中国科学院大学计算机组成原理实验课

实 验 报 告

学号： 2020K8009926006 姓名： 游昆霖 专业： 计算机科学与技术

实验序号： 5.3 实验名称： 增强功能型处理器设计

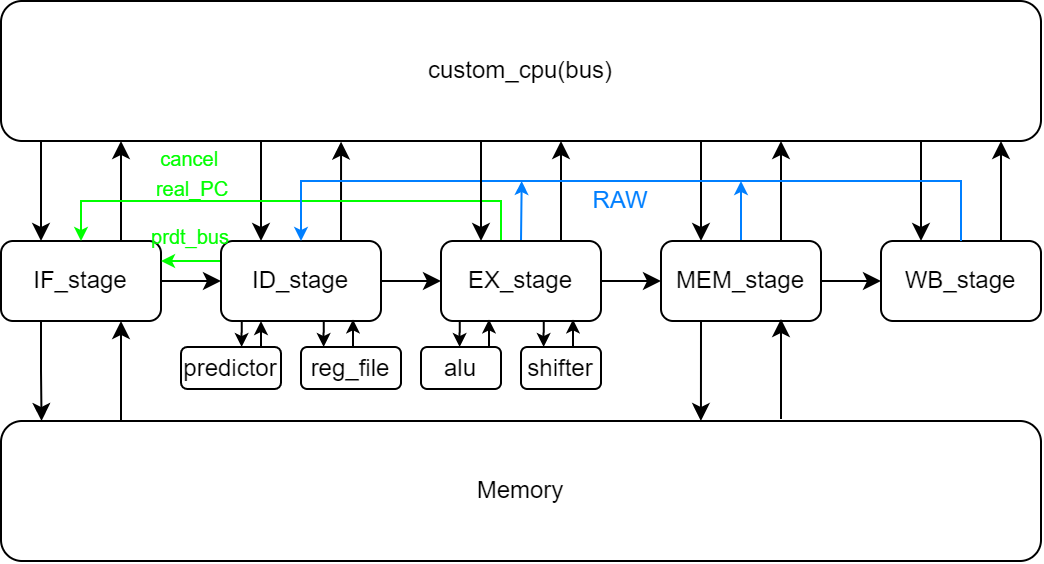
注1：撰写此Word格式实验报告后以PDF格式保存在~/COD-Lab/reports目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为1至4的阿拉伯数字。例如：prj1.pdf。PDF文件大小应控制在5MB以内。此外，实验项目5包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目5讲义。

注2：使用git add及git commit命令将实验报告PDF文件添加到本地仓库master分支，并通过git push推送到GitLab远程仓库master分支（具体命令详见实验报告）。

注3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

1. 逻辑电路结构与仿真波形的截图及说明（比如关键RTL代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

\*说明：本次实验基于RISCV指令集实现，结构为五级流水线（IF、ID、EX、MEM、WB）,支持分支预测功能，由于每个模块的实现方式和prj4类似，以下仅列出数据流向，并结合波形对模块逻辑关键部分进行说明。  
 数据流向示意图：



上图中模块间数据传递均通过custom\_cpu进行传递，为强调起见，将关于写后读（RAW）效应的数据通路使用蓝色在上图中进行表明，将关于分支预测的数据通路使用绿色在上图中进行表明。

1、 模块间数据传递  
 五级流水的每个模块均设计了相应代码，并用custom\_cpu.v文件进行了封装。前一级给后一级的数据通过总线(AA\_to\_BB\_bus)进行传递，且当AA\_to\_BB\_valid和BB\_ready信号同时为高时传输数据有效。  
 特别的，考虑到分支预测。将由ID\_stage.v向IF\_stage.v传递预测跳转地址和预测跳转结果prdt\_bus。由EX\_stage.v向IF\_stage.v传递预测失败信号cancel和PC正确结果real\_PC以保证预测失败后流水线的正确执行。  
 考虑到单发射流水线中的写后读效应，分别从EX、MEM、WB阶段向ID阶段进行数据前馈，且优先级EX>MEM>WB。考虑到例化reg\_file.v文件时，同时连接读写端口，将WB写向寄存器的值通过WB\_to\_RF\_bus前递至ID解读。

