Prj 6 报告

组员：
武弋 2022K8009929002
王泽黎 2022K8009929011
张家玮 2022K8009929010
箱子号：16

一、实验任务

Exp17：

1. 设计 TLB 模块。

2. 利用 TLB 模块级验证环境对所设计的 TLB 进行验证，通过仿真和上板验证。

Exp18：

1. 将实践任务 17 完成的 TLB 模块集成到实践任务 16 完成的 CPU 中。

2. 在 CPU 中增加 TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB 指令。

3. 在 CPU 中增加 TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY CSR 寄存器。

4. 在采用 AXI 总线的 SoC 验证环境里完成 exp18 对应 func 的功能验证，要求成功通过仿真和上板验证。

Exp19：

1. 为 CPU 增加 TLB 相关异常：TLB 重填例外、load/store/取指操作页无效例外、页修改例外、页特权等级不合规例外。

2. 在 CPU 中增加 DMWCSR 寄存器。

3. 为 CPU 增加虚实地址映射的功能。

4. 在采用 AXI 总线的 SoC 验证环境里完成 exp19 对应 func 的功能验证，要求成功通过仿真和上板验证。

二、实验设计

（一）Exp17 设计：

1. 为 TLB 模块添加读操作和写操作。

TLB 模块中读操作和写操作的实现与 CPU 中 regfile 的逻辑设计和 Verilog 代码实现类似，此处不做赘述。

```
//写操作
always @(posedge clk) begin
    if (we) begin
        tlb_vppn[w_index] <= w_vppn;
        tlb_ps[w_index] <= w_ps;
```

```
        tlb_asid[w_index] <= w_asid;
        tlb_g[w_index] <= w_g;
        tlb_ppn0[w_index] <= w_ppn0;
        tlb_plv0[w_index] <= w_plv0;
        tlb_mat0[w_index] <= w_mat0;
        tlb_d0[w_index] <= w_d0;
        tlb_v0[w_index] <= w_v0;
        tlb_ppn1[w_index] <= w_ppn1;
        tlb_plv1[w_index] <= w_plv1;
        tlb_mat1[w_index] <= w_mat1;
        tlb_d1[w_index] <= w_d1;
        tlb_v1[w_index] <= w_v1;
    end
end

//读操作
assign r_e = tlb_e[r_index];
assign r_vppn = tlb_vppn[r_index];
assign r_ps = tlb_ps[r_index];
assign r_asid = tlb_asid[r_index];
assign r_g = tlb_g[r_index];
assign r_ppn0 = tlb_ppn0[r_index];
assign r_plv0 = tlb_plv0[r_index];
assign r_mat0 = tlb_mat0[r_index];
assign r_d0 = tlb_d0[r_index];
assign r_v0 = tlb_v0[r_index];
assign r_ppn1 = tlb_ppn1[r_index];
assign r_plv1 = tlb_plv1[r_index];
assign r_mat1 = tlb_mat1[r_index];
assign r_d1 = tlb_d1[r_index];
assign r_v1 = tlb_v1[r_index];
```

2. 为 TLB 模块添加查找操作。

TLB 模块要支持取指和访存两个部分的虚实地址转换需求，即两部分都需要对 TLB 模块进行查找，且两部分对应的查找功能一致。查找时，需要向 TLB 模块输入 s_vppn、s_va_bit12 和 s_asid 信息，TLB 模块输出的信息包含 s_found、s_ppn、s_ps、s_plv、s_mat、s_d 和 s_v。其中输入的 s_vppn 来自访存虚地址的 31..13 位，s_va_bit12 来自访存虚地址的 12 位，s_asid 来自 CSR.ASID 的 ASID 域。TLB 输出的 s_ppn 和 s_ps 用于产生最终的物理地址，s_found 的结果用于判定是否产生 TLB 重填异常，s_found 和 s_v 结果用于判定是否产生页无效异常，s_found 和 s_plv 用于判定是否产生页特权等级不合规异常，s_found、s_v 和 s_d 结果用于判定是否产生页修改异常。

