实验 7 报告

学号：2017K80099290013　2017K8009929008
姓名：
箱子号：72

一、实验任务（10%）

1、 在流水线中增加转移指令 BGEZ、BGTZ、BLEZ、BLTZ、J、BLTZAL、BGEZAL、JALR；

2、 在流水线中增加访存指令 LB、LBU、LH、LHU、LWL、LWR、SB、SH、SWL、SWR；

二、实验设计（40%）

（一）总体设计思路

总体：指令实现

1、 转移指令实现：

* 可复用通路部分：在通路控制信号的判断上加入相应指令；

* 不可复用部分：根据转移指令的需求，在 ID 阶段增加传向 IF 阶段控制取指的逻辑；

2、 访存指令实现：

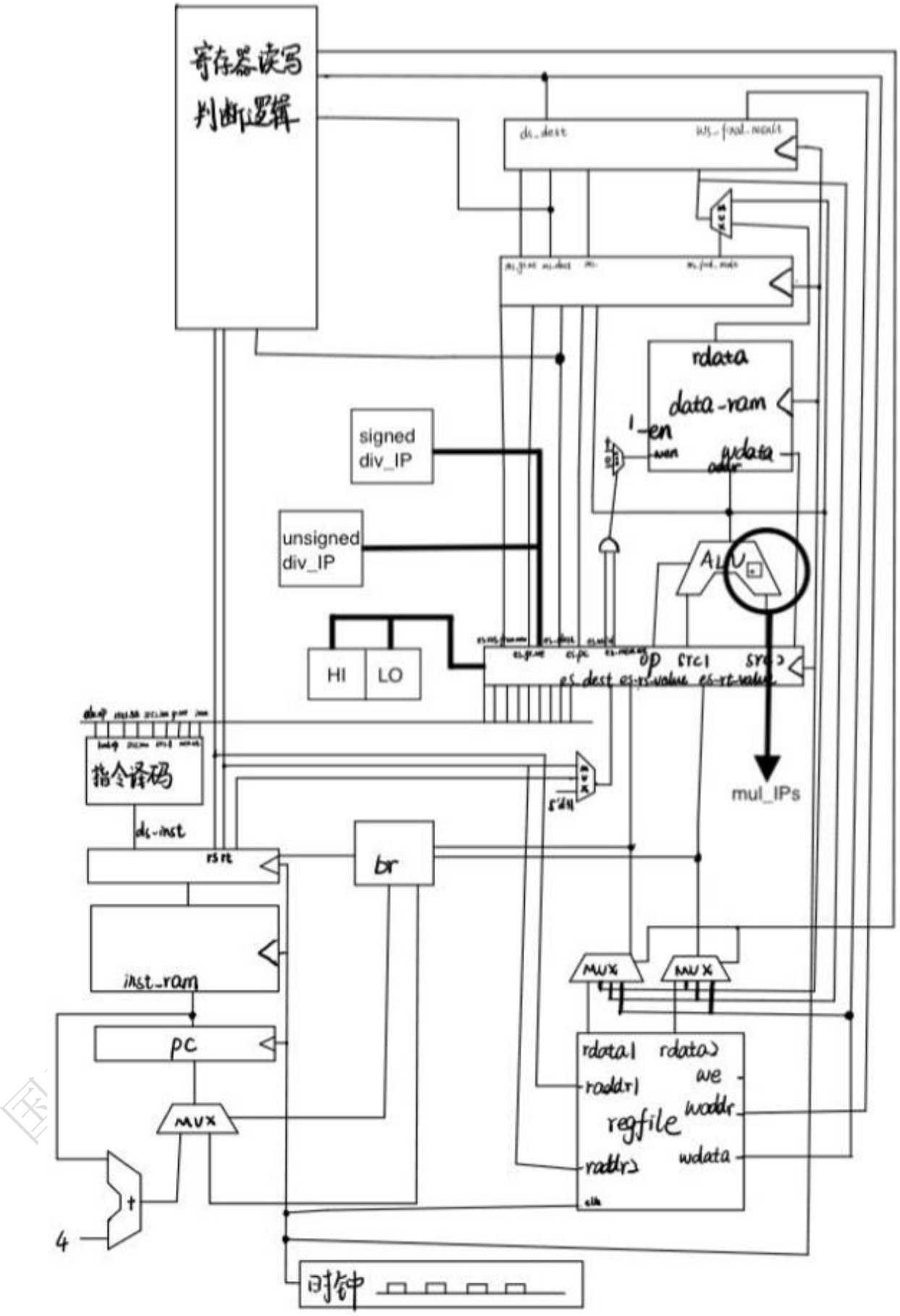
* 可复用通路部分： 在通路控制信号的判断上加入相应指令；

* load 指令：在 EXE 阶段判断需要写入寄存器的位数，在 MEM 阶段根据需要选择从内存中读出数据的位数，在 regfile 中根据 we 决定写入寄存器的位数；

* store 指令：在 EXE 阶段判断需要写入内存的位数和数据；

* 前递问题：在 ID 阶段判断是否需要前递，若需要，根据要写入寄存器的位数将要前递的数值和原寄存器的值进行拼接，得到前递结果；

五级流水的结构设计图如下：


寄存器读写
判断逻辑
rdata
data_ram
signed
div_IP
unsigned
div_IP
HI
LO
ALU
mul_IPs
指令译码
br
inst_ram
pc
MUX
MUX
MUX
rdata1
rdata2
raddr1
regfile
waddr
wdata
raddr2
we
4
时钟

图一：流水线结构图

（二）重要模块设计：ID 阶段：取指判断逻辑＋前递判断

1、部分代码

```
assign br_taken = (   (inst_beq  &&  rs_eq_rt)
                   || (inst_bne  && !rs_eq_rt)
                   || (inst_bgez && (~rs_value[31]))
                   || (inst_bgtz && ((~rs_value[31])&(|rs_value)))
                   || ( inst_blez && (rs_value[31]|(~(|rs_value)) ) )
                   || (inst_bltz && (rs_value[31]))
                   || (inst_bltzal && (rs_value[31]))
                   || (inst_bgezal && (~rs_value[31]))
                   || inst_jal
                   || inst_jr
                   || inst_jalr
                   || inst_j
                  ) && ds_valid;
assign br_target = (inst_beq |inst_bne |inst_bgez |inst_bgtz|inst_blez
                    |inst_bltz|inst_bltzal|inst_bgezal) ? (fs_pc + {{14{imm[15]}}, imm[15:0], 2'b0}) :
                   (inst_jr|inst_jalr)                  ? rs_value :
                  /*inst_jal*/              {fs_pc[31:28], jidx[25:0], 2'b0};
```