2、IF阶段  
 该部分需要实现考虑分支预测时下一条指令地址PC的获取和已取得有效指令有效性的判断。以下为关键代码部分。  
 （1）状态机

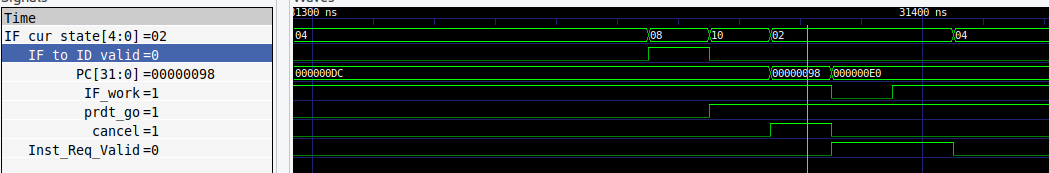
|  |
| --- |
| always @(\*) begin                  case (IF\_cur\_state)                          `RST : begin                                          IF\_next\_state = `IF;                          end                          `IF : begin                                  if(cancel)                                          IF\_next\_state = `IF;                                  else if(Inst\_Req\_Valid & Inst\_Req\_Ready)                                          IF\_next\_state = `IW;                                  else                                          IF\_next\_state = `IF;                          end                          `IW : begin                                  if(cancel)                                          IF\_next\_state = `IF;                                  else if(Inst\_Ready & Inst\_Valid)                                          IF\_next\_state = `RDS;                                  else                                          IF\_next\_state = `IW;                          end                          `RDS : begin                                  if(cancel)                                          IF\_next\_state = `IF;                                  else if(ID\_ready)                                          IF\_next\_state = `SDD;                                  else                                          IF\_next\_state = `RDS;                          end                          `SDD : begin                                  if(ID\_ready)                                          IF\_next\_state = `IF ;                                  else                                          IF\_next\_state = `SDD;                          end                          default: begin                                          IF\_next\_state = `RST;                          end                  endcase          end |

IF阶段的状态机共设置RST、IF、IW、RDS、SDD五个阶段，其中RDS阶段表示准备传输数据至ID阶段，SDD表示传输数据完毕。  
 （2）控制信号

|  |
| --- |
| //Signal show whether this IF\_pipeline really work          always @(posedge clk) begin                  if(rst)                          IF\_work <= 1'b0;                  else begin                          if(cancel)                                  IF\_work <= 1'b0;                          else if(IF\_ready)                                  IF\_work <= 1'b1;                  end          end  //Control Unit          assign IF\_ready = ~IF\_work | (IF\_done & ID\_ready);          assign IF\_done  = IF\_cur\_state ==`SDD & ~cancel; |

由于需要进行分支预测，设置IF\_work信号表示本条指令的取指是否有效。下一次IF阶段开始工作的条件是上一次分支预测失败，或上一个取指正确且被ID阶段接收。同时，为保证数据传输有效性，设置本阶段完成信号。  
 （3）PC更新逻辑

|  |
| --- |
| always @(posedge clk) begin                  if(rst)                          PC <= 32'b0;                  else if(cancel)  //CHECK the COND                          PC <= real\_PC;                  else if(IF\_done & ID\_ready) begin                          PC <= prdt\_go ? prdt\_tar : PC +4;                  end          end |

IF阶段在RDS状态将本条指令的PC传向ID阶段，从而在SDD阶段按照ID阶段前馈的分支预测信号进行更新。当需要进行分支预测的指令流转至EX阶段时，将得到该预测是否准确，并前递cancel信号和正确的PC。若分支预测失败，则PC需要再次进行更新以维护流水线的正确执行。  
 仿真波形说明：

上图中IF模块在RDS阶段（08）向ID模块传递信息并拉高有效信号，在SDD阶段（10）获得ID模块前馈的信息，由于预测进行跳转，在SDD阶段进行了PC的跳转更新，在下一个IF阶段（02）接受到EX阶段前馈的cancel信号，即分支预测失败，此时暂时拉低本模块操作有效信号IF\_work，拉低Inst\_Req\_Valid，设置次态仍为IF，根据前馈信息将PC更新为正确值，再进行取指及恢复IF\_work。

（4）数据传递

|  |
| --- |
| assign IF\_to\_ID\_valid = IF\_cur\_state ==`RDS & ID\_ready & ~cancel & IF\_work;  assign IF\_to\_ID\_bus ={                         PC              ,       //63:32                         Instruction\_Reg         //31:0                         }; |

