPCOut),                //输出下一条可能的PC值        To:  F级部件
    .PCSel(PCSel),                  //输出PC选择信号              To:  F级部件
    .RegWrite_D(RegWrite_D),        //输出D级指令的写信号，在D级就提前产生写信号，之后在流水线中传输       To:  E级寄存器
    .RSV_D(RSV_D),                  //输出GRFRD1    To:  E级寄存器
    .RTV_D(RTV_D)                   //输出GRFRD2    To:  E级寄存器
    );
//D级部件产生 立即数扩展结果 跳转指令后PC的值 PC选择信号 D级指令写信号 寄存器两个读取的值

////////////////////////////////////////////////////////////////////////////////
//E级寄存器
    REGISTER_E register_e(
    .CLK(clk),                      //输入来自MIPS模块的时钟信号                                          /**/
    .Reset(reset),                  //输入来自MIPS模块的复位信号                                          /**/
    .FlushE(Stall),                 //输入来自冲突单元的暂停信号，实质上是清空信号，相当于插入NOP
    .Instr_D(Instr_D),              //输入来自D级寄存器的指令
    .PCPlus4_D(PCPlus4_D),          //输入来自D级寄存器的PC+4
    .RSV_D(RSV_D),                  //输入来自D级部件（MFRSD）的GRFRD1
    .RTV_D(RTV_D),                  //输入来自D级部件（MFRTD）的GRFRD2
    .EXTOut_D(EXTOut),              //输入来自D级部件（EXT）的扩展数
    .RegWrite_D(RegWrite_D),        //输入来自D级部件（CONTROLLER_D）的写信号
    .RS_D(Instr_D[25:21]),          //输入来自D级寄存器的指令的读寄存器地址1
    .RT_D(Instr_D[20:16]),          //输入来自D级寄存器的指令的读寄存器地址2
    .RD_D(Instr_D[15:11]),          //输入来自D级寄存器的指令的写寄存器地址

    .Instr_E(Instr_E),              //输出E级的指令       To:  E级部件 M级寄存器 冲突单元
    .PCPlus4_E(PCPlus4_E),          //输出E级的PC+4       To:  E级部件 M级寄存器
    .RSV_E(RSV_E),                  //输出E级的GRFRD1     To:  E级部件
    .RTV_E(RTV_E),                  //输出E级的GRFRD2     To:  E级部件      /*store类型要用这个值存储，但是得经过E级部件选择*/
    .EXTOut_E(EXTOut_E),            //输出E级的扩展数     To:  E级部件
    .RegWrite_E(RegWrite_E),        //输出E级的写入信号   To:  M级寄存器 冲突单元
```

```
    .RS_E(RS_E),                        //输出E级指令的读寄存器地址1    To:  没啥用，不管了，实际上它的作用在E级部件中被Instr_E代替了
    .RT_E(RT_E),                        //输出E级指令的读寄存器地址2    To:  没啥用，不管了，实际上它的作用在E级部件中被Instr_E代替了
    .RD_E(RD_E)                         //输出E级指令的写寄存器地址     To:  没啥用，不管了，实际上它的作用在E级部件中被Instr_E代替了
    );
////////////////////////////////////////////////////////////////////////////////
//E级部件
//ALU 算术逻辑运算单元                                                          //cal_r  cal_i  load  store
//CONTROLLER_E E级控制器                                                        //cal_r  cal_i  load  store  jal

//MUXALUOutput 选择ALU的输出，                                                  //cal_r  cal_i  load  store  jal
//用来区分一般的对寄存器进行算术操作的指令和对31号寄存器进行操作的指令的输出

//MUXRDE 选择真正的写入寄存器的地址，应该将其输出连接至冲突单元，不然E级的写地址会错误//cal_r  cal_i  load  store  jal
//MUXALUB 选择ALU的第二个操作数是寄存器数还是立即数                            //cal_r  cal_i  load  store
//MFRSE 选择ALU的第一个操作数的转发                                             //cal_r  cal_i  load  store
//MFRTE 选择ALU的第二个操作数的转发                                             //cal_r  cal_i  load  store

   PIPELINE_E pipeline_e(
    .CLK(clk),                          //输入时钟信号
    .Reset(reset),                      //输入复位信号
    .Instr_E(Instr_E),                  //输入来自E级寄存器的指令
    .RTV_E(RTV_E),                      //输入来自E级寄存器的GRFRD2
    .RSV_E(RSV_E),                      //输入来自E级寄存器的GRFRD1
    .EXTOut_E(EXTOut_E),                //输入来自E级寄存器的扩展数
    .MUXRFWDOut(MUXRFWDOut),            //输入来自W级部件（MUXRFWD）的写入数据，转发所需
    .ALUOutput_M(ALUOutput_M),          //输入来自M级寄存器的ALU输出，转发所需
    .ForwardRSE(ForwardRSE),            //输入来自冲突单元的RSE控制信号
    .ForwardRTE(ForwardRTE),            //输入来自冲突单元的RTE控制信号
    .PCPlus4_E(PCPlus4_E),              //输入来自E级寄存器的PC+4，为对31号寄存器进行jal类操作的数准备

    .Start(Start),                      //输出乘除模块启动信号      To:  冲突单元
    .Busy(Busy),                        //输出乘除模块繁忙信号      To:  冲突单元
    .ALUOutput(ALUOutput),              //输出ALU的运算结果         To:  M级寄存器
    .WriteRd_E(WriteRd_E),              //输出真正的E级的写入地址    To:  M级寄存器              /*真正的经过的选择后的写入地址在E级确定*/
    .WriteData_E(WriteData_E)           //输出要写入内存的数据，应该来自经过转发MFRTE选择器选择的数据    To:  M级寄存器
    );
//E级部件产生 ALU的运算结果 E级的真正的写入地址 E级的真正的写入数据

////////////////////////////////////////////////////////////////////////////////
//M级寄存器
   REGISTER_M register_m(
    .CLK(clk),                          //输入来自MIPS模块的时钟信号                                  /**/
    .Reset(reset),                      //输入来自MIPS模块的复位信号                                  /**/
    .Instr_E(Instr_E),                  //输入来自E级寄存器的指令
    .RTV_E(WriteData_E),                //输入来自E级部件（MFRTE）的内存写入数据              /**/     /*此处应为来自E级部件的输出，而不是来自E级寄存器的值*/
    .PCPlus4_E(PCPlus4_E),              //输入来自E级寄存器的PC+4
    .ALUOutput_E(ALUOutput),            //输入来自E级部件（MUXALUOutput）的ALU运算结果
    .RegWrite_E(RegWrite_E),            //输入来自E级寄存器的写信号
    .WriteRd_E(WriteRd_E),              //输入来自E级部件（MUXRDE）的真正的写寄存器地址

    .RegWrite_M(RegWrite_M),            //输出M级的写信号    To:  W级寄存器 冲突单元
    .Instr_M(Instr_M),                  //输出M级的指令      To:  M级部件 W级寄存器 冲突单元
    .RTV_M(RTV_M),                      //输出M级的写入数据  To:  M级部件
    .PCPlus4_M(PCPlus4_M),              //输出M级的PC+4      To:  M级部件 W级寄存器
    .ALUOutput_M(ALUOutput_M),          //输出M级的ALU运算结果  To:  D级部件 E级部件 M级部件 W级寄存器
    .RD_M(RD_M)                         //输出M级的写寄存器地址  To:  W级寄存器 冲突单元
    );
////////////////////////////////////////////////////////////////////////////////
//M级部件
//DM 内存单元                                   //load  store
//CONTROLLER_M M级控制器                        //load  store
//MFRTM 选择写入寄存器的值的转发                //store
//如果添加LH LB SH SB之类的指令，在此加两个MUX即可     //load store

   PIPELINE_M pipeline_m(
    .CLK(clk),                          //输入来自MIPS模块的时钟信号              /**/
    .Reset(reset),                      //输入来自MIPS模块的复位信号              /**/
    .ALUOutput_M(ALUOutput_M),          //输入来自M级寄存器的的ALU输出  作为存取内存的地址
    .RTV_M(RTV_M),                      //输入来自M级寄存器的写入数据，这也是经过E级转发选择后的数据
    .Instr_M(Instr_M),                  //输入来自M级寄存器的指令
    .ForwardRTM(ForwardRTM),            //输入来自冲突单元的RT转发控制信号
    .PCPlus4_M(PCPlus4_M),              //输入来自M级寄存器的PC+4  display所用
    .MUXRFWDOut(MUXRFWDOut),            //输入来自W级部件（MUXRFWD）写数据，转发所需

    .MemOut(MemOut)                     //输出内存读数据  To:  W级寄存器
    );
//M级部件产生 内存读数据

////////////////////////////////////////////////////////////////////////////////
//W级寄存器
   REGISTER_W register_w(
    .CLK(clk),                          //输入来自MIPS模块的时钟信号          /**/
    .Reset(reset),                      //输入来自MIPS模块的复位信号          /**/
    .Instr_M(Instr_M),                  //输入来自M级寄存器的指令
    .PCPlus4_M(PCPlus4_M),              //输入来自M级寄存器的PC+4
    .ALUOutput_M(ALUOutput_M),          //输入来自M级寄存器的ALU运算结果
    .ReadData_M(MemOut),                //输入来自M级部件（DM）的内存读数据                    /**/        /*此处输入MemOut*/
    .RD_M(RD_M),                        //输入来自M级寄存器的写寄存器地址
    .RegWrite_M(RegWrite_M),            //输入来自M级寄存器的写寄存器数据

    .Instr_W(Instr_W),                  //输出W级的指令          To:  W级部件 冲突单元
    .PCPlus4_W(PCPlus4_W),              //输出W级的PC+4          To:  W级部件
    .ALUOutput_W(ALUOutput_W),          //输出W级的ALU运算结果   To:  W级部件
    .ReadData_W(ReadData_W),            //输出W级的内存读数据    To:  W级部件
    .RD_W(RD_W),                        //输出W级的寄存器写地址  To:  D级部件 冲突单元
    .RegWrite_W(RegWrite_W)             //输出W级的寄存器写信号  To:  D级部件 冲突单元
    );
////////////////////////////////////////////////////////////////////////////////
//W级部件
```

```
//CONTROLLER W W级的控制器      //cal_r  cal_i  load  jal
//MUXRFWD 选择真正的写入数据     //cal_r  cal_i  load  jal
//GRF 理论上存在的写入寄存器堆

   PIPELINE_W pipeline_w(
   .Instr_W(Instr_W),                  //输入来自W级寄存器的指令
   .ReadData_W(ReadData_W),            //输入来自W级寄存器的内存读数据
   .ALUOutput_W(ALUOutput_W),          //输入来自W级寄存器的ALU运算结果
   .PCPlus4_W(PCPlus4_W),              //输入来自W级寄存器的PC+4

   .MUXRFWDOut(MUXRFWDOut)             //输出寄存器要写入的数据       To:  D级部件 E级部件 M级部件      /*决定真正要写入的数据是在W级*/
   );
//W级部件产生 写入寄存器的值

////////////////////////////////////////////////////////////////////////////////
//冲突单元                            //Bclass  cal_r  cal_i  Jclass  load  store  jal
   HAZARDUNIT hazardunit(
   .Instr_D(Instr_D),                  //输入来自D级寄存器的指令
   .Instr_E(Instr_E),                  //输入来自E级寄存器的指令
   .Instr_M(Instr_M),                  //输入来自M级寄存器的指令
   .Instr_W(Instr_W),                  //输入来自W级寄存器的指令
   .RegWrite_E(RegWrite_E),            //输入来自E级寄存器的写信号
   .RegWrite_M(RegWrite_M),            //输入来自M级寄存器的写信号                          /*之前连线出错，连到了E级的写信号*/
   .RegWrite_W(RegWrite_W),            //输入来自W级寄存器的写信号
   .A3_E(WriteRd_E),                   //输入来自E级寄存器的写地址        /**/       /*由于我的设计像7.58那样，是在E级处理RegDst的，所以得传入MUX后的RD_E */
   .A3_M(RD_M),                        //输入来自M级寄存器的写地址        /**/
   .A3_W(RD_W),                        //输入来自W级寄存器的写地址        /**/
   .Busy(Busy),                        //输入来自E级部件的繁忙信号
   .Start(Start),                      //输入来自E级部件的启动信号

   .Stall(Stall),                      //输出暂停信号          To:  F级部件 D级寄存器 E级寄存器
   .ForwardRSD(ForwardRSD),            //输出RSD转发信号       To:  D级部件
   .ForwardRTD(ForwardRTD),            //输出RTD转发信号       To:  D级部件