图二：取指判断逻辑

```
assign rs_value8_ =(exe_pre1_)?(exe_value[7:0]&{8{exe_wen[0]}}):
                   (  (mem_pre1_)?(mem_value[7:0]&{8{mem_wen[0]}}):
                    ( ( wb_pre1_) ?(rf_wdata[7:0]&{8{wb_wen}}):
                       rf_rdata1[7:0]
                    ));
assign rs_value16_ =(exe_pre1_)?(exe_value[15:8]&{8{exe_wen[0]}}):
                   (  (mem_pre1_)?(mem_value[15:8]&{8{mem_wen[0]}}):
                    ( ( wb_pre1_) ?(rf_wdata[15:8]&{8{wb_wen}}):
                       rf_rdata1[15:8]
                    ));
assign rs_value24_ =(exe_pre1_)?(exe_value[23:16]&{8{exe_wen[0]}}):
                   (  (mem_pre1_)?(mem_value[23:16]&{8{mem_wen[0]}}):
                    ( ( wb_pre1_) ?(rf_wdata[23:16]&{8{wb_wen}}):
                       rf_rdata1[23:16]
                    ));
assign rs_value32_ =(exe_pre1_)?(exe_value[31:24]&{8{exe_wen[0]}}):
                   (  (mem_pre1_)?(mem_value[31:24]&{8{mem_wen[0]}}):
                    ( ( wb_pre1_) ?(rf_wdata[31:24]&{8{wb_wen}}):
                       rf_rdata1[31:24]
                    ));
assign rt_value8_ =(exe_pre2_)?(exe_value[7:0]&{8{exe_wen[0]}}):
                   (  (mem_pre2_)?(mem_value[7:0]&{8{mem_wen[0]}}):
                    ( ( wb_pre2_) ?(rf_wdata[7:0]&{8{wb_wen}}):
                       rf_rdata2[7:0]
                    ));
assign rt_value16_ =(exe_pre2_)?(exe_value[15:8]&{8{exe_wen[0]}}):
                   (  (mem_pre2_)?(mem_value[15:8]&{8{mem_wen[0]}}):
                    ( ( wb_pre2_) ?(rf_wdata[15:8]&{8{wb_wen}}):
                       rf_rdata2[15:8]
                    ));
assign rt_value24_ =(exe_pre2_)?(exe_value[23:16]&{8{exe_wen[0]}}):
                   (  (mem_pre2_)?(mem_value[23:16]&{8{mem_wen[0]}}):
                    ( ( wb_pre2_) ?(rf_wdata[23:16]&{8{wb_wen}}):
                       rf_rdata2[23:16]
                    ));
assign rt_value32_ =(exe_pre2_)?(exe_value[31:24]&{8{exe_wen[0]}}):
                   (  (mem_pre2_)?(mem_value[31:24]&{8{mem_wen[0]}}):
                    ( ( wb_pre2_) ?(rf_wdata[31:24]&{8{wb_wen}}):
                       rf_rdata2[31:24]
                    ));
assign rs_value = {rs_value32_,rs_value24_,rs_value16_,rs_value8_};
assign rt_value = {rt_value32_,rt_value24_,rt_value16_,rt_value8_};
```

图三：前递判断逻辑

2、功能描述：

* 根据指令和数值条件判断是否需要指令跳转；
* 根据指令计算出需要跳转时的下一条指令的地址；
* 在原有的前递逻辑上增加与数据位数有关的判断，将前递来的数据和寄存器中原有的数据合理拼接；

（三）重要模块设计：EXE 阶段：访存位数判断＋store 指令数据写入

1、部分代码：

```
assign data_sram_wen     = {4{es_mem_we&&es_valid}} &
                           ( ( {4{swl_op}} & ((last_addr==0)?4'b0001:( (last_addr==1)? 4'b0011: ( (last_addr==2)?4'b0111:4'b1111 ) )) ) |
                             ( {4{swr_op}} & ((last_addr==0)?4'b1111:( (last_addr==1)? 4'b1110: ( (last_addr==2)?4'b1100:4'b1000 ) )) ) |
                             ( {4{sb_op}}  & ((last_addr==0)?4'b0001:( (last_addr==1)? 4'b0010: ( (last_addr==2)?4'b0100:4'b1000 ) )) ) |
                             ( {4{sh_op}}  & ((last_addr==2)?4'b1100:4'b0011) ) |
                             ( {4{sw_op}}  & 4'b1111)
                           );
assign load_four_wen     =
                           ( ( {4{lwl_op}} & ((last_addr==0)?4'b1000:( (last_addr==1)? 4'b1100: ( (last_addr==2)?4'b1110:4'b1111 ) )) ) |
                             ( {4{lwr_op}} & ((last_addr==0)?4'b1111:( (last_addr==1)? 4'b0111: ( (last_addr==2)?4'b0011:4'b0001 ) )) )
                           );

assign four_wen = es_valid?(es_gr_we?((lwl_op|lwr_op)?load_four_wen: 4'b1111):4'b0000):4'b0000;
```

图四：访存位数判断

```
assign data_sram_wdata = sh_op?{2{es_rt_value[15:0]}}:
                         (sb_op?{4{es_rt_value[7:0]}}:
                         (swl_op?swl_data:
                         (swr_op?swr_data:
                         es_rt_value)));
```