在实现电路的时候，我们同时比较所有项，通过组合逻辑产生一个 16 位宽的查找结果 match[15:0]

```
assign match0[ 0] = (s0_vppn[18:9]==tlb_vppn[ 0][18:9])
                  && (tlb_ps4MB[ 0] || s0_vppn[8:0]==tlb_vppn[ 0][8:0])
```

```
                    && ((s0_asid==tlb_asid[ 0]) || tlb_g[ 0]);
assign match0[ 1] = (s0_vppn[18:9]==tlb_vppn[ 1][18:9])
                    && (tlb_ps4MB[ 1] || s0_vppn[8:0]==tlb_vppn[ 1][8:0])
                    && ((s0_asid==tlb_asid[ 1]) || tlb_g[ 1]);
assign match0[ 2] = (s0_vppn[18:9]==tlb_vppn[ 2][18:9])
                    && (tlb_ps4MB[ 2] || s0_vppn[8:0]==tlb_vppn[ 2][8:0])
                    && ((s0_asid==tlb_asid[ 2]) || tlb_g[ 2]);
assign match0[ 3] = (s0_vppn[18:9]==tlb_vppn[ 3][18:9])
                    && (tlb_ps4MB[ 3] || s0_vppn[8:0]==tlb_vppn[ 3][8:0])
                    && ((s0_asid==tlb_asid[ 3]) || tlb_g[ 3]);
assign match0[ 4] = (s0_vppn[18:9]==tlb_vppn[ 4][18:9])
                    && (tlb_ps4MB[ 4] || s0_vppn[8:0]==tlb_vppn[ 4][8:0])
                    && ((s0_asid==tlb_asid[ 4]) || tlb_g[ 4]);
assign match0[ 5] = (s0_vppn[18:9]==tlb_vppn[ 5][18:9])
                    && (tlb_ps4MB[ 5] || s0_vppn[8:0]==tlb_vppn[ 5][8:0])
                    && ((s0_asid==tlb_asid[ 5]) || tlb_g[ 5]);
assign match0[ 6] = (s0_vppn[18:9]==tlb_vppn[ 6][18:9])
                    && (tlb_ps4MB[ 6] || s0_vppn[8:0]==tlb_vppn[ 6][8:0])
                    && ((s0_asid==tlb_asid[ 6]) || tlb_g[ 6]);
assign match0[ 7] = (s0_vppn[18:9]==tlb_vppn[ 7][18:9])
                    && (tlb_ps4MB[ 7] || s0_vppn[8:0]==tlb_vppn[ 7][8:0])
                    && ((s0_asid==tlb_asid[ 7]) || tlb_g[ 7]);
assign match0[ 8] = (s0_vppn[18:9]==tlb_vppn[ 8][18:9])
                    && (tlb_ps4MB[ 8] || s0_vppn[8:0]==tlb_vppn[ 8][8:0])
                    && ((s0_asid==tlb_asid[ 8]) || tlb_g[ 8]);
assign match0[ 9] = (s0_vppn[18:9]==tlb_vppn[ 9][18:9])
                    && (tlb_ps4MB[ 9] || s0_vppn[8:0]==tlb_vppn[ 9][8:0])
                    && ((s0_asid==tlb_asid[ 9]) || tlb_g[ 9]);
assign match0[10] = (s0_vppn[18:9]==tlb_vppn[10][18:9])
                    && (tlb_ps4MB[10] || s0_vppn[8:0]==tlb_vppn[10][8:0])
                    && ((s0_asid==tlb_asid[10]) || tlb_g[10]);
assign match0[11] = (s0_vppn[18:9]==tlb_vppn[11][18:9])
                    && (tlb_ps4MB[11] || s0_vppn[8:0]==tlb_vppn[11][8:0])
                    && ((s0_asid==tlb_asid[11]) || tlb_g[11]);
assign match0[12] = (s0_vppn[18:9]==tlb_vppn[12][18:9])
                    && (tlb_ps4MB[12] || s0_vppn[8:0]==tlb_vppn[12][8:0])
                    && ((s0_asid==tlb_asid[12]) || tlb_g[12]);
assign match0[13] = (s0_vppn[18:9]==tlb_vppn[13][18:9])
                    && (tlb_ps4MB[13] || s0_vppn[8:0]==tlb_vppn[13][8:0])
                    && ((s0_asid==tlb_asid[13]) || tlb_g[13]);
assign match0[14] = (s0_vppn[18:9]==tlb_vppn[14][18:9])
                    && (tlb_ps4MB[14] || s0_vppn[8:0]==tlb_vppn[14][8:0])
                    && ((s0_asid==tlb_asid[14]) || tlb_g[14]);
assign match0[15] = (s0_vppn[18:9]==tlb_vppn[15][18:9])
                    && (tlb_ps4MB[15] || s0_vppn[8:0]==tlb_vppn[15][8:0])
```

```
                    && ((s0_asid==tlb_asid[15]) || tlb_g[15]);