   .ForwardRSE(ForwardRSE),            //输出RSE转发信号       To:  E级部件
   .ForwardRTE(ForwardRTE),            //输出RTE转发信号       To:  E级部件
   .ForwardRTM(ForwardRTM)             //输出RTM转发信号       To:  M级部件
   );
endmodule
```

二、模块设计

1、PIPELINE_F

（1）端口定义

信号	方向	描述
CLK	I	时钟信号
Reset	I	复位信号
Stall	I	输入来自冲突单元的暂停信号
PCSel	I	来自 D 级部件的 PC 选择信号
NPCOut	I	来自 D 级部件的 NPC 输出
Instr_F	O	输出 F 级的指令
PCPlus4_F	O	输出 F 级的 PC+4

（2）功能定义

序号	内部部件	功能描述
1	IM	存储和输出当前指令
2	IFU	存储和输出当前指令地址 PC
3	MUXPC	选择下一条指令地址

（3）内部部件

1）IM

①端口定义

信号	方向	描述
InstrAddr	I	输入指令地址
Instr_F	O	输出当前指令

②功能定义

序号	功能名称	功能描述
1	取指令	根据 InstrAddr 指定的地址从 IM 中取出指令

2）IFU

①端口定义

信号	方向	描述
CLK	I	时钟信号
Reset	I	复位信号
Stall	I	来自冲突单元的暂停信号
PCIn	I	来自 MUXPC 的下一个 PC 值
PCOut	O	输出 PC 值

PCPlus4_F	O	输出当前 PC+4

②功能定义

序号	功能名称	功能描述
1	复位	当 Reset 信号有效时，PC 被置为 0x00003000

3）MUX_PC（MUX_2_1_32）

①端口定义

信号	方向	描述
PCPlus4_F	I	输入来自 IFU 的 PC+4
NPCOut	I	输入来自 D 级部件的 B 或 J 类指令对应的 PC
PCSel	I	输入来自 D 级部件的 PC 选择信号
PCIn	O	输出下一条要更新的 PC 值

②功能定义

序号	功能名称	功能描述
1	选择下一条指令地址	当 PCSel 有效时选择 NPCOut,即跳转指令的地址

2、Regsiter_D

（1）模块接口

信号	方向	描述
Instr_F	I	来自 F 级部件的指令

PCPlus4_F	I	来自 F 级部件的 PC+4
CLK	I	时钟信号
Stall	I	来自冲突单元的暂停信号
Reset	I	复位信号
Instr_D	O	输出 D 级寄存器的指令
PCPlus4_D	O	输出 D 级寄存器的 PC+4

3、PIPELINE_D

（1）端口定义

信号	方向	描述
CLK	I	时钟信号
Reset	I	复位信号
RegWrite_W	I	输入来自 W 级寄存器的写信号
RD_W	I	输入来自 W 级寄存器的写地址
Instr_D	I	输入来自 D 级寄存器的指令
MUXRFWDOut	I	输入来自 W 级部件的写数据，也是转发所需要的数据
PCPlus4_D	I	输入来自 D 级寄存器的 PC+4

ALUOutput_M	I	输入来自 M 级寄存器的 ALU 运算结果，也是转发所需要的数据
ForwardRSD	I	输入来自冲突单元的 RSD 转发控制信号
ForwardRTD	I	输入来自冲突单元的 RTD 转发控制信号
PCPlus4_W	I	输入来自 W 级寄存器的 PC+4
EXTOut	O	输出 D 级部件产生的立即数扩展结果
NPCOut	O	输出 D 级部件的跳转指令产生的 PC 值
PCSel	O	输出 D 级部件产生的 PC 选择信号
RegWrite_D	O	输出 D 级部件的指令的写信号
RSV_D	O	输出寄存器堆读数据 1
RTV_D	O	输出寄存器堆读数据 2

（2）功能定义

序号	功能名称	功能描述
1	读数据	输出指令要读的 rs、rt 的寄存器对应的值
2	扩展立即数	扩展指令中的立即数作为 E 级的 ALU 操作数
3	计算跳转地址	计算跳转指令对应的跳转地

		址
4	控制跳转	判断当前指令是否为跳转指令
5	写数据	将指令要写的数据写入 rd 寄存器中

（3）内部部件

1）GRF

①模块接口

信号	方向	描述
GRF_Rs	I	读寄存器地址 1
GRF_Rt	I	读寄存器地址 2
GRF_Rd	I	写寄存器地址
GRF_WD	I	写寄存器数据
CLK	I	时钟信号
Reset	I	复位信号
RegWrite	I	寄存器写信号
PCPlus4_W	I	写指令的地址
GRF_RD1	O	读寄存器数据 1
GRF_RD2	O	读寄存器数据 2

②功能定义

序号	功能名称	功能描述
1	读数据	读出 GRF_Rs 和 GRF_Rt 地址

		对应寄存器中的数据到 GRF_RD1、GRF_RD2
2	写数据	当 RegWrite 信号有效且时钟上升沿到来时，将 GRF_WD 写入 GRF_Rd 对应的寄存器中

2）EXT

①模块接口

信号	方向	描述
Imm16	I	进行扩展的立即数
EXTSel	I	扩展选择信号
EXTOut	O	扩展结果

②功能定义

序号	功能名称	功能描述
1	零扩展	对立即数进行高位补 0 扩展
2	符号扩展	对立即数进行符号扩展至 32 位
3	加载到高位	将立即数加载到高位，低位补 0

3）CMP

①模块接口

信号	方向	描述
CMPD1	I	比较的第一个数
CMPD2	I	比较的第二个数
CMPOut	O	比较的结果
CMPZeroOut	O	第一个数与 0 的比较结果

②功能定义

序号	功能名称	功能描述
1	比较两个操作数	CMPOut: 00：CMPD1 == CMPD2 01：CMPD1>CMPD2 10:CMPD1<CMPD2
2	比较第一个数与 0	CMPZeroOut[0]: 0:CMPD1 ≤ 0 1:CMPD1 > 0 CMPZeroOut[1]: 0:CMPD1 == 0 1:CMPD1 != 0

4）NPC

①模块接口

信号	方向	描述
GRF_RD1(MFRSDOut)	I	输入 Ra 中存储的地址
PCPlus4_D	I	输入来自 D 级寄存器的 PC+4
Instr_D	I	输入 D 级寄存器的指令，实质上是传输指令中的偏移量

NPCSel	I	输入 NPC 选择信号
NPCOut	O	输出 B J JR 指令对应的 PC

②功能定义

序号	功能名称	功能描述
1	输出跳转地址	NPCSel00：B 型指令地址 NPCSel01：J 型指令地址 NPCSel10：JR 指令地址

5）MFRSD（MUX_4_1_32）

①模块接口

信号	方向	描述
GRF_RD1	I	输入 GRF 的读数据 1
MUXRFWDOut	I	输入来自 W 级部件的 GRF 要写入的数据
ALUOutput_M	I	输入来自 M 级寄存器的 ALU 运算结果
ForwardRSD	I	输入来自冲突单元的 RSD 转发信号
MFRSDOut	O	输出选择后的指令要用的 GRF 操作数 1

②功能定义

序号	功能名称	功能描述
1	输出转发选择后的 GRF 读数	ForwardRSD：

	据 1	00 GRF_RD1 01 MUXRFWDOut 10 ALUOutput_M

6）MFRTD（MUX_4_1_32）

①模块接口

信号	方向	描述
GRF_RD2	I	输入 GRF 的读数据 2
MUXRFWDOut	I	输入来自 W 级部件的 GRF 要写入的数据
ALUOutput_M	I	输入来自 M 级寄存器的 ALU 运算结果
ForwardRTD	I	输入来自冲突单元的 RTD 转发信号
MFRTDOut	O	输出选择后的指令要用的 GRF 操作数 2

②功能定义

序号	功能名称	功能描述
1	输出转发选择后的 GRF 读数据 2	ForwardRTD： 00 GRF_RD2 01 MUXRFWDOut 10 ALUOutput_M

7）Controller_D

①模块接口

信号	方向	描述
Instr_D	I	输入来自 D 级寄存器的指令
CMPOut	I	输入来自 D 级部件 CMP 的比较结果
CMPZeroOut	I	输入来自 D 级部件 CMP 的与零比较结果
NPCSel	O	输出给 NPC 的选择信号
EXTSel	O	输出给 EXT 的选择信号
RegWrite_D	O	输出给 E 级寄存器的 GRF 写信号
PCSel	O	输出给 F 级部件 MUXPC 的选择信号

②功能定义

序号	功能名称	功能描述
1	选择指令跳转类型	通过 NPCSel 选择指令跳转地址
2	选择是否跳转	通过 PCSel 选择是否跳转
3	选择立即数扩展类型	通过 EXTSel 选择扩展类型
4	判断指令是否进行写寄存器	通过 RegWrite_D 进行判断

	操作	

4、Register_E

（1）模块接口

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
FlushE	I	来自冲突单元的清空信号,实质上是 Stall
Instr_D	I	输入来自 D 级寄存器的指令
PCPlus4_D	I	输入来自 D 级寄存器的 PC+4
RSV_D	I	输入来自 D 级部件的 RS 的数据
RTV_D	I	输入来自 D 级部件的 RT 的数据
EXTOut_D	I	输入来自 D 级部件的 EXT 结果
RegWrite_D	I	输入来自 D 级部件的 GRF 写信号
RS_D	I	输入来自 D 级寄存器的 RS

		地址
RT_D	I	输入来自 D 级寄存器的 RT
RD_D	I	输入来自 D 级寄存器的 RD
Instr_E	O	输出 E 级的指令
PCPlus4E	O	输出 E 级的 PC+4
RSV_E	O	输出 E 级的 RS 对应的 GRF 读数据 1
RTV_E	O	输出 E 级的 RD 对应的 GRF 读数据 2
EXTOut_E	O	输出 E 级的 EXT 结果
RegWrite_E	O	输出 E 的 GRF 写信号
RS_E	O	输出 E 级的 RS 地址
RT_E	O	输出 E 的 RT 地址
RD_E	O	输出 E 的 RD 地址

5、PIPELINE_E

（1）模块接口

信号	方向	描述
Instr_E	I	输入来自 E 级寄存器的指令
RTV_E	I	输入来自 E 级寄存器的 GRF 读数据 2
RSV_E	I	输入来自 E 级寄存器的 GRF 读数据 1

EXTOut_E	I	输入来自 E 级寄存器的扩展数
MUXRFWDOut	I	输入来自 W 级部件的 GRF 写数据，是转发所需数据
ALUOutput_M	I	输入来自 M 级寄存器的 ALU 运算结果，是转发所需数据
ForwardRSE	I	输入来自冲突单元的 RSE 转发选择信号
ForwardRTE	I	输入来自冲突单元的 RTE 转发选择信号
PCPlus4_E	I	输入来自 E 级寄存器的 PC+4，也是 jal 类指令要写入 31 寄存器的数
ALUOutput	O	输出 ALU 运算结果
WriteRd_E	O	输出寄存器写地址
WriteData_E	O	输出内存写数据
Start	O	输出乘除模块开始运行信号
Busy	O	输出乘除模块正在运行信号

（2）功能定义

序号	功能名称	功能描述
1	ALU 运算	通过 ALU 进行指令要求的运算
2	选择 ALU 输出，ALU 输出可作为 load store 类指令的地址，可作为 cal 运算的结果，可作为 jal 指令的写入值	如果是 JAL 类对 31 号寄存器进行地址写入的指令，置为 PCPlus4_E+4，否则如果是正常的 ALU 运算，就置为 ALU 运算结果

3	选择写入寄存器地址	如果是三寄存器操作指令，置为 rd，如果是二寄存器操作指令，置为 rt，如果是 jal 类指令，置为 31
4	选择写入内存数据，仅仅针对 store 类指令	store 类指令要写入内存的数据可在 E 级部件通过转发获取

（3）内部部件

1）ALU

①模块接口

信号	方向	描述
ALUOprand_A	I	ALU 操作数 A
ALUOprand_B	I	ALU 操作数 B
ALUSel	I	ALU 选择信号
ALUOutput	O	ALU 运算结果

②功能定义

序号	功能名称	功能描述
1	加运算	ALUOutput=ALUOprand_A+ ALUOprand_B
2	减运算	ALUOutput=ALUOprand_A- ALUOprand_B
3	或运算	ALUOutput=ALUOprand_A| ALUOprand_B
4	逻辑左移运算	ALUOutput=ALUOprand_B << ALUOprand_A[4:0]

5	逻辑右移运算	ALUOutput=ALUOprand_B >> ALUOprand_A[4:0]
6	算术右移运算	ALUOutput=$signed(ALUOprand_B) >>> ALUOprand_A[4:0]
7	与运算	ALUOutput=ALUOprand_A & ALUOprand_B
8	小于则置位	ALUOutput=$signed(ALUOprand_A) < $signed(ALUOprand_B)
9	异或运算	ALUOutput=ALUOprand_A ^ ALUOprand_B
10	或非运算	ALUOutput=~(ALUOprand_A | ALUOprand_B)
11	无符号小于则置位	ALUOutput=ALUOprand_A < ALUOprand_B

2）Controller_E

①模块接口

信号	方向	描述
Instr_E	I	输入来自 E 级寄存器的指令
ALUSel	O	输出 ALU 操作选择信号