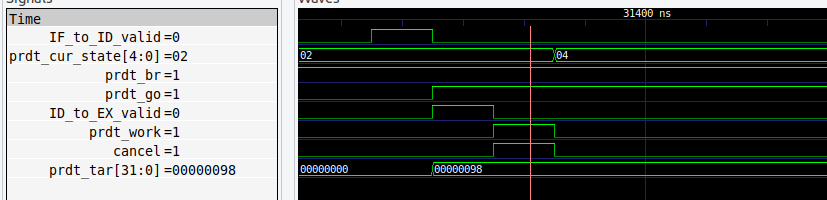
IF阶段需要向ID阶段传输的数据及控制信号如上所示。特别的，设置IF\_to\_ID\_valid信号在RDS阶段拉高，从而可在SDD阶段得到ID前馈的是否跳转和跳转结果，如果分支预测失败，可以在下一个IF阶段得到EX前馈的cancel信号和真实PC。  
  
 3、ID阶段  
 该部分需要实现指令译码，分支预测，和寄存器堆Reg\_file的交互，写后读的处理，以及数据传递。  
 （1）指令译码  
 和prj4所实现基本一致，对IF阶段传输过来的指令进行译码即可。  
 （2）分支预测  
 predictor.v

|  |
| --- |
| `include "turbo\_macro.v"  module predictor(          input   clk,          input   rst,            input   Branch, //type is branch or jump          input   prdt\_work,//limit the state shift to one clk          //the signal is equal to cancel hold          //signal of not taken          input   cancel,          output  prdt\_br  );          reg [4:0] prdt\_cur\_state;          reg [4:0] prdt\_next\_state;          always @ (posedge clk) begin                  if(rst)                          prdt\_cur\_state <= `RST ;                  else                          prdt\_cur\_state <= prdt\_next\_state ;          end            //note that state will not change unless Branch          always @ (\*) begin                  case (prdt\_cur\_state)                          `RST : begin                                  if(rst)                                          prdt\_next\_state = `RST ;                                  else                                          prdt\_next\_state = `S\_Taken ;                          end                          `S\_Taken : begin                                  if(prdt\_work & cancel & Branch)                                          prdt\_next\_state = `W\_Taken ;                                  else                                          prdt\_next\_state = `S\_Taken ;                          end                          `W\_Taken : begin                                  if(prdt\_work & cancel & Branch)                                          prdt\_next\_state = `W\_NTaken ;                                  else if (prdt\_work & ~cancel & Branch)                                          prdt\_next\_state = `S\_Taken ;                                  else                                          prdt\_next\_state = `W\_Taken ;                          end                          `W\_NTaken : begin                                  if(prdt\_work & cancel & Branch)                                          prdt\_next\_state = `S\_NTaken ;                                  else if (prdt\_work & ~cancel & Branch)                                          prdt\_next\_state = `W\_Taken ;                                  else                                          prdt\_next\_state = `W\_NTaken ;                          end                          `S\_NTaken : begin                                  if(prdt\_work & cancel & Branch)                                          prdt\_next\_state = `S\_NTaken ;                                  else if(prdt\_work & ~cancel & Branch)                                          prdt\_next\_state = `W\_NTaken ;                                  else                                          prdt\_next\_state = `S\_NTaken ;                          end                          default:                                  prdt\_next\_state = `RST;                  endcase          end            assign  prdt\_br = (prdt\_cur\_state == `S\_Taken) | (prdt\_cur\_state == `W\_Taken); |

Custom\_cpu.v中相关通路

|  |
| --- |
| //predict Unit          assign prdt\_go  = (I\_jalr | J\_type) | (B\_type & prdt\_br);          assign jalr\_tar = rs1\_value + I\_imm;          assign prdt\_tar =       B\_type ? ID\_PC + B\_imm :                                  J\_type ? ID\_PC + J\_imm :                                  I\_jalr ? {jalr\_tar[31:1] ,1'b0} :                                  0; //default          assign prdt\_bus = {prdt\_go , prdt\_tar};          always @(posedge clk) begin                  if(ID\_to\_EX\_valid)                          prdt\_work <= 1'b1;                  else                          prdt\_work <= 1'b0;          end          predictor prdt\_inst (                  .clk            (clk),                  .rst            (rst),                  .Branch         (B\_type),                  .prdt\_work      (prdt\_work),                  .cancel         (cancel),                  .prdt\_br        (prdt\_br)          ); |

为提高分支预测精度，使用2位分支预测。为控制1次分支预测只产生一次分支预测状态转移，限制cancel只维持一拍（EX阶段完成），且设置prdt\_work表示cancel有效的周期。  
 仿真波形说明：