assign match1[ 0] = (s1_vppn[18:9]==tlb_vppn[ 0][18:9])
                    && (tlb_ps4MB[ 0] || s1_vppn[8:0]==tlb_vppn[ 0][8:0])
                    && ((s1_asid==tlb_asid[ 0]) || tlb_g[ 0]);
assign match1[ 1] = (s1_vppn[18:9]==tlb_vppn[ 1][18:9])
                    && (tlb_ps4MB[ 1] || s1_vppn[8:0]==tlb_vppn[ 1][8:0])
                    && ((s1_asid==tlb_asid[ 1]) || tlb_g[ 1]);
assign match1[ 2] = (s1_vppn[18:9]==tlb_vppn[ 2][18:9])
                    && (tlb_ps4MB[ 2] || s1_vppn[8:0]==tlb_vppn[ 2][8:0])
                    && ((s1_asid==tlb_asid[ 2]) || tlb_g[ 2]);
assign match1[ 3] = (s1_vppn[18:9]==tlb_vppn[ 3][18:9])
                    && (tlb_ps4MB[ 3] || s1_vppn[8:0]==tlb_vppn[ 3][8:0])
                    && ((s1_asid==tlb_asid[ 3]) || tlb_g[ 3]);
assign match1[ 4] = (s1_vppn[18:9]==tlb_vppn[ 4][18:9])
                    && (tlb_ps4MB[ 4] || s1_vppn[8:0]==tlb_vppn[ 4][8:0])
                    && ((s1_asid==tlb_asid[ 4]) || tlb_g[ 4]);
assign match1[ 5] = (s1_vppn[18:9]==tlb_vppn[ 5][18:9])
                    && (tlb_ps4MB[ 5] || s1_vppn[8:0]==tlb_vppn[ 5][8:0])
                    && ((s1_asid==tlb_asid[ 5]) || tlb_g[ 5]);
assign match1[ 6] = (s1_vppn[18:9]==tlb_vppn[ 6][18:9])
                    && (tlb_ps4MB[ 6] || s1_vppn[8:0]==tlb_vppn[ 6][8:0])
                    && ((s1_asid==tlb_asid[ 6]) || tlb_g[ 6]);
assign match1[ 7] = (s1_vppn[18:9]==tlb_vppn[ 7][18:9])
                    && (tlb_ps4MB[ 7] || s1_vppn[8:0]==tlb_vppn[ 7][8:0])
                    && ((s1_asid==tlb_asid[ 7]) || tlb_g[ 7]);
assign match1[ 8] = (s1_vppn[18:9]==tlb_vppn[ 8][18:9])
                    && (tlb_ps4MB[ 8] || s1_vppn[8:0]==tlb_vppn[ 8][8:0])
                    && ((s1_asid==tlb_asid[ 8]) || tlb_g[ 8]);
assign match1[ 9] = (s1_vppn[18:9]==tlb_vppn[ 9][18:9])
                    && (tlb_ps4MB[ 9] || s1_vppn[8:0]==tlb_vppn[ 9][8:0])
                    && ((s1_asid==tlb_asid[ 9]) || tlb_g[ 9]);
assign match1[10] = (s1_vppn[18:9]==tlb_vppn[10][18:9])
                    && (tlb_ps4MB[10] || s1_vppn[8:0]==tlb_vppn[10][8:0])
                    && ((s1_asid==tlb_asid[10]) || tlb_g[10]);
assign match1[11] = (s1_vppn[18:9]==tlb_vppn[11][18:9])
                    && (tlb_ps4MB[11] || s1_vppn[8:0]==tlb_vppn[11][8:0])
                    && ((s1_asid==tlb_asid[11]) || tlb_g[11]);
assign match1[12] = (s1_vppn[18:9]==tlb_vppn[12][18:9])
                    && (tlb_ps4MB[12] || s1_vppn[8:0]==tlb_vppn[12][8:0])
                    && ((s1_asid==tlb_asid[12]) || tlb_g[12]);
assign match1[13] = (s1_vppn[18:9]==tlb_vppn[13][18:9])
                    && (tlb_ps4MB[13] || s1_vppn[8:0]==tlb_vppn[13][8:0])
                    && ((s1_asid==tlb_asid[13]) || tlb_g[13]);
assign match1[14] = (s1_vppn[18:9]==tlb_vppn[14][18:9])
                    && (tlb_ps4MB[14] || s1_vppn[8:0]==tlb_vppn[14][8:0])
```

```
                    && ((s1_asid==tlb_asid[14]) || tlb_g[14]);
assign match1[15] = (s1_vppn[18:9]==tlb_vppn[15][18:9])
                    && (tlb_ps4MB[15] || s1_vppn[8:0]==tlb_vppn[15][8:0])
                    && ((s1_asid==tlb_asid[15]) || tlb_g[15]);
```