MUXALUBSel	O	输出 ALUB 功能选择信号，选择 ALU 操作数 B 是寄存器数还是立即数
RegDst	O	输出当前指令的写寄存器选择信号

ALUOutputSel	O	输出 ALU 的结果选择信号，是 jal 型对应的 PC+8 还是正常的 ALU 运算结果，还是 HI 或 LO 寄存器的值
Start	O	输出乘除模块开始工作的信号
MDSel	O	输出乘除模块功能选择信号
MUXALUASel	O	输出 ALUA 功能选择信号，选择 ALU 操作数 A 是寄存器 RS，还是立即数

②功能定义

序号	功能名称	功能描述
1	选择 ALU 运算类型	通过 ALUSel 选择当前 E 级指令对应的 ALU 运算
2	选择 ALU 运算数	通过 MUXALUBSel 选择当前 E 级指令对应的 ALU 运算数是寄存器数还是立即数
3	选择写寄存器地址	通过 RegDst 选择当前 E 级指令写入寄存器的地址是 rt 还是 rd 还是 31
4	选择 ALU 输出	通过 ALUOutputSel 选择当前 E 级指令要用的 ALU 输出是 PC+8（JAL）还是正常的运算结果（正常运算指令），还是 HI 或 LO 寄存器的值

3）MUXALUB

①模块接口

信号	方向	描述
MFRTEOut	I	输入 ALUB 转发多选器的选择结果，为寄存器操作数
EXTOut_E	I	输入来自 E 级寄存器的扩展数
MUXALUBSel	I	输入来自 E 级控制器的 MUXALUB 的选择信号
MUXALUBOut	O	输出 ALUB

②功能定义

序号	功能名称	功能描述
1	选择 ALU 的第二个操作数	当 MUXALUBSel 为 1 时选择立即数，为 0 时选择寄存器操作数

4）MFRSE

①模块接口

信号	方向	描述
RSV_E	I	来自 E 级寄存器的的 GEF 读数据 1
MUXRFWDOut	I	来自 W 级部件的 GRF 写回数据
ALUOutput_M	I	来自 M 级寄存器的 ALU 的输出
ForwardRSE	I	来自 E 级控制器的 MFRSE 的选择信号
MFRSEOut	O	输出 MFRSE 选择后的数据

②功能定义

序号	功能名称	功能描述
1	选择 ALU 的第一个操作数	ForwardRSE: 00 RSV_E 01 MUXRFWDOut 10 ALUOutput_M

5）MFRTE

①模块接口

信号	方向	描述
RTV_E	I	来自 E 级寄存器的 GRF 的读数据 2
MUXRFWDOut	I	来自 W 级寄存器的 GRF 写回数据
ALUOutput_M	I	来自 M 级寄存器的 ALU 的输出
ForwardRTE	I	来自 E 级控制器的 MFRTE 的选择信号
MFRTEOut	O	输出 MFRTE 选择后的数据

②功能定义

序号	功能名称	功能描述
1	选择 ALU 的第二个操作数中的寄存器数	ForwardRTE: 00 RTV_E

		01 MUXRFWDOut 10 ALUOutput_M

6）MUXRDE

①模块接口

信号	方向	描述
RT_E	I	来自 E 级寄存器的 GRF 写地址 1
RD_E	I	来自 E 级寄存器的 GRF 写地址 2
5’b11111	I	31 号寄存器
RegDst	I	来自 E 级控制器的写寄存器选择信号
WriteRd_E	O	输出 E 级当前指令要写的寄存器地址

②功能定义

序号	功能名称	功能描述
1	选择当前指令要写入的寄存器地址	RegDst: 00 RT_E 01 RD_E 10 31

7）MUXALUOutput

①模块接口

信号	方向	描述
ALUOutput_ALU	I	输入来自 ALU 的运算结果
PCPlus4_E+4	I	输入来自 E 级寄存器的 PCPlus4_E+4，作为 JAL 指令对应要输入的值
HI	I	输入来自 E 级乘除模块的 HI 寄存器的值
LO	I	输入来自 E 级乘除模块的 LO 寄存器的值
ALUOutputSel	I	输入来自 E 级寄存器 E 级的 ALU 输出选择信号
ALUOutput	O	输出 E 级的 ALU 运算结果，其可能是正常指令的运算结果，也可能是对应 JAL 指令的 PC+8

②功能定义

序号	功能名称	功能描述
1	选择 E 级部件的 ALU 运算输出是正常 ALU 运算指令的运算结果还是 JAL 指令对应的 PC+8	ALUOutputSel 0 ALUOutput_ALU 1 PCPlus4_E+4

8）MUXALUA

①模块接口

信号	方向	描述
MFRSEOut	I	输入 ALUA 转发多选器的选择结果，为寄存器操作数
EXTOut_E	I	输入来自 E 级寄存器的扩展数
MUXALUASel	I	输入来自 E 级控制器的 MUXALUA 的选择信号
MUXALUAOut	O	输出 ALUA

②功能定义

序号	功能名称	功能描述
1	选择 ALU 的第一个操作数	当 MUXALUASel 为 1 时选择来自 EXT 的立即数，为 0 时选择来自 RS 的寄存器数，为满足移位指令特设

9）MULTDIVPART

①模块接口

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
Start	I	输入乘除模块开始工作信号

ALUOprand_A	I	输入乘除模块操作数 1
ALUOprand_B	I	输入乘除模块操作数 2
MDSel	I	输入乘除模块控制信号
HI	O	输出 HI 寄存器的值
LO	O	输出 LO 寄存器的值
Busy	O	输出乘除模块正在工作的信号

②功能定义

序号	功能名称	功能描述
1	进行乘除相关运算	MDSel： 1： MULT 2: MULTU 3: DIV 4: DIVU 5: MTHI 6: MTLO
2	输出正在工作标志	Busy = Counter > 0;

6、Register_M

（1）模块接口

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
Instr_E	I	输入来自 E 级寄存器的指令
RTV_E	I	输入来自 E 级部件的要写入内存的数据
PCPlus4_E	I	输入来自 E 级寄存器的的 PC+4
ALUOutput_E	I	输入来自 E 级部件的 ALU 运算结果
RegWrite_E	I	输入来自 E 级寄存器的 GRF 写信号
WriteRd_E	I	输入来自 E 级部件的当前指令对应的写寄存器地址
RegWrite_M	O	输出 M 级寄存器的 GRF 写信号
Instr_M	O	输出 M 级寄存器的指令
RTV_M	O	输出 M 级寄存器的要写入内存的数据
PCPlus4_M	O	输出 M 级寄存器的 PC+4
ALUOutput_M	O	输出 M 级寄存器的 ALU 运算结果
RD_M	O	输出 M 级寄存器的写寄存器地址

7、PIPELINE_M

（1）模块接口

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
ALUOutput_M	I	输入 M 级寄存器的 ALU 运算结果，作为存取内存的地址
RTV_M	I	输入来自 M 级寄存器的写入内存数据
Instr_M	I	输入来自 M 级寄存器的指令
ForwardRTM	I	输入来自冲突单元的 RTM 转发控制信号
PCPlus4_M	I	输入来自 M 级寄存器的 PC+4，用来 display
MUXRFWDOut	I	输入来自 W 级部件的 GRF 写入数据，是转发所需数据
MemOut	O	输出 M 级部件读出的内存数据

（2）功能定义

序号	功能名称	功能描述
1	读内存	根据输入的 ALUOutput_M 作为内存地址读出内存中的数据
2	写内存	当写内存信号有效且在时钟

		上升沿将数据写入对应的内存地址
3	复位	当 Reset 信号有效且时钟上升沿到来时将所有内存数据置零

（3）内部部件

1）DM

①模块接口

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
MemWrite	I	输入来自 M 级控制器的内存写使能信号
MemAddr	I	输入来自 M 级寄存器额内存写地址，即 ALU 运算结果
MemWD	I	输入来自 MFRTM 的内存写数据
PCPlus4_M	I	输入来自 M 级寄存器的 PC+4，用来 display
MemOut	O	输出内存读数据

②功能定义

（4）

序号	功能名称	功能描述
1	读内存	根据输入的 ALUOutput_M 作为内存地址读出内存中的数

		据
2	写内存	当写内存信号有效且在时钟上升沿将数据写入对应的内存地址
3	复位	当 Reset 信号有效且时钟上升沿到来时将所有内存数据置零

2）Controller_M

①模块接口

信号	方向	描述
Instr_M	I	来自 M 级的指令
MemWrite	O	输出内存写使能信号
StoreSel	O	输出写入内存的数据选择信号

②功能定义

序号	功能名称	功能描述
1	判断是否写内存	根据当前的 M 级指令输出 MemWrite 信号控制是否写内存
2	选择写入内存的数据	根据当前 M 级指令输出

		StoreSel 信号选择写入内存的数据

3）MFRTM

①模块接口

信号	方向	描述
RTV_M	I	输入来自 M 级寄存器的 GRF 读数据 2
MUXRFWDOut	I	输入来自 W 级部件的 GRF 写回数据，是转发所需数据
ForwardRTM	I	输入来自冲突单元的 MFRTM 选择信号
MFRTMOut	O	输出 MFRTM 的转发选择结果，作为写入内存的数据

②功能定义

序号	功能名称	功能描述
1	选择写入内存的数据	ForwardRTM: 0 RTV_M 1 MUXRFWDOut

4）StoreBE

①模块接口

信号	方向	描述
MFRTMOut	I	输入经过转发选择后的要存

		储的数据
StoreSel	I	输入来自 M 级控制器的数据存储方式选择信号
BSel	I	输入存储数据的字段选择信号
MemRD	I	输入存储地址当前对应的数据
MemWD	O	输出要存入存储地址的数据

②功能定义

序号	功能名称	功能描述
1	选择写入内存的数据的字段	BSel: 00: [7:0] 01: [15:8] 10: [23:16] 11: [31:24]
2	选择写入内存的数据的方式	StoreSel: 00: sw 01: sb 10: sh

8、Register_W

信号	方向	描述
CLK	I	输入时钟信号
Reset	I	输入复位信号
Instr_M	I	输入来自 M 级寄存器的指令
PCPlus4_M	I	输入来自 M 级寄存器的 PC+4

ALUOutput_M	I	来自M级寄存器的ALU输出
ReadData_M	I	输入来自 M 级部件的内存读数据
RD_M	I	输入来自 M 级寄存器的 GRF 写地址
RegWrite_M	I	输入来自 M 级寄存器的 GRF 写信号
Instr_W	O	输出 W 级寄存器的指令
PCPlus4_W	O	输出 W 级寄存器的 PC+4
ALUOutput_W	O	输出W级寄存器的ALU输出
ReadData_W	O	输出 W 级寄存器的内存读数据
RD_W	O	输出W级寄存器的GRF写地址地址
RegWrite_W	O	输出W级寄存器的GRF写信号

9、PIPELINE_W

（1）模块接口

信号	方向	描述
Instr_W	I	输入来自 W 级寄存器的指令
ReadData_W	I	输入来自 W 级寄存器的内存读数据
ALUOutput_W	I	输入来自W级寄存器的ALU运算结果
PCPlus4_W	I	输入来自 W 级寄存器的 PC+4
MFRDWDOut	O	输出经过选择后的 GRF 要写入的数据

（2）功能定义

序号	功能名称	功能描述
1	选择写入 GRF 的数据	MemtoReg_W: 0 ALUOutput_W 1 ReadData_W

（3）内部部件

1）Controller_W

①模块接口

信号	方向	描述
Instr_W	I	来自 W 级的指令
MemtoReg_W	O	MUXRFWD 的选择信号
LoadSel	O	输出内存取出的数据的选择信号

②功能定义

序号	功能名称	功能描述
1	输出选择写入 GRF 的数据的控制信号	Memtoreg: load 指令 1，其他指令 0
2	输出选择写入 GRF 的内存读数据的方式的选择信号	LoadSel: 000 lw 001 lb 010 lub

		011 lh 100 lhu

2）MUXRFWD

①模块接口

信号	方向	描述
ReadData_W	I	来自 W 级的内存数据
ALUOutput_W	I	来自 W 级的 ALU 输出
MemtoReg_W	I	MUXRFWD 选择信号
MUXRFWDOut	O	输出寄存器写入数据

②功能定义

序号	功能名称	功能描述
1	选择写入 GRF 的数据	MemtoReg_W: 0 ALUOutput_W 1 ReadData_W

3）LoadBE

①模块接口

信号	方向	描述
ReadData_W	I	输入来自 W 级寄存器的内存数据
BSel	I	输入要读取的内存读数据的字段选择信号
LoadSel	I	输入要读取的内存读数据的数据形式的选择信号
ReadData	O	输出指令要读取的数据

②功能定义

序号	功能名称	功能描述
1	选择指令要读取的内存的数据的字段	BSel： 00：[7:0] 01：[15:8] 10：[23:16] 11：[31:24]
2	选择指令要读取的内存的数据的形式	LoadSel： 000 lw 001 lb 010 lub 011 lh 100 lhu

10、CONTROLLER