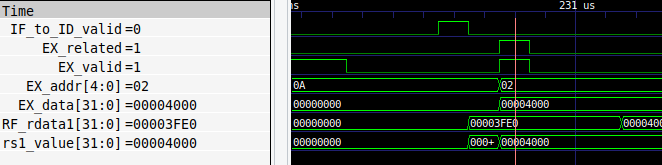
Prdt\_br根据此时预测器的状态确定，当预测器处于强接受状态（02）或弱接受状态（04）时该信号拉高。当ID译码得到无条件跳转指令，或条件跳转指令且prdt\_br为高时拉高prdt\_go信号，表示分支预测需要进行跳转。由于EX可在一个阶段内给出运算结果，在ID\_to\_EX\_valid拉高的下一个周期拉高prdt\_work信号，表示cancel信号实际作用，并令预测器根据cancel信号进行状态转移。如上图，由于cancel信号在对应周期拉高，令预测器从强接受状态转移至弱接受状态。  
 （3）与寄存器堆Reg\_file的交互

|  |
| --- |
| //Channel to regfile          //read data from ID, write data from WB          assign RF\_raddr1 = rs1;          assign RF\_raddr2 = rs2;          assign {WB\_RF\_wen , WB\_RF\_waddr , WB\_RF\_wdata} = WB\_to\_RF\_bus;            reg\_file reg\_file\_inst(                  .clk(clk),                  .raddr1(RF\_raddr1),                  .raddr2(RF\_raddr2),                  .rdata1(RF\_rdata1),                  .rdata2(RF\_rdata2),                  .waddr(WB\_RF\_waddr),                  .wdata(WB\_RF\_wdata),                  .wen(WB\_RF\_wen)          ); |

由于读写寄存器堆均在ID阶段进行，因此将隐退指令WB阶段的写寄存器信号传至ID阶段，作为寄存器堆写端口数据，将本条指令译码所得地址作为寄存器堆读地址。  
 （4）写后读的处理

|  |
| --- |
| //register value considering pipeline          assign {EX\_load , EX\_valid , EX\_addr , EX\_data} = EX\_fw\_bus;          assign {MEM\_load , MEM\_done , MEM\_valid , MEM\_addr , MEM\_data} = MEM\_fw\_bus;          assign {WB\_valid , WB\_addr , WB\_data} = WB\_fw\_bus;            assign rs1\_value =  ~(|rs1)                     ? 32'b0         :                          EX\_valid  & (EX\_addr  == rs1)   ? EX\_data       :                          MEM\_valid & (MEM\_addr == rs1)   ? MEM\_data      :                          WB\_valid  & (WB\_addr  == rs1)   ? WB\_data       :                          RF\_rdata1;          assign rs2\_value =  ~(|rs2)                     ? 32'b0         :                          EX\_valid  & (EX\_addr  == rs2)   ? EX\_data       :                          MEM\_valid & (MEM\_addr == rs2)   ? MEM\_data      :                          WB\_valid  & (WB\_addr  == rs2)   ? WB\_data       :                          RF\_rdata2;    //Control Unit          assign EX\_related = (|EX\_addr) & EX\_valid & (rs1==EX\_addr | rs2==EX\_addr);          assign MEM\_related = (|MEM\_addr) & MEM\_valid & (rs1==MEM\_addr | rs2==MEM\_addr);            assign block    = ( EX\_load & EX\_related ) | (MEM\_load & MEM\_related & ~MEM\_done);          assign ID\_done  = ~block & ~cancel;  always @(posedge clk) begin                  if(rst)                          ID\_work <= 1'b0;                  else begin                          if(cancel)                                  ID\_work <= 1'b0;                          else if(ID\_ready)                                  ID\_work <= IF\_to\_ID\_valid;                  end          end            assign ID\_ready = ~ID\_work | (ID\_done & EX\_ready);          assign ID\_to\_EX\_valid = ID\_work & ID\_done & EX\_ready; |

考虑写后读效应，从寄存器堆对应地址取出来的值可能会被之前指令的后续阶段改变，因此需要由EX、MEM和WB阶段进行数据前递。通过地址和有效信号判断前递数据是否和寄存器读数据相关，并选择得到该寄存器读地址对应的正确数据，选择优先级为EX>MEM>WB，与本条指令约接近则优先级越高。特别的，当EX阶段或MEM阶段的指令为load类型且数据相关时，需要进行阻塞，等待该条指令执行完访存操作后才可得到正确数据，从而继续运行流水线。  
 仿真波形说明：