只要把这个查询比较结果生成好，那么是否查找命中的 found 就是看 match 是否不等于全 0。

```
//查找结果
assign s0_found = |match0;
assign s1_found = |match1;
```

3. 为 TLB 模块添加无效化操作。

TLB 模块需要支持 INVTLB 指令的查找、无效操作。invtlb 的无效操作都是在 TLB 模块内部根据查找结果直接将符合条件的 TLB 表项的 E 位置为 0。

```
//无效化操作
integer i;
always @(posedge clk) begin
    if (we)
        tlb_e[w_index] <= w_e;
    else if (invtlb_valid) begin
        for (i = 0; i < TLBNUM; i = i + 1) begin
            if (invtlb_op == 5'h0 || invtlb_op == 5'h1) begin
                tlb_e[i] <= 1'b0;
            end
            else if (invtlb_op == 5'h2 && tlb_g[i]) begin
                tlb_e[i] <= 1'b0;
            end
            else if (invtlb_op == 5'h3 && !tlb_g[i]) begin
                tlb_e[i] <= 1'b0;
            end
            else if (invtlb_op == 5'h4 && tlb_asid[i] == s1_asid) begin
                tlb_e[i] <= 1'b0;
            end
            else if (invtlb_op == 5'h5 && !tlb_g[i] && tlb_asid[i] == s1_asid &&
tlb_vppn[i] == s1_vppn) begin
                tlb_e[i] <= 1'b0;
            end
            else if (invtlb_op == 5'h6 && (tlb_g[i] || tlb_asid[i] == s1_asid) &&
tlb_vppn[i] == s1_vppn) begin
                tlb_e[i] <= 1'b0;
            end
        end
    end
end
```

（二）Exp18 设计：

1. 将实践任务 17 完成的 TLB 模块集成到实践任务 16 完成的 CPU 中。

此处主要是将实践任务 17 中的 tlb.v 模块在 mycpu_top.v 中实例化，代码不做展示。

2. 在 CPU 中增加 TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB 指令。

为了在 CPU 中添加 MMU 功能，除了集成上一节设计的 TLB 模块，还需要实现一系列 MMU 相关的控制状态寄存器和指令。

首先是 TLBSRCH 指令，这个指令需要对 TLB 进行查找，意味着它执行时需要利用 TLB 模块中的查找逻辑。我们选择复用访存指令的 TLB 查找端口并在 EX 级发起查找请求是最合适的。

```
assign s1_vppn = inst_tlbsrch_MEM ? tlbehi_vppn : // for tlbsrch
                 inst_invtlb_MEM ? rkd_value_MEM[31:13] : //for invtlb
                 data_sram_vaddr[31:13]; // for other instructions
```

然后是 TLBWR 和 TLBFILL 指令。由于 TLB 模块设计了一组独立的写入接口，因此 TLBWR 和 TLBFILL 指令什么时候写 TLB 与何时读取该指令源操作数直接相关。我们选择 TLBWR 和 TLBFILL 指令在写回级写 TLB 是合适的，因为此时读取的 CSR 值都是正确的。

```
assign tlb_we = inst_tlbwr_WB || inst_tlbfill_WB;
assign w_index = inst_tlbwr_WB ? tlbidx_index : tlbfill_dest;
assign w_e = (inst_tlbwr_WB || inst_tlbfill_WB) && estat_ecode != 6'h3f && !tlbidx_ne
          || (inst_tlbfill_WB || inst_tlbwr_WB) && estat_ecode == 6'h3f;
```

接下来是 TLBRD 指令。考虑到 TLBR 指令对 CSR 的影响，我们认为 TLBRD 指令在写回级读 TLB 并更新相关 CSR 是最合适的。

```
assign tlbidx_ne_we = inst_tlbsrch_MEM || inst_tlbrd_WB;
assign tlbidx_ps_we = inst_tlbrd_WB;
assign tlbidx_ne_wdata = (inst_tlbsrch_MEM && !s1_found) | (inst_tlbrd_WB) & !r_e;

assign tlbidx_ps_wdata = (inst_tlbrd_WB && r_e) ? r_ps : 6'b0;

assign tlb_ex_WB = (csr_ecode_WB == 6'h3f || csr_ecode_WB == 6'h1 || csr_ecode_WB == 6'h2
                || csr_ecode_WB == 6'h3  || csr_ecode_WB == 6'h4 || csr_ecode_WB == 6'h7)
&& !inst_need_refetch_WB;
assign tlbe_we = inst_tlbrd_WB;
assign asid_asid_we = inst_tlbrd_WB;
assign asid_asid_wdata = (inst_tlbrd_WB && !r_e) ? 10'b0 : r_asid;
assign tlbehi_vppn_wdata = (inst_tlbrd_WB && r_e) ? r_vppn :
                           (tlb_ex_WB && ex_from_IF_WB) ? pc_WB[31:13] :
                           (tlb_ex_WB && !ex_from_IF_WB) ? alu_result_WB[31:13] :
                            19'b0;
assign tlbelo0_wdata = (inst_tlbrd_WB && r_e) ? {4'b0, r_ppn0, 1'b0, r_g, r_mat0, r_plv0,
r_d0, r_v0} : 32'b0;
```

```
assign tlbelo1_wdata = (inst_tlbrd_WB && r_e) ? {4'b0, r_ppn1, 1'b0, r_g, r_mat1, r_plv1,
r_d1, r_v1} : 32'b0;
```