```
21  `define OPCODE 31:26
22  `define FUNC   5:0
23  `define RS     25:21
24  `define RT     20:16
25  `define RD     15:11
26  `define RCLASS 6'b000000
27  `define REGIMM 6'b000001
28
29  `define LB     6'b100000
30  `define LBU    6'b100100
31  `define LH     6'b100001
32  `define LHU    6'b100101
33  `define LW     6'b100011
34  `define SB     6'b101000
35  `define SH     6'b101001
36  `define SW     6'b101011
37  `define ADD    6'b100000
38  `define ADDU   6'b100001
39  `define ADDI   6'b001000
40  `define ADDIU  6'b001001
41  `define SUB    6'b100010
42  `define SUBU   6'b100011
43  `define AND    6'b100100
44  `define ANDI   6'b001100
45  `define OR     6'b100101
46  `define ORI    6'b001101
47  `define XOR    6'b100110
48  `define XORI   6'b001110
49  `define NOR    6'b100111
50  `define SLLV   6'b000100
51  `define SLL    6'b000000
52  `define SRLV   6'b000110
53  `define SRL    6'b000010
```

```
`define SRAV   6'b000111
`define SRA    6'b000011
`define SLT    6'b101010
`define SLTU   6'b101011
`define SLTI   6'b001010
`define SLTIU  6'b001011
`define LUI    6'b001111
`define BEQ    6'b000100
`define BNE    6'b000101
`define BLEZ   6'b000110
`define BGTZ   6'b000111
`define BLTZ   5'b00000////////
`define BGEZ   5'b00001////////
`define J      6'b000010
`define JAL    6'b000011
`define JR     6'b001000
`define JALR   6'b001001
`define MULT   6'b011000
`define MULTU  6'b011001
`define DIV    6'b011010
`define DIVU   6'b011011
`define MFHI   6'b010000
`define MFLO   6'b010010
`define MTHI   6'b010001
`define MTLO   6'b010011

module CONTROLLER_D(
    input [31:0] Instr_D,                                   //输入来自D级寄存器的指令
    input  CMPOut,                                        //输入来自D级部件（CMP）的比较结果
    input [1:0] CMPZeroOut,
    //拼接传输的，{ifeqz,ifgtz}
    output [1:0] NPCSel,                                    //输出给D级部件NPC的NPC选择信号
    output [3:0] EXTSel,                                    //输出给D级部件EXT的EXT选择信号

    output RegWrite_D,                                      //输出给E级寄存器的当前D级指令的写寄存器信号
    output PCSel                                            //输出给F级部件的PC选择信号
     );
    wire [5:0] opcode = Instr_D[`OPCODE], func = Instr_D[`FUNC];

    assign NPCSel[0] = (opcode == `J) || (opcode == `JAL);   //Bclass Jclass JR
    assign NPCSel[1] = (opcode == `RCLASS && func == `JR) || (opcode == `RCLASS && func == `JALR);

    assign EXTSel[0] = (opcode == `LW) || (opcode == `SW) ||(opcode == `ADDI) || (opcode == `ADDIU) ||
                       (opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SRL) ||
                       (opcode == `RCLASS && func == `SRA) ||(opcode == `SLTI) || (opcode == `SLTIU)||
                       (opcode == `SB) || (opcode == `SH) || (opcode == `LB) || (opcode == `LBU) || (opcode == `LH)||(opcode == `LHU);
                      //应该是SW LW要SIGN_EXT而不是BEQ store load cal_i

    assign EXTSel[1] = (opcode == `LUI) || (opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SRL)||
                       (opcode == `RCLASS && func == `SRA);

    assign EXTSel[2] = 0;//还没用到
    assign EXTSel[3] = 0;//还没用到

    assign RegWrite_D = (opcode == `LW) || (opcode == `RCLASS && func == `ADDU) || (opcode == `RCLASS && func == `SUBU) || (opcode == `ORI) ||
                        (opcode == `LUI) || (opcode == `JAL) || (opcode == `RCLASS && func == `ADD) || (opcode == `ADDI) ||
                        (opcode == `ADDIU) || (opcode == `RCLASS && func == `SUB) || (opcode == `RCLASS && func == `OR) ||
                        (opcode == `RCLASS && func == `AND) || (opcode == `ANDI) || (opcode == `RCLASS && func == `XOR) ||
                        (opcode == `XORI) || (opcode == `RCLASS && func == `NOR) || (opcode == `RCLASS && func == `SLL) ||
                        (opcode == `RCLASS && func == `SLLV) || (opcode == `RCLASS && func == `SRL) ||
                        (opcode == `RCLASS && func == `SRLV) ||(opcode == `RCLASS && func == `SRA)|| (opcode == `RCLASS && func == `SRAV)||
                        (opcode == `RCLASS && func == `SLT) || (opcode == `SLTI) || (opcode == `RCLASS && func == `SLTU) ||
                        (opcode == `SLTIU) || (opcode == `LB) || (opcode == `LBU)||(opcode == `LH)||(opcode == `LHU)||
                        (opcode == `RCLASS && func == `JALR) || (opcode == `RCLASS && func == `MFHI)||
                        (opcode == `RCLASS && func == `MFLO);//cal_r cal_i load jal

    assign PCSel = (CMPOut && (opcode == `BEQ)) || (~CMPOut && (opcode == `BNE)) ||
                   ((~CMPZeroOut[0] && ~CMPZeroOut[1]) && (opcode == `REGIMM && Instr_D[`RT] == `BLTZ))||
                   (CMPZeroOut[0] && (opcode == `BGTZ)) || ( ~CMPZeroOut[0] && (opcode == `BLEZ)) ||
                   ((CMPZeroOut[0] || CMPZeroOut[1]) && (opcode == `REGIMM && Instr_D[`RT] == `BGEZ)) || (opcode == `J) ||
                   (opcode == `JAL) || (opcode == `RCLASS && func == `JR)||
                   (opcode == `RCLASS && func == `JALR);
                   //Bclass Jclass JR jal
endmodule
////////////////////////////////////////////////////////////////////////////////
module CONTROLLER_E(
    input [31:0] Instr_E,
    output Start,
    output [3:0] MDSel,
    output [3:0] ALUSel,
    output MUXALUBSel,
    output MUXALUASel,
    output [1:0] RegDst,                        //处理JAL
    output [1:0] ALUOutputSel                       //处理普通的运算指令和对RA进行操作的指令的ALU输出的问题
     );
    wire [5:0] opcode = Instr_E[`OPCODE], func = Instr_E[`FUNC];

    assign ALUSel[0] = (opcode == `RCLASS && func == `SUBU) || (opcode == `RCLASS && func == `SUB) || (opcode == `RCLASS && func == `NOR)||
                       (opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SLLV) || (opcode == `RCLASS && func == `SRA)||
                       (opcode == `RCLASS && func == `SRAV)|| (opcode == `RCLASS && func == `SLT) || (opcode == `SLTI);
                       //cal_r cal_i load store

    assign ALUSel[1] = (opcode == `ORI) || (opcode == `RCLASS && func == `OR) || (opcode == `RCLASS && func == `AND) ||
                         (opcode == `ANDI) || (opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SLLV)||
                         (opcode == `RCLASS && func == `SLT) || (opcode == `SLTI) || (opcode == `RCLASS && func == `SLTU)||
                         (opcode == `SLTIU);

    assign ALUSel[2] = (opcode == `RCLASS && func == `AND) || (opcode == `ANDI) || (opcode == `RCLASS && func == `SRL)||
                       (opcode == `RCLASS && func == `SRLV) || (opcode == `RCLASS && func == `SRA) || (opcode == `RCLASS && func == `SRAV)||
                       (opcode == `RCLASS && func == `SLT) || (opcode == `SLTI);
```