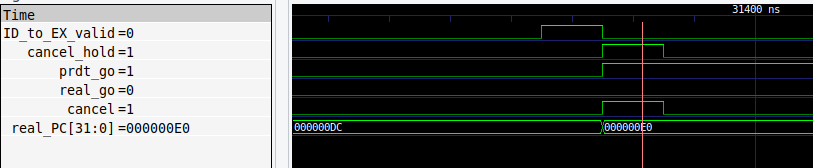
光标所示位置，由于EX传输的信号有效时，其写地址和寄存器堆的读地址相同，因此考虑写后读效应，使用EX传递的写数据而非直接从寄存器堆取出的数据表示rs1号寄存器的数据。

（5）数据传递

|  |
| --- |
| //to EX bus          //send rs2\_value to get Write\_data later          assign R\_Icalc  = R\_type | I\_calc;          assign load     = I\_load;          assign store    = S\_type;          assign jump     = I\_jalr | J\_type;          assign RF\_wen   = R\_type | I\_type | J\_type | U\_type;          assign RF\_waddr = rd;          assign RF\_ID\_alter = (I\_jalr | J\_type) | U\_lui | U\_auipc ;          assign RF\_ID\_alter\_data =       I\_jalr | J\_type ? ID\_PC +4 :                                          U\_lui           ? U\_imm         :                                          U\_auipc         ? ID\_PC + U\_imm :                                          0; //default          assign ID\_to\_EX\_bus = {                                    ID\_PC,                //247:216                                    prdt\_go,              //215:215                                    prdt\_tar,             //214:183                                    ALU\_op,               //182:182                                    ALU\_A,                //181:150                                    ALU\_B,                //149:118                                    Shifter\_op,           //117:116                                    Shifter\_A,            //115:84                                    Shifter\_B,            //83:79                                    B\_type,               //78:78                                    R\_Icalc,              //77:77                                    load,                 //76:76                                    store,                //75:75                                    jump,                 //74:74                                    funct3,               //73:71                                    RF\_wen,               //70:70                                    RF\_waddr,             //69:65                                    RF\_ID\_alter,          //64:64                                    RF\_ID\_alter\_data,     //63:32                                    rs2\_value             //31:0                                    }; |

ALU、Shifter操作数和控制信号与多周期处理器类似。为节约传输数据的位宽，根据后续阶段的操作类型对已有指令进行简单分类，且将ID阶段可确定的寄存器堆写数据进行选择后再传输。  
  
 4、EX阶段  
 该阶段需要与ALU、Shifter进行交互，判断分支预测是否成功并给出真实PC，前递本条指令写寄存器堆的相关信息，向后传递访存相关数据及当前确定的写寄存器堆相关数据。  
 （1）ALU、Shifter的交互  
 将ID阶段传递的控制信号和操作数传递至对应组件即可，与多周期类似。  
 （2）判断分支预测并进行相应操作

|  |
| --- |
| //Test predict result , cancel if fail          assign br\_en = B\_type & ( ( ~funct3[2] & ~funct3[0] &  ALU\_Zero )       |                                    ( ~funct3[2] &  funct3[0] & ~ALU\_Zero)        |                                    (  funct3[2] & ~funct3[0] &  ALU\_Result[0])   |                                    (  funct3[2] &  funct3[0] & ~ALU\_Result[0])                                  );          assign real\_go = br\_en | jump;          always @(posedge clk) begin                  if(ID\_to\_EX\_valid)                          cancel\_hold <= 1'b1;                  else                          cancel\_hold <= 1'b0;          end          assign cancel = ( real\_go ^ prdt\_go ) & cancel\_hold;    //set high if not equal          assign real\_PC = real\_go ? prdt\_tar : EX\_PC +4; |

根据ALU所计算的结果获取分支预测是否成功，将真实跳转结果与预测结果进行异或及得到了cancel信号，cancel为高表示分支预测失败需要重启流水线。同时设置cancel\_hold信号，保证cancel只拉高一拍，从而保证流水线和分支预测器的正确操作。  
 仿真波形说明：  