图五：store 写入数据

2、功能描述：

* 根据不同的 store 或 load 指令得到相应的访存位数；
* 根据不同的 store 指令得到写入内存中的数据；

（四）重要模块设计：MEM 阶段：得到 load 指令访存结果

1、部分代码：

```
assign lbu_data=((last_addr==0)?{24'd0,{data_sram_rdata[7:0]}}:
            ((last_addr==1)?({24'd0,{data_sram_rdata[15:8]}}):
            ((last_addr==2)?({24'd0,{data_sram_rdata[23:16]}}):
            {24'd0,data_sram_rdata[31:24]} ) ));

assign lb_data= ((last_addr==0)?{{24{data_sram_rdata[7]}},data_sram_rdata[7:0]}:
                ((last_addr==1)?({{24{data_sram_rdata[15]}},data_sram_rdata[15:8]}):
            ((last_addr==2)?({{24{data_sram_rdata[23]}},data_sram_rdata[23:16]}):
            {{24{data_sram_rdata[31]}},data_sram_rdata[31:24]} ) ));

assign lh_data= (last_addr==2)?{{16{data_sram_rdata[31]}},data_sram_rdata[31:16]}:
            {{16{data_sram_rdata[15]}},data_sram_rdata[15:0]};

assign lhu_data=(last_addr==2)?{16'd0,data_sram_rdata[31:16]}:
            {16'd0,data_sram_rdata[15:0]};


assign lwl_data=((last_addr==0)?{4{data_sram_rdata[7:0]}}:
            ((last_addr==1)?({2{data_sram_rdata[15:0]}}):
            ((last_addr==2)?({{data_sram_rdata[23:0]},8'd0}):
            data_sram_rdata[31:0] ) ));

assign lwr_data=((last_addr==0)?data_sram_rdata[31:0]:
            ((last_addr==1)?({8'd0,data_sram_rdata[31:8]}):
            ((last_addr==2)?({2{data_sram_rdata[31:16]}}):
            {4{data_sram_rdata[31:24]}} ) ));

assign mem_result = lb_op?lb_data:
                    (lbu_op?lbu_data:
                    (lh_op?lh_data:
                    (lhu_op?lhu_data:
                    (lwl_op?lwl_data:
                    (lwr_op?lwr_data:
                    data_sram_rdata)))));
```

图六：load 访存结果

2、功能描述：

* 因为不同 load 指令需要内存数据的位数不同，所以对内存中取出的数据进行拼接操作，得到最终 load 指令的访存结果

（五）重要模块设计：regfile：确定写入寄存器的值

1、部分代码：

```
//WRITE
always @(posedge clk) begin
    if ((waddr!=0)&(we[0] || we[1] || we[2] || we[3])) begin
        rf[waddr][ 7: 0] <= we[0] ? wdata[ 7: 0] : rf[waddr][ 7: 0];
        rf[waddr][15: 8] <= we[1] ? wdata[15: 8] : rf[waddr][15: 8];
        rf[waddr][23:16] <= we[2] ? wdata[23:16] : rf[waddr][23:16];
        rf[waddr][31:24] <= we[3] ? wdata[31:24] : rf[waddr][31:24];
    end
end
```