```
153
154     assign ALUSel[3] = (opcode == `RCLASS && func == `XOR) || (opcode == `XORI) || (opcode == `RCLASS && func == `NOR) ||
155                        (opcode == `RCLASS && func == `SLTU)|| (opcode == `SLTIU);
156
157     assign MUXALUASel =(opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SRL)||(opcode == `RCLASS && func == `SRA);
158
159     assign MUXALUBSel = (opcode == `ORI) || (opcode == `LUI) || (opcode == `LW) || (opcode == `SW) || (opcode == `ADDI) ||
160                         (opcode == `ADDIU) || (opcode == `ANDI) || (opcode == `XORI) || (opcode == `SLTI) || (opcode == `SLTIU)||
161                         (opcode == `SB) || (opcode == `SH) || (opcode == `LB) || (opcode == `LBU) || (opcode == `LH)||(opcode == `LHU);
162                         // 1 的话就是立即数运算
163     //cal_r cal_i load store
164     assign RegDst = ((opcode == `JAL))?                                    2:
165                    ((opcode == `RCLASS&& func == `ADDU) || (opcode == `RCLASS && func == `SUBU)||
166                      (opcode == `RCLASS && func == `SUB) || (opcode == `RCLASS && func == `ADD)
167                      || (opcode == `RCLASS && func == `OR) || (opcode == `RCLASS && func == `AND)
168                      || (opcode == `RCLASS && func == `XOR) || (opcode == `RCLASS && func == `NOR)
169                      || (opcode == `RCLASS && func == `SLL) || (opcode == `RCLASS && func == `SLLV)
170                      || (opcode == `RCLASS && func == `SRL) || (opcode == `RCLASS && func == `SRLV)
171                      || (opcode == `RCLASS && func == `SRA) || (opcode == `RCLASS && func == `SRAV)
172                      || (opcode == `RCLASS && func == `SLT) || (opcode == `RCLASS && func == `SLTU)
173                      ||(opcode == `RCLASS && func == `JALR) || (opcode == `RCLASS && func == `MFHI)
174                      ||(opcode == `RCLASS && func == `MFLO))?       1 :
175                                                                                                    0 ;
176     //cal_r jal
177     assign ALUOutputSel[0] = (opcode == `JAL) || (opcode == `RCLASS && func == `JALR) || (opcode == `RCLASS &&func == `MFLO);
178           //jal 增加一位，选择乘除的输出，还得看选择的是HI还是LO
179
180     assign ALUOutputSel[1] = (opcode == `RCLASS && func == `MFHI) || (opcode == `RCLASS && func == `MFLO);
181
182     assign Start = (opcode == `RCLASS && func == `MULT) || (opcode == `RCLASS && func == `MULTU) || (opcode == `RCLASS && func == `DIV) ||
183                    (opcode == `RCLASS && func == `DIVU) || (opcode == `RCLASS && func == `MTHI) ||(opcode == `RCLASS && func == `MTLO);
184
185     assign MDSel[0] = (opcode == `RCLASS && func == `MULT) || (opcode == `RCLASS && func == `DIV) || (opcode == `RCLASS && func == `MTHI);
186     assign MDSel[1] = (opcode == `RCLASS && func == `MULTU) || (opcode == `RCLASS && func == `DIV) || (opcode == `RCLASS && func == `MTLO);
187     assign MDSel[2] = (opcode == `RCLASS && func == `DIVU) || (opcode == `RCLASS && func == `MTHI) || (opcode == `RCLASS && func == `MTLO);
188     assign MDSel[3] = 0;//暂时未用到
189
190 endmodule
191 //////////////////////////////////////////////////////////////////////////////////
192 module CONTROLLER_M(
193     input [31:0] Instr_M,
194     output [1:0] StoreSel,
195     output MemWrite
196      );
197     wire [5:0] opcode = Instr_M[`OPCODE], func = Instr_M[`FUNC];
198
199     assign StoreSel[0] = (opcode == `SB);
200     assign StoreSel[1] = (opcode == `SH);
201
202     assign MemWrite = (opcode == `SW) || (opcode == `SB) || (opcode == `SH);   //store
203
204 endmodule
205 //////////////////////////////////////////////////////////////////////////////////
206 module CONTROLLER_W(
207     input [31:0] Instr_W,
208     output [2:0] LoadSel,
209     output [1:0] MemtoReg_W
210      );
211     wire [5:0] opcode = Instr_W[`OPCODE], func = Instr_W[`FUNC];
212
213     assign LoadSel[0] = (opcode == `LB) || (opcode == `LH);
214     assign LoadSel[1] = (opcode == `LBU) || (opcode == `LH);
215     assign LoadSel[2] = (opcode == `LHU);
216
217     assign MemtoReg_W = (opcode == `JAL)? 2:  //load jal
218                         ((opcode == `LW) || (opcode == `LB) || (opcode == `LBU)||(opcode == `LH)||(opcode == `LHU))?  1:
219                                             0;
220
221 endmodule
222
```

	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
addu	000000	100001	X	X	0	1	1	0	0	0	0	0
add	000000	100000	X	X	0	1	1	0	0	0	0	0
addi	001000	N/A	X	1	0	1	0	0	0	1	0	0

addiu	001001	N/A	X	1	0	1	0	0	0	1	0	0
subu	000000	100011	X	X	0	1	1	0	1	0	0	0
sub	000000	100010	X	X	0	1	1	0	1	0	0	0
ori	001101	N/A	X	0	0	1	0	0	2	1	0	0
or	000000	100101	X	X	0	1	1	0	2	0	0	0
sll	000000	000000	X	3	0	1	1	0	3	0	0	0
	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
sllv	000000	000100	X	X	0	1	1	0	3	0	0	0
srl	000000	000010	X	3	0	1	1	0	4	0	0	0
srlv	000000	000110	X	X	0	1	1	0	4	0	0	0
sra	000000	000011	X	3	0	1	1	0	5	0	0	0
srav	000000	000111	X	X	0	1	1	0	5	0	0	0
and	000000	100100	X	X	0	1	1	0	6	0	0	0
andi	001100	N/A	X	0	0	1	0	0	6	1	0	0
slt	000000	101010	X	X	0	1	1	0	7	0	0	0
slti	001010	N/A	X	1	0	1	0	0	7	1	0	0
sltu	000000	101011	X	X	0	1	1	0	10	0	0	0
sltiu	001011	N/A	X	1	0	1	0	0	10	1	0	0
xor	000000	100110	X	X	0	1	1	0	8	0	0	0
xori	001110	N/A	X	0	0	1	0	0	8	1	0	0

nor	000000	100111	X	X	0	1	1	0	9	0	0	0
lui	001111	N/A	X	2	0	1	0	0	0	1	0	0
lw	100011	N/A	X	1	0	1	0	0	0	1	0	1
lb	100000	N/A	X	1	0	1	0	0	0	1	0	1
lbu	100100	N/A	X	1	0	1	0	0	0	1	0	1
lh	100001	N/A	X	1	0	1	0	0	0	1	0	1
	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
lhu	100101	N/A	X	1	0	1	0	0	0	1	0	1
sw	101011	N/A	X	1	0	0	X	0	0	1	1	0
sb	101000	N/A	X	1	0	0	X	0	0	1	1	0
sh	101001	N/A	X	1	0	0	X	0	0	1	1	0
beq	000100	N/A	0	X	1	0	X	X	0	0	0	0
bne	000101	N/A	0	X	1	0	X	X	0	0	0	0
blez	000110	N/A	0	X	1	0	X	X	0	0	0	0
bgtz	000111	N/A	0	X	1	0	X	X	0	0	0	0
bltz	000001	00000	0	X	1	0	X	X	0	0	0	0
bgez	000001	00001	0	X	1	0	X	X	0	0	0	0
j	000010	N/A	1	X	1	0	X	X	0	X	0	0
jr	000000	001000	2	X	1	0	X	X	0	X	0	0
jal	000011	N/A	1	X	1	1	2	1	0	X	0	0

jalr	000000	001001	2	X	1	1	1	1	0	X	0	0
mult	000000	011000	0	0	0	0	0	0	0	0	0	0
multu	000000	011001	0	0	0	0	0	0	0	0	0	0
div	000000	011010	0	0	0	0	0	0	0	0	0	0
divu	000000	011011	0	0	0	0	0	0	0	0	0	0
	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
mfhi	000000	010000	0	0	0	1	1	2	X	X	0	0
mflo	000000	010010	0	0	0	1	1	3	X	X	0	0
mthi	000000	010001	0	0	0	0	0	0	0	0	0	0
mtlo	000000	010011	0	0	0	0	0	0	0	0	0	0

特殊指令的控制信号：

（1）移位类指令

	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUXALUASel	MUXALUB Sel	Mem Write	Mem toReg
sll	000000	000000	X	3	0	1	1	0	3	1	0	0	0
sllv	000000	000100	X	X	0	1	1	0	3	0	0	0	0
srl	000000	000010	X	3	0	1	1	0	4	1	0	0	0
srlv	000000	000110	X	X	0	1	1	0	4	0	0	0	0
sra	000000	000011	X	3	0	1	1	0	5	1	0	0	0

srav	000000	000111	X	X	0	1	1	0	5	0	0	0	0

（2）跳转类指令

	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
beq	000100	N/A	0	X	1	0	X	X	0	0	0	0
bne	000101	N/A	0	X	1	0	X	X	0	0	0	0
blez	000110	N/A	0	X	1	0	X	X	0	0	0	0
bgtz	000111	N/A	0	X	1	0	X	X	0	0	0	0
bltz	000001	00000	0	X	1	0	X	X	0	0	0	0
bgez	000001	00001	0	X	1	0	X	X	0	0	0	0

（3）小于则置位类指令

	op code	func	NPC Sel	EXT Sel	PC Sel	Reg Write	Reg Dst	ALU Output Sel	ALU Sel	MUX ALU B Sel	Mem Write	Mem toReg
slt	000000	101010	X	X	0	1	1	0	7	0	0	0
slti	001010	N/A	X	1	0	1	0	0	7	1	0	0

sltu	000000	101011	X	X	0	1	1	0	7	10	0	0
sltiu	001011	N/A	X	1	0	1	0	0	7	10	0	0

（4）Load 和 Store 类指令

	opcode	func	NPC Sel	EXT Sel	PC Sel	RegWrite	RegDst	ALU Output Sel	ALU Sel	MUXALUB Sel	MemWrite	StoreSel	MemtoReg	LoadSel
lw	100011	N/A	X	1	0	1	0	0	0	1	0	X	1	000
lb	100000	N/A	X	1	0	1	0	0	0	1	0	X	1	001
lbu	100100	N/A	X	1	0	1	0	0	0	1	0	X	1	010
lh	100001	N/A	X	1	0	1	0	0	0	1	0	X	1	011
lhu	100101	N/A	X	1	0	1	0	0	0	1	0	X	1	100
sw	101011	N/A	X	1	0	0	X	0	0	1	1	00	0	X
sb	101000	N/A	X	1	0	0	X	0	0	1	1	01	0	X
sh	101001	N/A	X	1	0	0	X	0	0	1	1	10	0	X

（5）乘除类指令

	opcode	func	NPC Sel	EXT Sel	PC Sel	RegWrite	RegDst	ALU OutputSel	ALU Sel	MUXALUB Sel	start	MD Sel	MemWrite	MemtoReg
mult	000000	011000	0	0	0	0	0	0	0	0	1	1	0	0
multu	000000	011001	0	0	0	0	0	0	0	0	1	2	0	0
div	000000	011010	0	0	0	0	0	0	0	0	1	3	0	0
divu	000000	011011	0	0	0	0	0	0	0	0	1	4	0	0
mfhi	000000	010000	0	0	0	1	1	2	X	X	0	0	0	0
mflo	000000	010010	0	0	0	1	1	3	X	X	0	0	0	0
mthi	000000	010001	0	0	0	0	0	0	0	0	1	5	0	0
mtlo	000000	010011	0	0	0	0	0	0	0	0	1	6	0	0

（1）Controller_D

①模块接口

信号	方向	描述
Instr_D	I	输入来自 D 级寄存器的指令
CMPOut	I	输入来自 D 级部件 CMP 的比较结果
NPCSel	O	输出给 NPC 的选择信号
EXTSel	O	输出给 EXT 的选择信号
RegWrite_D	O	输出给 E 级寄存器的 GRF 写信号
PCSel	O	输出给 F 级部件 MUXPC 的选择信号

②功能定义

序号	功能名称	功能描述
1	选择指令跳转类型	通过 NPCSel 选择指令跳转地址
2	选择是否跳转	通过 PCSel 选择是否跳转
3	选择立即数扩展类型	通过 EXTSel 选择扩展类型
4	判断指令是否进行写寄存器操作	通过 RegWrite_D 进行判断

（2）Controller_E

①模块接口

信号	方向	描述
Instr_E	I	输入来自 E 级寄存器的指令
ALUSel	O	输出 ALU 操作选择信号
MUXALUBSel	O	输出 ALUB 功能选择信号，

		选择 ALU 操作数 B 是寄存器数还是立即数
MUXALUASel	O	输出 ALUA 功能选择信号，选择 ALU 操作数 A 是来自寄存器 RS 还是立即数的位移字段
RegDst	O	输出当前指令的写寄存器选择信号
ALUOutputSel	O	输出 ALU 的结果选择信号，是 jal 型对应的 PC+8 还是正常的 ALU 运算结果
Start	O	输出乘除模块的运行开始信号
MDSel	O	输出乘除模块的操作选择信号

②功能定义

序号	功能名称	功能描述
1	选择 ALU 运算类型	通过 ALUSel 选择当前 E 级指令对应的 ALU 运算
2	选择 ALU 运算数	通过 MUXALUBSel 选择当前 E 级指令对应的 ALU 运算数是寄存器数还是立即数
3	选择写寄存器地址	通过 RegDst 选择当前 E 级指令写入寄存器的地址是 rt 还是 rd 还是 31
4	选择 ALU 输出	通过 ALUOutputSel 选择当前 E 级指令要用的 ALU 输出是 PC+8（JAL）还是正常的运算结果（正常运算指令）

6	选择乘除模块是否开始工作	通过 Start 信号控制乘除模块是否开始工作
7	选择乘除模块运算类型	通过 MDSel 选择乘除模块的运算类型

（3）Controller_M

①模块接口

信号	方向	描述
Instr_M	I	来自 M 级的指令
MemWrite	O	输出内存写使能信号
StoreSel	O	输出内存写数据的形式选择信号

②功能定义

序号	功能名称	功能描述
1	判断是否写内存	根据当前的 M 级指令输出 MemWrite 信号控制是否写内存
2	选择内存写入数据的形式	根据当前的 M 级指令输出 StoreSel 信号控制内存写入数据的形式

（4）Controller_W

①模块接口

信号	方向	描述
Instr_W	I	来自 W 级的指令

MemtoReg_W	O	MUXRFWD 的选择信号
LoadSel	O	输出读内存指令要读的数据的形式控制信号

②功能定义

序号	功能名称	功能描述
1	输出选择写入 GRF 的数据的控制信号	load 指令 1，其他指令 0
2	输出读内存指令要读的形式控制信号	LoadSel： 000 lw 001 lb 010 lub 011 lh 100 lhu

11、HAZARDUNIT

```
21  `define OPCODE 31:26
22  `define FUNC   5:0
23  `define RS     25:21
24  `define RT     20:16
25  `define RD     15:11
26  `define RCLASS 6'b000000
27  `define REGIMM 6'b000001
28
29  `define LB     6'b100000
30  `define LBU    6'b100100
31  `define LH     6'b100001
32  `define LHU    6'b100101
33  `define LW     6'b100011
34  `define SB     6'b101000
35  `define SH     6'b101001
36  `define SW     6'b101011
37  `define ADD    6'b100000
38  `define ADDU   6'b100001
39  `define ADDI   6'b001000
40  `define ADDIU  6'b001001
41  `define SUB    6'b100010
42  `define SUBU   6'b100011
43  `define AND    6'b100100
44  `define ANDI   6'b001100
45  `define OR     6'b100101
46  `define ORI    6'b001101
47  `define XOR    6'b100110
48  `define XORI   6'b001110
49  `define NOR    6'b100111
50  `define SLLV   6'b000100
51  `define SLL    6'b000000
52  `define SRLV   6'b000110
53  `define SRL    6'b000010
```

```
`define SRAV   6'b000111
`define SRA    6'b000011
`define SLT    6'b101010
`define SLTU   6'b101011
`define SLTI   6'b001010
`define SLTIU  6'b001011
`define LUI    6'b001111
`define BEQ    6'b000100
`define BNE    6'b000101
`define BLEZ   6'b000110
`define BGTZ   6'b000111
`define BLTZ   5'b00000////////
`define BGEZ   5'b00001////////
`define J      6'b000010
`define JAL    6'b000011
`define JR     6'b001000
`define JALR   6'b001001
`define MULT   6'b011000
`define MULTU  6'b011001
`define DIV    6'b011010
`define DIVU   6'b011011
`define MFHI   6'b010000
`define MFLO   6'b010010
`define MTHI   6'b010001
`define MTLO   6'b010011
module HAZARDUNIT(
   input Start,
   input Busy,
   input [31:0] Instr_D, Instr_E, Instr_M, Instr_W,//传入指令
   input RegWrite_E, RegWrite_M, RegWrite_W,//传入写信号
   input [4:0] A3_E, A3_M, A3_W,//传入A3_M方便，不然在这里判断很麻烦
   output Stall,
   output [1:0] ForwardRSD, ForwardRTD, ForwardRSE, ForwardRTE,
   output ForwardRTM
    );

    wire MDInstr, MDStall;
    wire Tuse_RS0, Tuse_RS1, Tuse_RT0, Tuse_RT1, Tuse_RT2;//这个用来表示指令，这样容易判断寄存器使用情况，如果仅仅是单纯的译码出时间，不行
    wire Stall_RS0_E1, Stall_RS0_E2, Stall_RS0_M1, Stall_RS1_E2, Stall_RS,
         Stall_RT0_E1, Stall_RT0_E2, Stall_RT0_M1, Stall_RT1_E2, Stall_RT;
    wire [1:0] Tnew_E, Tnew_M;

   //产生乘除停止信号
   assign MDInstr = (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULT) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULTU) ||
                    (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIV) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIVU) ||
                    (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MFHI) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MFLO) ||
                    (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MTHI) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MTLO);
   assign MDStall = Busy || (Start && MDInstr);
   //产生Tnew
    DECODER_Tnew_E decoder_tnew_e(
    .Instr(Instr_E),
    .Tnew(Tnew_E)
    );

    DECODER_Tnew_M decoder_tnew_m(
    .Instr(Instr_M),
    .Tnew(Tnew_M)
    );
   //产生Tuse
   //Tuse和Tnew更多的是用来判断暂停的

   //在D级还有0个周期需要使用RS  beq jr
   assign Tuse_RS0 = (Instr_D[`OPCODE] == `BEQ) || (Instr_D[`OPCODE] == `BNE) || (Instr_D[`OPCODE] == `REGIMM && Instr_D[`RT] == `BLTZ) ||
                     (Instr_D[`OPCODE] == `BGTZ) || (Instr_D[`OPCODE] == `BLEZ) || (Instr_D[`OPCODE] == `REGIMM && Instr_D[`RT] == `BGEZ) ||
                     ((Instr_D[`OPCODE] == `RCLASS) && (Instr_D[`FUNC] == `JR)) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `JALR);

   //在D级还有1个周期需要使用RS  cal_r cal_i load store
   assign Tuse_RS1 = ((Instr_D[`OPCODE] == `RCLASS) && (Instr_D[`FUNC] == `ADDU)) ||
                     ((Instr_D[`OPCODE] == `RCLASS) && (Instr_D[`FUNC] == `SUBU)) ||
                     (Instr_D[`OPCODE] == `LW) || (Instr_D[`OPCODE] == `SW) || (Instr_D[`OPCODE] == `ORI) || (Instr_D[`OPCODE] == `LUI)
                     || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `ADD) || (Instr_D[`OPCODE] == `ADDI) ||
                     (Instr_D[`OPCODE] == `ADDIU) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SUB) ||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `OR) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `AND)||
                     (Instr_D[`OPCODE] == `ANDI) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `XOR) || (Instr_D[`OPCODE] == `XORI)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `NOR) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLLV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRLV)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRAV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLT) || (Instr_D[`OPCODE] == `SLTI) ||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLTU)|| (Instr_D[`OPCODE] == `SLTIU)|| (Instr_D[`OPCODE] == `SB)||
                     (Instr_D[`OPCODE] == `SH) || (Instr_D[`OPCODE] == `LB) || (Instr_D[`OPCODE] == `LBU) || (Instr_D[`OPCODE] == `LH)||
                     (Instr_D[`OPCODE] == `LHU)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULT)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULTU)||(Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIVU)||(Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MTHI)||
                     ((Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MTLO));