此处根据ALU计算的结果获取了真实的跳转结果real\_go，和预测结果prdt\_go不同，即分支预测失败，应当拉高cancel。设置cancel\_hold延后ID\_to\_EX\_valid一拍拉高，从而使得cancel只拉高一拍。  
 （3）前递数据  
 考虑写后读效应，将本条指令是否写寄存器、写地址及EX当前所得数据向ID阶段前递。同时传递本条指令是否为load类型，如是且数据相关，下一条指令的ID阶段将阻塞至本条指令流向WB并访存获得数据。  
 （4）向后传递

|  |
| --- |
| always @(posedge clk) begin                  if(rst)                          EX\_work <= 1'b0;                  else begin                          if(EX\_ready)                                  EX\_work <= ID\_to\_EX\_valid;                  end          end            assign EX\_done = 1'b1; //always done in one cycle          assign EX\_ready = ~EX\_work | (EX\_done & MEM\_ready);          assign EX\_to\_MEM\_valid = EX\_work & EX\_done & MEM\_ready;  //send MEM bus          assign EX\_to\_MEM\_bus = {                                  EX\_PC,                  //145:114                                  load,                   //113:113                                  store,                  //112:112                                  Address,                //111:80                                  load\_tag,               //79:78                                  funct3,                 //77:75                                  Write\_strb,             //74:71                                  Write\_data,             //70:39                                  RF\_wen,                 //38:38                                  RF\_waddr,               //37:33                                  RF\_EX\_alter,            //32:32                                  RF\_EX\_alter\_data        //31:0                                  }; |

向后传递写寄存器堆、访存和隐退指令的相关信号。

5、MEM阶段  
 该阶段需要实现访存及相关数据的处理，前递写寄存器堆相关数据，向后传递最终写寄存器堆相关信号和隐退指令信号。  
 （1）访存状态机

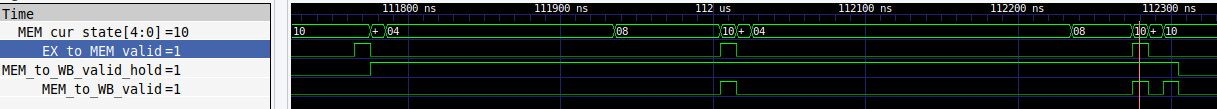
|  |
| --- |
| always @(\*) begin                  case(MEM\_cur\_state)                          `RST: begin                                  MEM\_next\_state = `BSL;                          end                            `BSL: begin                                  if(MEM\_work) begin                                          if(load | store)                                                  MEM\_next\_state = `SL;                                          else                                                  MEM\_next\_state = `SLD;                                  end                                  else                                          MEM\_next\_state = `BSL;                          end                            `SL: begin                                  if(MEM\_work) begin                                          if(load & Mem\_Req\_Ready)                                                  MEM\_next\_state = `RDW;                                          else if(store & Mem\_Req\_Ready )                                                  MEM\_next\_state = `SLD;                                          else                                                  MEM\_next\_state = `SL;                                  end                                  else                                          MEM\_next\_state = `SL;                          end                            `RDW: begin                                  if(Read\_data\_Ready & Read\_data\_Valid)                                          MEM\_next\_state = `SLD;                                  else                                          MEM\_next\_state = `RDW;                          end                            `SLD: begin                                  if(EX\_to\_MEM\_valid) //WB ready                                          MEM\_next\_state = `BSL;                                  else                                          MEM\_next\_state = `SLD;                          end                            default: begin                                  MEM\_next\_state = `RST;                          end                  endcase          end |

整体流程和多周期访存部分的状态转移类似，额外考虑了模块传输数据的有效信号。对于访存读写的数据，操作基本和多周期处理器类似。  
 （2）前递写寄存器堆信号

与EX阶段类似，但本阶段需要额外添加MEM\_done，表示访存是否完成，所取得的相关数据是否有效。  
 （3）向后传递数据

|  |
| --- |
| always @(posedge clk) begin          if(rst)                  MEM\_work <= 1'b0;          else begin                  if(MEM\_ready)                          MEM\_work <= EX\_to\_MEM\_valid;          end  end  always @(posedge clk) begin          if(rst)                  MEM\_to\_WB\_valid\_hold <= 1'b0;          else begin                  if(EX\_to\_MEM\_valid)                          MEM\_to\_WB\_valid\_hold <= 1'b1;                  else if(MEM\_to\_WB\_valid)                          MEM\_to\_WB\_valid\_hold <= 1'b0;          end  end  assign MEM\_done = MEM\_cur\_state == `SLD ;  assign MEM\_ready = ~MEM\_work | (MEM\_done & WB\_ready);  assign MEM\_to\_WB\_valid = MEM\_work & MEM\_done & WB\_ready & MEM\_to\_WB\_valid\_hold; |

MEM\_work表示本次MEM访存有效，MEM\_done表示访存结束，WB\_ready表示WB阶段数据准入，且同时设置MEM\_to\_WB\_valid\_hold保证MEM\_to\_WB\_valid只能拉高一拍。以上信号共同作用，保证了后续隐退信号最高位写使能信号只拉高一拍。  
 仿真波形说明：