图七：写入寄存器的

2、功能描述：

* 根据 load 指令要写寄存器的位数设置寄存器写使能 we；

* 根据写使能 we 的每一位是否拉高判断是否向寄存器中写入新的数据；

三、实验过程（50%）

（一）实验流水账

10 月 17 日下午：阅读任务书，更新环境，整理实验思路；

10 月 18 日：逻辑设计，代码实现，仿真调试，上板运行；

10 月 19 日晚上：小组交流代码实现和报告内容；

10 月 20 日上午：写报告；

（二）错误记录

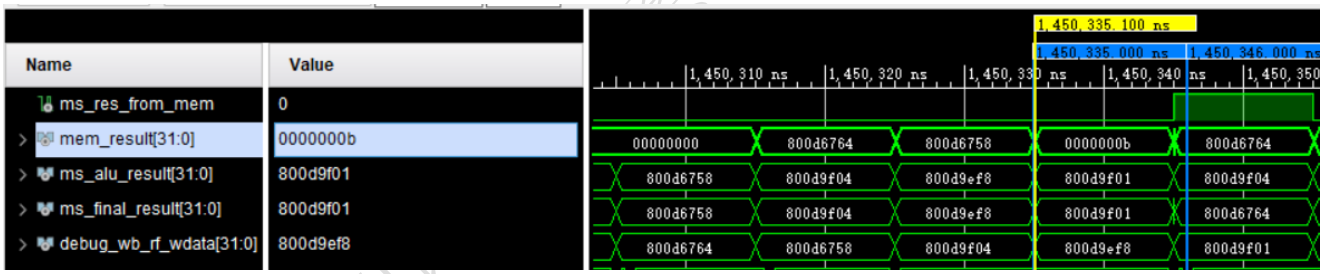
1、错误 1：load_op 赋值时未完全覆盖 load 指令

错误现象：仿真报错：

```
[1450347 ns] Error!!!
    reference: PC = 0xbfc6e0a4, wb_rf_wnum = 0x02, wb_rf_wdata = 0x0000000b
    mycpu    : PC = 0xbfc6e0a4, wb_rf_wnum = 0x02, wb_rf_wdata = 0x800d9f01
```

图八：错误现象 1

错误查找过程：



图九：错误波形 1

发现对应的上一级流水中的 mem_result 的值应该传入 WB 阶段，但是选择了 ms_alu_result 对应的值，ms_res_from_mem 信号错误，EXE 阶段传入的 es_load_op 出现错误；

找到错误原因：load_op 出现错误，没有考虑新加入的访存指令；

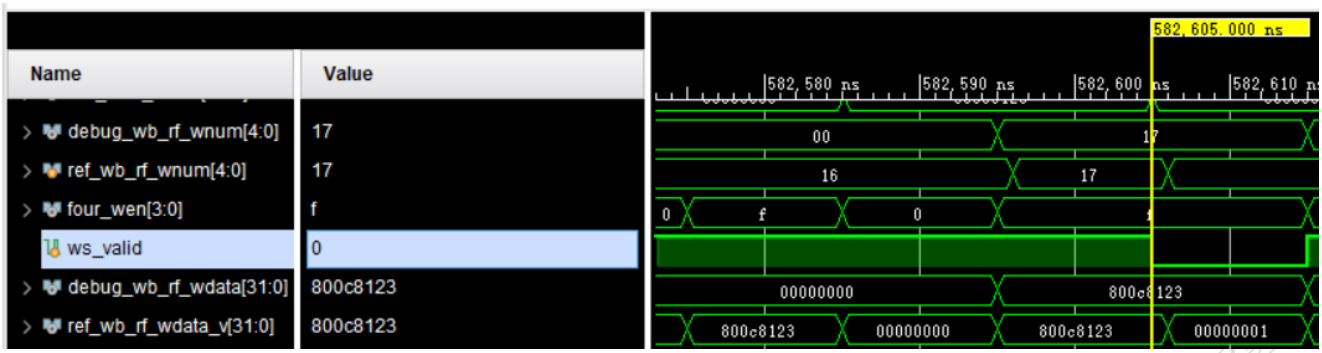
修改方法：如下图所示，考虑新加入的访存指令：

```
assign load_op      = inst_lw|inst_lwr|inst_lwl|inst_lb|inst_lbu|inst_lh|inst_lhu ;
```

图十：修正方法 1

2、错误２：WB 阶段写使能信号出错

错误现象：仿真报错：



图十一：错误现象 2

错误查找过程：

发现写入地址改变且写入数据，但此时 ws_valid 为 0，不应该有写入操作，考虑 debug_wb_rf_wen 信号出错；

```
assign debug_wb_rf_wen   = four_wen;
```

图十二：错误原因 2

修正方法：如下图所示，在 debug_wb_rf_wen 处考虑 valid 的值：

```
assign debug_wb_rf_wen   = four_wen&{4{ws_valid}};
```

图十三：修正方法 2