   //在D级还有0个周期需要使用RT   beq
   assign Tuse_RT0 = (Instr_D[`OPCODE] == `BEQ) || (Instr_D[`OPCODE] == `BNE);

   //在D级还有1个周期需要使用RT   cal_r
   assign Tuse_RT1 = ((Instr_D[`OPCODE] == `RCLASS) && (Instr_D[`FUNC] == `ADDU)) ||
                     ((Instr_D[`OPCODE] == `RCLASS) && (Instr_D[`FUNC] == `SUBU)) ||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `ADD) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SUB)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `OR) || (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `AND)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `XOR)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `NOR)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLL)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLLV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRL)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRLV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRA)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SRAV)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLT)|| (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `SLTU)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULT)||(Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `MULTU)||
                     (Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIV)||(Instr_D[`OPCODE] == `RCLASS && Instr_D[`FUNC] == `DIVU);
```

```
153
154     //在D级还有2个周期需要使用RT   store
155     assign Tuse_RT2 = (Instr_D[`OPCODE] == `SW) || (Instr_D[`OPCODE] == `SB) || (Instr_D[`OPCODE] == `SH);
156
157     //产生Stall
158
159     //如果在D级还有0个周期要使用RS，而E级指令仍需1个周期才能产生所要写入的数据
160     assign Stall_RS0_E1 = Tuse_RS0 && (Tnew_E == 2'b01) && (Instr_D[`RS] == A3_E) && RegWrite_E;
161
162     //如果在D级还有0个周期要使用RS，而E级指令仍需2个周期才能产生所要写入的数据
163     assign Stall_RS0_E2 = Tuse_RS0 && (Tnew_E == 2'b10) && (Instr_D[`RS] == A3_E) && RegWrite_E;
164
165     //如果在D级还有0个周期要使用RS，而M级指令仍需1个周期才能产生所要写入的数据
166     assign Stall_RS0_M1 = Tuse_RS0 && (Tnew_M == 2'b01) && (Instr_D[`RS] == A3_M) && RegWrite_M;
167
168     //如果在D级还有1个周期要使用RS，而E级指令仍需2个周期才能产生所要写入的数据
169     assign Stall_RS1_E2 = Tuse_RS1 && (Tnew_E == 2'b10) && (Instr_D[`RS] == A3_E) && RegWrite_E;
170
171     assign Stall_RS = Stall_RS0_E1 || Stall_RS0_E2 || Stall_RS0_M1 || Stall_RS1_E2;
172
173
174     //如果在D级还有0个周期要使用RT，而E级指令仍需1个周期才能产生所要写入的数据
175     assign Stall_RT0_E1 = Tuse_RT0 && (Tnew_E == 2'b01) && (Instr_D[`RT] == A3_E) && RegWrite_E;
176
177     //如果在D级还有0个周期要使用RT，而E级指令仍需2个周期才能产生所要写入的数据
178     assign Stall_RT0_E2 = Tuse_RT0 && (Tnew_E == 2'b10) && (Instr_D[`RT] == A3_E) && RegWrite_E;
179
180     //如果在D级还有0个周期要使用RT，而M级指令仍需1个周期才能产生所要写入的数据
181     assign Stall_RT0_M1 = Tuse_RT0 && (Tnew_M == 2'b01) && (Instr_D[`RT] == A3_M) && RegWrite_M;
182
183     //如果在D级还有1个周期要使用RT，而E级指令仍需2个周期才能产生所要写入的数据
184     assign Stall_RT1_E2 = Tuse_RT1 && (Tnew_E == 2'b10) && (Instr_D[`RT] == A3_E) && RegWrite_E;
185
186     assign Stall_RT = Stall_RT0_E1 || Stall_RT0_E2 || Stall_RT0_M1 || Stall_RT1_E2;
187
188     assign Stall = Stall_RS || Stall_RT || MDStall;
189
190     //产生Forward
191     //跳转指令之后规定不能再接跳转指令
192     //由于对一条指令如SW的RT可能存在多次转发，所以的确得保证优先级问题，最新更新最优先
193     //指令在越后面要用到寄存器的值，所用的值经过的转发更新就越多
194
195     //在D级的指令（beq）需要用到RS，但是如果别的指令在这里需要的话，会提前转发
196     assign ForwardRSD = ((Instr_D[`RS] == A3_M) && (Tnew_M == 0) && (A3_M != 0) && RegWrite_M)? 2 :
197                                 ((Instr_D[`RS] == A3_W) && (A3_W != 0) && RegWrite_W)? 1 :
198                                                                                       0 ;
199     //在D级的指令（beq）需要用到RT
200     assign ForwardRTD = ((Instr_D[`RT] == A3_M) && (Tnew_M == 0) && (A3_M != 0) && RegWrite_M)? 2 :
201                                 ((Instr_D[`RT] == A3_W) && (A3_W != 0) && RegWrite_W)? 1 :
202                                                                                       0 ;
203     //在E级的指令（cal_r cal_i load store）需要用到RS
204     assign ForwardRSE = ((Instr_E[`RS] == A3_M) && (Tnew_M == 0) && (A3_M != 0) && RegWrite_M)? 2 :
205                                 ((Instr_E[`RS] == A3_W) && (A3_W != 0) && RegWrite_W)? 1 :
206                                                                                       0 ;
207     //在E级的指令（cal_r）需要用到RT
208     assign ForwardRTE = ((Instr_E[`RT] == A3_M) && (Tnew_M == 0) && (A3_M != 0) && RegWrite_M)? 2 :
209                                 ((Instr_E[`RT] == A3_W) && (A3_W != 0) && RegWrite_W)? 1 :
210                                                                                       0 ;
211     //在M级的指令（sw）需要用到RT
212     assign ForwardRTM = ((Instr_M[`RT] == A3_M) && (Tnew_M == 0) && (A3_M != 0) && RegWrite_M)? 2 :
213                                 ((Instr_M[`RT] == A3_W) && (A3_W != 0) && RegWrite_W)? 1 :
214                                                                                       0 ;
215
216 endmodule
217
```

（1）模块接口

信号	方向	描述
Start	I	输入来自 E 级部件的 E 级控制器的乘除模块运行开始标志
Busy	I	输入来自 E 级部件的 E 级乘除模块的运行中的信号
Instr_D	I	输入来自 D 级寄存器的指令
Instr_E	I	输入来自 E 级寄存器的指令
Instr_M	I	输入来自 M 级寄存器的指令
Instr_W	I	输入来自 W 级寄存器的指令
RegWrite_E	I	输入来自 E 级的写使能信号
RegWrite_M	I	输入来自 M 级的写使能信号

RegWrite_W	I	输入来自 W 级的写使能信号
A3_E	I	输入来自 E 级的写地址
A3_M	I	输入来自 M 级的写地址
A3_W	I	输入来自 W 级的写地址
Stall	O	输出暂停信号
ForwardRSD	O	输出 RSD 转发控制信号
ForwardRTD	O	输出 RTD 转发控制信号
ForwardRSE	O	输出 RSE 转发控制信号
ForwardRTE	O	输出 RTE 转发控制信号
ForwardRTM	O	输出 RTM 转发控制信号

（2）功能定义

序号	功能名称	功能描述
1	判断暂停	当前 D 级指令所需使用的寄存器的时间周期大于 E、M、W 级相同的寄存器写入指令的产生写入数据时间周期时，需要暂停。有两类寄存器和八种情况 RS0_E1　RS0_E2 RS0_M1　RS1_E2 RT0_E1　RT0_E2 RT0_M1　RT1_E2
2	判断转发	当前 D 级指令所需使用的寄

		存器的时间周期小于或等于E、M、W 级相同的寄存器写入指令的产生写入数据时间周期时，可以转发。有两类寄存器和五种情况 FRSD FRTD FRSE FRTE FRTM

	cal_r		cal_i		store	load					
	addu	subu	ori	lui	sw	lw	beq	jr	jal	nop	j
Tuse_RS0	0	0	0	0	0	0	1	1	X	X	X
Tuse_RS1	1	1	1	1	1	1	0	0	X	X	X
Tuse_RT0	0	0	X	X	0	X	1	X	X	X	X
Tuse_RT1	1	1	X	X	0	X	0	X	X	X	X
Tuse_RT2	0	0	X	X	1	X	0	X	X	X	X
Tnew	1	1	1	1	3	2	3	3	3	3	3

注：Tnew=3 代表不产生写入寄存器数据

IF/ID当前指令			ID/EX(Tnew)/E							EX/MEM(Tnew)/M						MEM/WB(Tnew)/W				
指令类型	源寄存器	Tuse	cal_r 1/rd	cal_i 1/rt	load 2/rt	jal 0/ra	jalr 0/rd	mfhilo 0/rd		cal_r 0/rd	cal_i 0/rt	load 1/rt	jal 0/ra	jalr 0/rd		cal_r 0/rd	cal_i 0/rt	load 0/rt	jal 0/ra	jalr
jr	rs	0	stall	stall	stall			stall				stall								
bgez	rs	0	stall	stall	stall			stall				stall								
blez	rs	0	stall	stall	stall			stall				stall								
bgtz	rs	0	stall	stall	stall			stall				stall								
bltz	rs	0	stall	stall	stall			stall				stall								
bne	rs/rt	0	stall	stall	stall			stall				stall								
beq	rs/rt	0	stall	stall	stall			stall				stall								
cal_r	rs/rt	1			stall															
cal_i	rs	1			stall															
load	rs	1			stall															
store	rs	1			stall															
store	rt	2																		
jalr	rs	0	stall	stall	stall			stall				stall								
sll	rt	1			stall															
srl	rt	1			stall															
sra	rt	1			stall															
mult	rs/rt	1			stall															
multu	rs/rt	1			stall															
div	rs/rt	1			stall															
divu	rs/rt	1			stall															
mthilo	rs	1			stall															

（3）内部部件 DECODER

```
`define OPCODE 31:26
`define FUNC   5:0
`define RS     25:21
`define RT     20:16
`define RD     15:11
`define RCLASS 6'b000000
`define REGIMM 6'b000001

`define LB     6'b100000
`define LBU    6'b100100
`define LH     6'b100001
`define LHU    6'b100101
`define LW     6'b100011
`define SB     6'b101000
`define SH     6'b101001
`define SW     6'b101011
`define ADD    6'b100000
`define ADDU   6'b100001
`define ADDI   6'b001000
`define ADDIU  6'b001001
`define SUB    6'b100010
`define SUBU   6'b100011
`define AND    6'b100100
`define ANDI   6'b001100
`define OR     6'b100101
`define ORI    6'b001101
`define XOR    6'b100110
`define XORI   6'b001110
`define NOR    6'b100111
`define SLLV   6'b000100
`define SLL    6'b000000
`define SRLV   6'b000110
`define SRL    6'b000010
```

```
`define SRAV   6'b000111
`define SRA    6'b000011
`define SLT    6'b101010
`define SLTU   6'b101011
`define SLTI   6'b001010
`define SLTIU  6'b001011
`define LUI    6'b001111
`define BEQ    6'b000100
`define BNE    6'b000101
`define BLEZ   6'b000110
`define BGTZ   6'b000111
`define BLTZ   5'b00000////////
`define BGEZ   5'b00001////////
`define J      6'b000010
`define JAL    6'b000011
`define JR     6'b001000
`define JALR   6'b001001
`define MULT   6'b011000
`define MULTU  6'b011001
`define DIV    6'b011010
`define DIVU   6'b011011
`define MFHI   6'b010000
`define MFLO   6'b010010
`define MTHI   6'b010001
`define MTLO   6'b010011
module DECODER_Tnew_E(//这里产生的Tnew是针对E而言的
   input [31:0] Instr,
   output reg [1:0] Tnew = 2'b11
   );
    always @(*)begin
      case(Instr[`OPCODE])
         `RCLASS:
            case(Instr[`FUNC])
```

```
                `ADDU:begin
                   Tnew = 2'b01;
                end
                `SUBU:begin
                   Tnew = 2'b01;
                end
                `ADD:begin
                   Tnew = 2'b01;
                end
                `SUB:begin
                   Tnew = 2'b01;
                end
                `OR:begin
                   Tnew = 2'b01;
                end
                `AND:begin
                   Tnew = 2'b01;
                end
                `XOR:begin
                   Tnew = 2'b01;
                end
                `NOR:begin
                   Tnew = 2'b01;
                end
                `SLL:begin
                   Tnew = 2'b01;
                end
                `SLLV:begin
                   Tnew = 2'b01;
                end
                `SRL:begin
                   Tnew = 2'b01;
                end