当EX\_to\_MEM\_valid拉高时，将MEM\_to\_WB\_valid\_hold拉高，当MEM\_to\_WB\_valid拉高是，将其拉低，从而保证MEM不会重复将相同的隐退指令传递至WB阶段。特别的，当EX\_to\_MEM\_valid和MEM\_to\_WB\_valid同时为高的时候，表示EX已经向MEM传递了新的数据，不需担心重复隐退的问题，此时为使前一条指令成功隐退，仍需保持MEM\_to\_WB\_valid\_hold为高。

1. WB阶段  
    该阶段需要实现隐退指令相关信号的处理、向ID前馈寄存器堆写信号。  
    （1）隐退指令相关信号的处理

|  |
| --- |
| always @(posedge clk) begin          if(rst)                  WB\_work <= 1'b0;          else begin                  if(WB\_ready)                          WB\_work <= MEM\_to\_WB\_valid;          end  end  assign WB\_done = 1'b1;  //always done in one clk  assign WB\_ready = ~WB\_work | WB\_done;  //inst retire          //hold inst\_retire\_valid to one cycle to count valid inst          //hold inst\_retire\_wen to one cycle to avoid repeated comparison  assign inst\_retire\_valid = WB\_done & WB\_work ;  assign inst\_retire\_wen = inst\_retire\_valid & WB\_RF\_wen ;  assign inst\_retire = {inst\_retire\_wen , RF\_waddr , RF\_wdata,WB\_PC}; |

inst\_retire\_valid表示该条指令成功隐退，inst\_retire\_wen则表示需要写寄存器的指令成功隐退，用于进行比对。  
 （2）前馈信号的处理

和EX阶段类似，且为最终写入寄存器堆的数据。  
  
 7、性能计数器  
 与多周期处理器计数对象相同，分散在各模块中，并基于各个模块的对应条件进行略微改动。  
 RISCV指令集下多周期处理器和五级流水线处理器（未加cache）性能比对如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 处理器 | 时钟周期数 | 指令总数 | 访存延迟周期数 | CPI |
| 15pz | 多周期 | 324238832 | 5224477 | 90004486 | 62.06 |
| 流水线 | 236498270 | 5224477 | 94278671 | 45.27 |
| bf | 多周期 | 23775610 | 452850 | 3956925 | 52.50 |
| 流水线 | 19655008 | 452850 | 3947242 | 43.40 |
| dinic | 多周期 | 906059 | 16687 | 171982 | 54.30 |
| 流水线 | 723066 | 16687 | 172866 | 43.33 |
| fib | 多周期 | 110676382 | 2549521 | 183886 | 43.41 |
| 流水线 | 110479201 | 2549521 | 185795 | 43.33 |
| md5 | 多周期 | 236203 | 4911 | 22927 | 48.10 |
| 流水线 | 212774 | 4911 | 22781 | 43.33 |
| qsort | 多周期 | 483070 | 9476 | 65405 | 50.98 |
| 流水线 | 410526 | 9476 | 66011 | 43.32 |
| queen | 多周期 | 4284106 | 81486 | 637557 | 52.57 |
| 流水线 | 3532577 | 81486 | 640402 | 43.35 |
| sieve | 多周期 | 456038 | 10191 | 14335 | 44.75 |
| 流水线 | 441542 | 10191 | 14452 | 43.33 |
| ssort | 多周期 | 27431823 | 619041 | 514622 | 44.31 |
| 流水线 | 26922713 | 619041 | 554778 | 43.49 |
| 平均结果 | 多周期 | 54720902.56 | 996515.56 | 10619125 | 50.33 |
| 流水线 | 44319519.67 | 996515.56 | 11098110.89 | 43.57 |

由上图可见，经过流水线对组件的复用，使不同指令可以进行并行计算，整体时钟周期数和CPI均有了一定的降低，但降低幅度仍未符合流水线的预期。进一步对访存周期进行统计对比，可见访存延迟仍占用了大量时间周期，降低了指令并行度，阻碍了流水线运行效率的提高，需要在后续实验中加入高速缓存cache进行进一步优化。