                `SRLV:begin
                   Tnew = 2'b01;
                end
                `SRA:begin
                   Tnew = 2'b01;
                end
                `SRAV:begin
                   Tnew = 2'b01;
                end
                `SLT:begin
                   Tnew = 2'b01;
                end
                `SLTU:begin
                   Tnew = 2'b01;
                end
                `JALR:begin
                   Tnew = 2'b00;
                end
                `MFHI:begin
                   Tnew = 2'b01;
                end
                `MFLO:begin
                   Tnew = 2'b01;
                end
                default:begin//////////////其他指令默认不产生吧
                   Tnew = 2'b11;
                end
             endcase
          `LHU:begin
             Tnew = 2'b10;
          end
          `LH:begin
             Tnew = 2'b10;

          end
          `LBU:begin
             Tnew = 2'b10;
          end
          `LB:begin
             Tnew = 2'b10;
          end
          `SLTIU:begin
             Tnew = 2'b01;
          end
          `SLTI:begin
             Tnew = 2'b01;
          end
          `XORI:begin
             Tnew = 2'b01;
          end
          `ANDI:begin
             Tnew = 2'b01;
          end
          `ADDIU:begin
             Tnew = 2'b01;
          end
          `ADDI:begin
             Tnew = 2'b01;
          end
          `ORI:begin
             Tnew = 2'b01;
          end
          `LUI:begin
             Tnew = 2'b01;
          end
          `LW:begin
             Tnew = 2'b10;
```

```
        end
        `JAL:begin
                Tnew = 2'b00;
        end
      default:begin
         Tnew = 2'b11;
      end
      endcase
    end
endmodule
////////////////////////////////////////////////////////////
module DECODER_Tnew_M(//这里产生的Tnew是针对M而言的
   input [31:0] Instr,
   output reg [1:0] Tnew = 2'b11
   );
    always @(*)begin
      case(Instr[`OPCODE])
         `RCLASS:
            case(Instr[`FUNC])
               `ADDU:begin
                  Tnew = 2'b00;
               end
               `SUBU:begin
                  Tnew = 2'b00;
               end
               `ADD:begin
                  Tnew = 2'b00;
               end
               `SUB:begin
                  Tnew = 2'b00;
               end
               `OR:begin
                  Tnew = 2'b00;
               end
               `AND:begin
                  Tnew = 2'b00;
               end
               `XOR:begin
                  Tnew = 2'b00;
               end
               `NOR:begin
                  Tnew = 2'b00;
               end
               `SLL:begin
                  Tnew = 2'b00;
               end
               `SLLV:begin
                  Tnew = 2'b00;
               end
               `SRL:begin
                  Tnew = 2'b00;
               end
               `SRLV:begin
                  Tnew = 2'b00;
               end
               `SRA:begin
                  Tnew = 2'b00;
               end
               `SRAV:begin
                  Tnew = 2'b00;
               end
               `SLT:begin
                  Tnew = 2'b00;
               end
               `SLTU:begin
                  Tnew = 2'b00;
               end
               `JALR:begin
                  Tnew = 2'b00;
               end
               `MFHI:begin
                  Tnew = 2'b00;
               end
               `MFLO:begin
                  Tnew = 2'b00;
               end
               default:begin//////////////
                  Tnew = 2'b11;
               end
            endcase
         `LHU:begin
            Tnew = 2'b01;
         end
         `LH:begin
            Tnew = 2'b01;
         end
         `LBU:begin
            Tnew = 2'b01;
         end
         `LB:begin
            Tnew = 2'b01;
         end
         `SLTIU:begin
            Tnew = 2'b00;
         end
         `SLTI:begin
            Tnew = 2'b00;
         end
         `XORI:begin
```

```
285             Tnew = 2'b00;
286         end
287         `ANDI:begin
288             Tnew = 2'b00;
289         end
290         `ADDIU:begin
291             Tnew = 2'b00;
292         end
293         `ADDI:begin
294             Tnew = 2'b00;
295         end
296         `ORI:begin
297             Tnew = 2'b00;
298         end
299         `LUI:begin
300             Tnew = 2'b00;
301         end
302         `LW:begin
303             Tnew = 2'b01;
304         end
305         `JAL:begin
306                 Tnew = 2'b00;
307         end
308       default:begin
309         Tnew = 2'b11;
310       end
311       endcase
312     end
313 endmodule
314
```

12、MUX

```
21  module MUX_2_1_32(
22     input [31:0] A,
23     input [31:0] B,
24     input Sel,
25     output [31:0] C
26      );
27      assign C = Sel ? B : A ;
28  endmodule
29
30  module MUX_4_1_32(
31     input [31:0] A,
32     input [31:0] B,
33     input [31:0] C,
34     input [31:0] D,
35     input [1:0] Sel,
36     output [31:0] E
37     );
38     assign E = (Sel == 2'b00)? A :
39                (Sel == 2'b01)? B :
40                (Sel == 2'b10)? C :
41                                D ;
42  endmodule
43
44  module MUX_4_1_5(
45     input [4:0] A,B,C,D,
46     input [1:0] Sel,
47     output [4:0] E
48     );
49     assign E = (Sel == 2'b00)? A :
50                (Sel == 2'b01)? B :
51                (Sel == 2'b10)? C :
52                                D ;
53  endmodule
```

三、测试程序

```
# basic compute
ori $t0 $0 1
ori $t1 $0 2
addu $t2 $t0 $t1
add $t3 $t2 $t1
lui $t4 0xffff
ori $t4 $t4 0xfff0
addu $t5 $t3 $t4
add $t6 $t5 $t4
lui $t0 0x7fff
ori $t0 $t0 0x1234
addiu $t0 $t0 0xffff
```

```
addi $t0 $t0 0xffff
add $t7 $t0 $t6
lui $t8 0xaaaa
addu $t9 $t7 $t8
addiu $t9 $t9 0xffffaaaa
addiu $t9 $t9 -1
addi $t9 $t9 -1
subu $s0 $t9 $t7
subu $s1 $t7 $t9
subu $s2 $s1 $s0
lui $t0 0xffff
ori $t0 $t0 0xfff0
ori $t1 $0 10
sub $s3 $t1 $t0
sub $s4 $t0 $t1
sub $s3 $s4 $s3

# load store

addi $s7 $0 8
addi $t4 $t4 -1
sw $t4 -8($s7)
lb $t0 -8($s7)
addi $t5 $t5 2
sb $t5 -4($s7)
lbu $t1 -7($s7)
lh $t2 -4($s7)
subu $t6 $t6 $s7
sb $t6 -3($s7)
lhu $t3 -4($s7)
sh $t7 -2($s7)
lw $t0 -4($s7)
ori $s0 $s0 1
sw $s0 0($s7)
ori $s1 $s1 2
sh $s1 4($s7)
lw $t1 0($s7)
addi $s2 $s2 -2
sb $s2 6($s7)
lh $t3 6($s7)
sub $s3 $s3 $s7
sb $s3 7($s7)
lbu $t0 7($s7)
```

```
# logic
lui $t0 0xabcd
lui $t1 0xcdef
ori $t0 $t0 0x1234
ori $t1 $t1 0x5678
and $t2 $t0 $t1
or $t3 $t2 $t1
and $t4 $t3 $t2
or $t5 $t4 $t3
andi $t6 $t5 0xa8b6
ori $t7 $t6 0x12b4
xor $t8 $t7 $t6
xor $t9 $t8 $t7
xori $t0 $t9 0x7823
andi $t1 $t0 0xffff
nor $t2 $t1 $t0
nor $t3 $t2 $t1
xori $t4 $t3 0xabcd
nor $t5 $t4 $t3

#branch
ori $t0 $0 1
ori $t1 $0 2
lui $t2 0xffff
ori $t2 $t2 0xffff
sub $t2 $t2 $t0
sub $t2 $t2 $t0
bne $t0 $t1 mark1
lui $t4 0xffff
mark1_back:
ori $s0 $0 1
beq $t3 $t4 mark2
addi $t3 $t3 1
mark2_back:
bltz $t2 mark3
addi $t2 $t2 1
mark3_back:
addi $t2 $t2 1
blez $t2 mark4
subu $t2 $t2 $t1
mark4_back:
bgtz $t2 mark5
add $t2 $t2 $t1
mark5_back:
```

```
bgez $t2 mark6
addi $t2 $t2 1
mark6_back:
j end
nop

mark1:
ori $s0 $0 2
bne $t4 $0 mark1_back
lui $t3 0xffff
mark2:
addi $t4 $t4 1
beq $t3 $t4 mark2_back
nop
mark3:
sub $t2 $t2 $t0
bltz $t2 mark3_back
nop
mark4:
blez $t2 mark4_back
addi $t2 $t2 112
mark5:
bgtz $t2 mark5_back
sub $t2 $t2 $t1
mark6:
bgez $0 mark6_back

end:
#shift

lui $t0 0xfff0
ori $t0 $t0 0xabcd
ori $t1 $0 3
sll $t2 $t0 2
sllv $t2 $t2 $t1
srl $t2 $t2 3
addi $t1 $t1 -1
srlv $t2 $t2 $t1
addi $t1 $t1 3
srlv $t2 $t2 $t1
lui $t3 0x8000
or $t2 $t3 $t2
sra $t2 $t2 4
addi $t1 $t1 -3
```

```
srav $t2 $t2 $t1
lui $t3 0x7fff
ori $t3 $t3 0xffff
and $t2 $t2 $t3
sra $t2 $t2 2
srav $t2 $t2 $t1

#jump
j mark7
ori $s0 $0 0xffff
mark7_back:
jal mark8
nop
ori $s0 $0 0xaaaa
addiu $s3 $s3 8
jr $s3
nop
ori $s0 $0 0xbbbb
j mark9
nop

mark7:
j mark7_back
addu $s1 $s0 $s0
mark8:
addiu $s2 $ra 16
jalr $s3 $s2
lui $s7 0xffff
mark9:
jr $ra
nop

jal mark10
lui $s6 0xffff
mark10:
addi $s5 $ra 16
jalr $s4 $s5
lui $t7 0xbbbb
lui $t9 0xcccc
j end_2
ori $s7 0xffff
lui $t9 0xffff
lui $t8 0xaaaa
jr $s4
```

```
end_2:

#mult_div
ori $t0 $0 3
ori $t1 $0 4
lui $t2 0xffff
ori $t2 $t2 0xfff0
lui $t3 0x7000
ori $t3 $t3 0xffff
lui $t4 0xabcd
ori $t4 $t4 0xefef
mult $t0 $t1
mflo $s0
mthi $t4
mfhi $t4
mult $t2 $t3
mfhi $s1
mflo $s2
multu $t3 $t2
mfhi $s1
mflo $s2
or $s3 $s1 $0
mthi $s2
mtlo $s3
multu $t2 $t4
mfhi $s3
mflo $s4
div $t0 $t1
mfhi $s0
mflo $s1
div $t2 $t0
mfhi $s2
mflo $s3
divu $t2 $t0
mfhi $s2
mflo $s3
div $t3 $t4
mfhi $s2
mflo $s3
mthi $0
mtlo $0
mflo $s3
mfhi $s2
divu $t4 $t3
```

```
mfhi $s0
mflo $s1

#setbit
ori $t0 $0 2
ori $t1 $0 5
lui $t2 0x7000
lui $t3 0xffff
ori $t3 $t3 0xffff
slt $s0 $t0 $t3
slt $s0 $t3 $t0
slt $s0 $t3 $t3
slti $s1 $t2 0x7fff
slti $s1 $t3 0
slti $s1 $t3 -1
sltu $s2 $t0 $t3
sltu $s2 $t3 $t2
sltiu $s3 $t1 0x0fff
sltiu $s3 $t3 1
```

```
 90@00003000: $ 8 <= 00000001
110@00003004: $ 9 <= 00000002
130@00003008: $10 <= 00000003
150@0000300c: $11 <= 00000005
170@00003010: $12 <= ffff0000
190@00003014: $12 <= fffffff0
210@00003018: $13 <= fffffff5
230@0000301c: $14 <= ffffffe5
250@00003020: $ 8 <= 7fff0000
270@00003024: $ 8 <= 7fff1234
290@00003028: $ 1 <= 00000000
310@0000302c: $ 1 <= 0000ffff
330@00003030: $ 8 <= 80001233
350@00003034: $ 1 <= 00000000
370@00003038: $ 1 <= 0000ffff
390@0000303c: $ 8 <= 80011232
410@00003040: $15 <= 80011217
430@00003044: $24 <= aaaa0000
450@00003048: $25 <= 2aab1217
470@0000304c: $25 <= 2aaabcc1
490@00003050: $25 <= 2aaabcc0
510@00003054: $25 <= 2aaabcbf
530@00003058: $16 <= aaa9aaa8
550@0000305c: $17 <= 55565558
570@00003060: $18 <= aaacaab0
590@00003064: $ 8 <= ffff0000
610@00003068: $ 8 <= fffffff0
630@0000306c: $ 9 <= 0000000a
650@00003070: $19 <= 0000001a
670@00003074: $20 <= ffffffe6
690@00003078: $19 <= ffffffcc
710@0000307c: $23 <= 00000008
730@00003080: $12 <= ffffffef
730@00003084: *00000000 <= ffffffef
770@00003088: $ 8 <= ffffffef
790@0000308c: $13 <= fffffff7
790@00003090: *00000004 <= 000000f7
```

```
830@00003094: $ 9 <= 000000ff
850@00003098: $10 <= 000000f7
870@0000309c: $14 <= ffffffdd
870@000030a0: *00000004 <= 0000ddf7
910@000030a4: $11 <= 0000ddf7
910@000030a8: *00000004 <= 1217ddf7
950@000030ac: $ 8 <= 1217ddf7
970@000030b0: $16 <= aaa9aaa9
970@000030b4: *00000008 <= aaa9aaa9
ISim>
# run 1.00us
1010@000030b8: $17 <= 5556555a
1010@000030bc: *0000000c <= 0000555a
1050@000030c0: $ 9 <= aaa9aaa9
1070@000030c4: $18 <= aaacaaae
1070@000030c8: *0000000c <= 00ae555a
1110@000030cc: $11 <= 000000ae
1130@000030d0: $19 <= ffffffc4
1130@000030d4: *0000000c <= c4ae555a
1170@000030d8: $ 8 <= 000000c4
1190@000030dc: $ 8 <= abcd0000
1210@000030e0: $ 9 <= cdef0000
1230@000030e4: $ 8 <= abcd1234
1250@000030e8: $ 9 <= cdef5678
1270@000030ec: $10 <= 89cd1230
1290@000030f0: $11 <= cdef5678
1310@000030f4: $12 <= 89cd1230
1330@000030f8: $13 <= cdef5678
1350@000030fc: $14 <= 00000030
1370@00003100: $15 <= 000012b4
1390@00003104: $24 <= 00001284
1410@00003108: $25 <= 00000030
1430@0000310c: $ 8 <= 00007813
1450@00003110: $ 9 <= 00007813
1470@00003114: $10 <= ffff87ec
1490@00003118: $11 <= 00000000
1510@0000311c: $12 <= 0000abcd
1530@00003120: $13 <= ffff5432
1550@00003124: $ 8 <= 00000001
1570@00003128: $ 9 <= 00000002
1590@0000312c: $10 <= ffff0000
1610@00003130: $10 <= ffffffff
1630@00003134: $10 <= fffffffe
1650@00003138: $10 <= fffffffd
1690@00003140: $12 <= ffff0000
1710@0000317c: $16 <= 00000002
1750@00003184: $11 <= ffff0000
1770@00003144: $16 <= 00000001
1810@0000314c: $11 <= ffff0001
1830@00003188: $12 <= ffff0001
1930@00003154: $10 <= fffffffe
1950@00003194: $10 <= fffffffd
ISim>
# run 1.00us
2030@00003158: $10 <= fffffffe
2090@00003160: $10 <= fffffffc
2150@000031a4: $10 <= 0000006c
2210@00003168: $10 <= 0000006e
2270@000031ac: $10 <= 0000006c
2330@00003170: $10 <= 0000006d
2370@000031b4: $ 8 <= fff00000
2430@000031b4: $ 8 <= fff00000
2450@000031b8: $ 8 <= fff0abcd
2470@000031bc: $ 9 <= 00000003
2490@000031c0: $10 <= ffc2af34
2510@000031c4: $10 <= fe1579a0
2530@000031c8: $10 <= 1fc2af34
2550@000031cc: $ 9 <= 00000002
2570@000031d0: $10 <= 07f0abcd
2590@000031d4: $ 9 <= 00000005
```

```
                2610@000031d8: $10 <= 003f855e
                2630@000031dc: $11 <= 80000000
                2650@000031e0: $10 <= 803f855e
                2670@000031e4: $10 <= f803f855
                2690@000031e8: $ 9 <= 00000002
                2710@000031ec: $10 <= fe00fe15
                2730@000031f0: $11 <= 7fff0000
                2750@000031f4: $11 <= 7fffffff
                2770@000031f8: $10 <= 7e00fe15
                2790@000031fc: $10 <= 1f803f85
                2810@00003200: $10 <= 07e00fe1
                2850@00003208: $16 <= 0000ffff
                2890@00003234: $17 <= 0001fffe
                2910@0000320c: $31 <= 00003214
                2950@00003238: $18 <= 00003224
                2990@0000323c: $19 <= 00003244
ISim>
# run 1.00us
                3010@00003240: $23 <= ffff0000
                3030@00003224: $16 <= 0000bbbb
                3130@00003214: $16 <= 0000aaaa
                3150@00003218: $19 <= 0000324c
                3230@0000324c: $31 <= 00003254
                3250@00003250: $22 <= ffff0000
                3270@00003254: $21 <= 00003264
                3310@00003258: $20 <= 00003260
                3330@0000325c: $15 <= bbbb0000
                3370@00003268: $23 <= ffffffff
                3390@00003278: $ 8 <= 00000003
                3410@0000327c: $ 9 <= 00000004
                3430@00003280: $10 <= ffff0000
                3450@00003284: $10 <= fffffff0
                3470@00003288: $11 <= 70000000
                3490@0000328c: $11 <= 7000ffff
                3510@00003290: $12 <= abcd0000
                3530@00003294: $12 <= abcdefef
                3690@0000329c: $16 <= 0000000c
```

```
                3750@000032a4: $12 <= abcdefef
                3910@000032ac: $17 <= fffffff8
                3930@000032b0: $18 <= fff00010
ISim>
# run 1.00us
                4090@000032b8: $17 <= 7000fff7
                4110@000032bc: $18 <= fff00010
                4130@000032c0: $19 <= 7000fff7
                4370@000032d0: $19 <= abcdefe4
                4390@000032d4: $20 <= 43210110
                4650@000032dc: $16 <= 00000003
                4670@000032e0: $17 <= 00000000
                4930@000032e8: $18 <= ffffffff
                4950@000032ec: $19 <= fffffffb
ISim>
# run 1.00us
                5210@000032f4: $18 <= 00000000
                5230@000032f8: $19 <= 55555550
                5490@00003300: $18 <= 1bceefee
                5510@00003304: $19 <= ffffffff
                5610@00003310: $19 <= 00000000
                5630@00003314: $18 <= 00000000
                5890@0000331c: $16 <= 3bcceff0
                5910@00003320: $17 <= 00000001
                5930@00003324: $ 8 <= 00000002
                5950@00003328: $ 9 <= 00000005
                5970@0000332c: $10 <= 70000000
                5990@00003330: $11 <= ffff0000
ISim>
# run 1.00us
                6010@00003334: $11 <= ffffffff
                6030@00003338: $16 <= 00000000
                6050@0000333c: $16 <= 00000001
                6070@00003340: $16 <= 00000000
                6090@00003344: $17 <= 00000000
                6110@00003348: $17 <= 00000001
                6130@0000334c: $17 <= 00000000
```

```
                6150@00003350: $18 <= 00000001
                6170@00003354: $18 <= 00000000
                6190@00003358: $19 <= 00000001
                6210@0000335c: $19 <= 00000000
ISim>
# run 1.00us
```

四、思考题

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

实际中乘除法运算有延时，并且在我们的 CPU 设计中也模拟了该延迟，但是正常的 ALU 运算不模拟延迟，所以如果乘除部件整合进 ALU 的话会增加 CPU 的工作延迟，如果设置单独的乘法部件的话，在进行乘除运算的同时不会影响其他运算指令的运行，能提高 CPU 工作效率，还有就是乘除部件整合进 ALU 的话，还会增加 ALU 的控制信号和复杂度，非常不便。

设置独立的 HI、LO 寄存器是因为，通用寄存器堆中的每一个寄存器都可以被读写，如果 HI、LO 混进通用寄存器堆的话，很容易产生乘除数据被修改的意外情况。其次，如果 HI、LO 寄存器混进通用寄存器堆的话，就会造成乘除指令要进行写入寄存器的操作，从而会影响到其他写回寄存器指令的运行，还得设置相应的暂停机制。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

MIPS 的延迟槽是伴随着流水线的结构而产生的。实验中考虑更多的是分支延迟槽，只有分支指令进入 ID 级执行后，下一条指令的地址才能被获知，假若无延迟槽，在等待下一条指令地址的时候必须让流水线暂停一个周期，而有了延迟槽之后，在等待下一条指令地址的过程中可以执行分支指令的下一条指令，这样做的好处是提前假设分支不发生，执行了下

一条指令，提高效率，还有就是即使分支发生了，分支指令的下一条指令也会被执行，提高了指令运行的效率。

3. 为何上文文末提到的 lb 等指令使用的数据扩展模块应该在 MEM/WB 之后，而不能在 DM 之后？

在实际情况中，DM 的写入一般是先写入到 cache 中，然后再向主存储器写入，即只要写入 cache，向主存储器写入的过程与 CPU 时钟周期，但是 DM 的读取，最坏情况下是从主存储器中读取数据，这就意味着，平均情况下 DM 的写入要比读取快。所以，假若把数据扩展模块放在 DM 之后，读取数据在 M 级会经过 DM 和扩展部件，会使读取的时钟周期变长，进而会增加流水线的 CPU 时钟周期。但是如果放在 MEM/WB 之后的话，LOAD 类指令在 M 级的执行时间不变，在 W 级写入寄存器时不影响后续存取指令在 M 级的执行，所以大概率不会增加 CPU 时钟周期。

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

C 语言中每个字符占一字节的存储空间，如果按字访问的话，在运行 C 程序时为避免字符串的读取错误，势必会将一个字符存储在一个字内，造成巨大的空间浪费。所以相比于按字存储，按字节存储则方便和节省空间得多。

5. 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

我觉得我的设计风格更偏向于第一种“侦测者”型。从产生数据冲突的原因入手，展示了数据冲突产生的逻辑所在，代码短小，很多时候加指令时不用改动。

对抗复杂性：一、在总体设计中多抽象了一层不同的流水级部件 F、D、E、M、W 级，这样做的好处是将不同的流水级部件封装到一个模块中，只保留其与其他模块的接口，从而其内部的连线不会对其他模块产生影响，使顶层连线更为简单和清晰。二、在设计流水线信号时会在信号后面加上下划线和流水级的缩写，表明这个信号所处的流水线级。三、使用宏定义避免大量输入 01 识别码而产生的输入错误。

6. 你对流水线 CPU 设计风格有何见解？

在设计暂停和转发时，按逻辑进行设计会更加抓住 CPU 的本质，但是不直观；按指令类型进行设计，直观但是又略显冗杂。

7. 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。（非常重要）

序号	类型	Tuse_RS	Tuse_RT	Tnew_E	冲突寄存器	解决方案	测试程序
1	R-J	0		1	RS	暂停 1 个时钟周期后从 M 到 D 转发	ori $ra $0,0x0000300c jr $ra
2	L-J	0		2	RS	暂停 2 个时钟周期后从 W 到 D 转发	lw $ra,0($0) jr $ra

3	R-B	0	0	1	RS 或 RT	暂停 1 个时钟周期后从 M 到 D 转发	addu $t1,$t1,$t2 beq $t1,$t2,loop
4	L-B	0	0	2	RS 或 RT	暂停两个时钟周期后从 W 到 D 转发	lw $t1,0($0) beq $t1,$t2,loop
5	R-R	1	1	1	RS 或 RT	从 M 到 E 转发	addu $t1,$t1,$t2 subu $t1,$t1,$t2
6	L-R	1	1	2	RS 或 RT	暂停一个时钟周期后从 W 到 E 转发	lw $t1,0($0) addu $t1,$t1,$t2
7	R-R	1		1	RS	从 M 到 E 级转发	ori $t1,$0,1 lui $t1,1
8	L-R	1		2	RS	暂停一个时钟周期后从 W 到 E 转发	lw $t1,0($0) lui $t1,1
9	R-S	1	2	1	RS	从 M 到 E 级转发	lui $t1,1 sw $t1,0($0)
10	L-S	1	2	2	RS	暂停 1 个周期后从 W 到 E 级转发	lw $t1,0($0) sw $t2,0($t1)
11	L-S	1	2	2	RT	从 W 到 M 级转发	lw $t1,0($0) sw $t1,0($0)

序号	类型	Tuse_RS	Tuse_RT	Tnew_M	冲突寄存器	解决方案	测试程序
1	L-R-J	0		1	RS	暂停一个周期后从 W 到 D 级转发	lw $ra,0($0) lui $t2,1 jr $ra
2	L-R-B	0	0	1	RT 或 RT	暂停一个周期后从 W 到 D 级转发	lw $t1,0($0) lui $t2,1 beq $t1,$t2,loop

3	L-R-R	1	1	1	RS 或 RT	从 W 到 E 级转发	lw $t1,0($0)
							lui $t2,1
							addu $t1,$t3,$t1

4	L-R-R	1		1	RS	从 W 到 E 级转发	lw $t1,0($0) lui $t2,1

							ori $t1,$t1,1

5	L-R-S	1	2	1	RS	从 W 到 E 级转发	lw $t1,0($0)
							lui $t2,1
							sw $t1,0($0)

6	R-R-J	0		0	RS	从 M 级到 D 级转发	addu $ra,$t1,$t2 lui $t2,1 jr $ra
7	R-R-B	0	0	0	RS 或 RT	从 M 级到 D 级转发	addu $t1,$t1,$t2 lui $t3,1 beq $t1,$t2,loop

序号	类型	Tuse_RS	Tuse_RT	Tnew_W	冲突寄存器	解决方案	测试程序
1	R-N-N-J	0		0	RS	从 W 到 D 级转发	addu $ra,$t1,$t2 nop nop jr $ra
2	R-N-N-B	0	0	0	RS 或 RT	从 W 到 D 级转发	addu $t1,$t1,$t2 nop nop beq $t1,$t2,loop