1. 性能优化  
    由于在性能评测方法发布时，流水线和cache均已经实现，因此将二者作为整体进行代码优化，且由于将时钟频率修改为300hz时，未加cache的流水线bit-gen任务中WNS均大于0.1，而在加上cache后有部分bit-gen任务出现WNS小于0的情形，且查看报告可见关键路径与流水线处理器的架构无关，出现在cache部分，因此主要对后续cache进行结构优化的考虑，详见cache部分报告。
2. 实验过程中遇到的问题、对问题的思考过程及解决方法（比如RTL代码中出现的逻辑bug，逻辑仿真和FPGA调试过程中的难点等）

问题1：隐退指令比对问题  
 根据仿真报错发现多条指令重复与第一条写寄存器堆隐退指令进行比较导致出错。询问芦溶民助教本次实验中比对机制以及查看波形，发现WB直接使用寄存器堆写使能信号作为隐退指令最高位，拉高了多个周期造成隐退异常。  
 设置了inst\_retire\_valid和inst\_retire\_wen等信号，保证了隐退指令最高位只拉高一拍，从而正确进行比对。  
  
问题2：分支预测与PC更新逻辑  
 PC更新出现分支预测指令之前额外更新、分支预测失败后恢复错误、和指令对应错误等问题，通过反复查看波形，进行了多处更改。

首先，以倒数第二个状态RDS作为IF向ID传输指令有效信号，从而在最后一个状态SDD可以得到ID阶段前馈的分支预测结果和地址，并在此状态进行分支预测更新或顺序更新。  
 其次，根据以上状态设置，可以在下一次流程的IF状态获得EX阶段前馈的分支预测是否成功信号cancel以及真实PC，如果分支预测失败，则需设置次态仍为IF，并重新进行取指。  
 另外，为保证指令内存响应正确，Inst\_Req\_Valid同样需要考虑cancel指令，被取消的PC不进行取指，防止PC和Instruction不对应的问题。  
  
问题3：IF阶段堵塞、预测状态转移异常  
 查看波形过程中，发现当分支预测失败时，IF阶段会产生阻塞，且2位分支预测将直接从强接受状态转移至强拒绝状态，不符合预期。  
 设置cancel\_hold信号，控制cancel信号只维持一拍，同时对分支预测器添加相关通路，保证仅在分支预测时改变，且一次分支预测只进行一次状态转移，从而进一步提高分支预测精度，增加流水线效率。  
  
问题4：IF阶段和MEM阶段的状态转移

查看波形中发现数据传递出错，出现遗漏或重复的现象。  
 设置IF状态机末态转初态条件为ID\_ready，因为此时该条指令可被向后传递，防止取得的指令由于ID阶段的阻塞导致遗漏。且此时状态机才满足PC更新条件。

设置MEM状态机末态转初态条件为EX\_to\_MEM\_valid，这是为了保证MEM的PC更新，使得WB阶段隐退的PC不会重复写。且由于WB阶段操作只需1周期即可完成，WB\_ready始终为高，所以不需考虑数据遗漏问题。   
  
问题5：内存旁路访问请求冲突问题  
 medium组max和select-sort任务在本地仿真和云平台仿真均无法结束，直至运行超时。但所有任务均可通过fpga\_eval及硬件仿真，且在后续添加icache和dcache后这两个任务的仿真也可正常通过。

查看框架代码cpu\_test\_top.v 可知CPU的inst\_req\_valid和Memread将通过inst\_if\_wrapper和mem\_if\_wrapper转换为cpu\_inst\_arvalid和cpu\_mem\_arvalid。而在cpu\_to\_mem\_axi\_2x1\_arb.v中二者是并列条件，只有一个有效。因此取指和访存不能同时进行，将MemRead信号传输至IF模块，在指令请求时进行考虑。

1. 对讲义中思考题（如有）的理解和回答

本次实验无思考题。

1. 在课后，你花费了大约\_\_\_\_30\_\_\_\_小时完成此次实验。
2. 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

心得感受：

相比其余选做实验，本实验难度较大，需要对五级流水不同模块的功能和模块之间的数据传递有较为深刻的理解。分支预测的应用和写后读效应的存在使得本实验对信号传递的时序精确度提出了更高的要求，以保证PC能够正确更新、寄存器堆数据能够正确获取、流水线能够正确而高效的工作和重启。另外，在进行基于RISCV指令集实现的多周期处理器和流水线处理器性能对比时，发现由于访存延迟过高，流水线频繁阻塞，性能和多周期相差不大，需要后续实验添加高速缓存cache以进一步提高处理器性能。  
致谢：  
 感谢常轶松老师和陈欲晓在硬件仿真工具方面提供的帮助，感谢芦溶民助教对隐退指令比对机制的讲解，感谢刘士祺助教对模块间接口规范的介绍。