最后是 INVTLB 指令。INVTLB 指令的源操作数都来自通用寄存器或立即数，并且它需要（部分）复用 TLB 模块中已有的查找逻辑。

```
assign invtlb_op  = inst_ID[4:0];
assign invtlb_op_not_exist = !(invtlb_op == 5'h0 || invtlb_op == 5'h1 || invtlb_op == 5'h2
|| invtlb_op == 5'h3 || invtlb_op == 5'h4 || invtlb_op == 5'h5 || invtlb_op == 5'h6) &&
inst_invtlb;
assign s1_vppn = inst_tlbsrch_MEM ? tlbehi_vppn : // for tlbsrch
                 inst_invtlb_MEM ? rkd_value_MEM[31:13] : //for invtlb
                 data_sram_vaddr[31:13]; // for other instructions
assign s1_asid = invtlb_valid ? rj_value_MEM[9:0] : // for invtlb
                                asid_asid; // for other instructions
```

3. 在 CPU 中增加 TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY CSR 寄存器。

主要在控制状态寄存器模块增加 TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY CSR 寄存器，代码如下：

```
// TLBIDX
always @(posedge clk) begin
    if (tlbidx_index_we)
        csr_tlbidx_index <= tlbidx_index_wdata;
    else if (csr_we && csr_waddr==`CSR_TLBIDX)
        csr_tlbidx_index <= csr_wmask[`CSR_TLBIDX_INDEX] & csr_wdata[`CSR_TLBIDX_INDEX]
                          | ~csr_wmask[`CSR_TLBIDX_INDEX] & csr_tlbidx_index;

    if (tlbidx_ps_we)
        csr_tlbidx_ps <= tlbidx_ps_wdata;
    else if (csr_we && csr_waddr==`CSR_TLBIDX)
        csr_tlbidx_ps <= csr_wmask[`CSR_TLBIDX_PS] & csr_wdata[`CSR_TLBIDX_PS]
                       | ~csr_wmask[`CSR_TLBIDX_PS] & csr_tlbidx_ps;

    if (tlbidx_ne_we)
        csr_tlbidx_ne <= tlbidx_ne_wdata;
    else if (csr_we && csr_waddr==`CSR_TLBIDX)
        csr_tlbidx_ne <= csr_wmask[`CSR_TLBIDX_NE] & csr_wdata[`CSR_TLBIDX_NE]
                       | ~csr_wmask[`CSR_TLBIDX_NE] & csr_tlbidx_ne;
End

// ASID
always @(posedge clk) begin
    if (asid_asid_we)
        csr_asid_asid <= asid_asid_wdata;
    else if (csr_we && csr_waddr==`CSR_ASID)
        csr_asid_asid <= csr_wmask[`CSR_ASID_ASID] & csr_wdata[`CSR_ASID_ASID]
```

```
                    | ~csr_wmask[`CSR_ASID_ASID] & csr_asid_asid;
end
assign csr_asid_asidbits = 8'd10;

// TLBEHI
always @(posedge clk) begin
    if (tlbe_we || tlb_ex_WB)
        csr_tlbehi_vppn <= tlbehi_vppn_wdata;
    else if (csr_we && csr_waddr==`CSR_TLBEHI)
        csr_tlbehi_vppn <= csr_wmask[`CSR_TLBEHI_VPPN] & csr_wdata[`CSR_TLBEHI_VPPN]
                        | ~csr_wmask[`CSR_TLBEHI_VPPN] & csr_tlbehi_vppn;
end

// TLBELO0
always @(posedge clk) begin
    if (tlbe_we) begin
        csr_tlbelo0_v <= tlbelo0_wdata[`CSR_TLBELO0_V];
        csr_tlbelo0_d <= tlbelo0_wdata[`CSR_TLBELO0_D];
        csr_tlbelo0_plv <= tlbelo0_wdata[`CSR_TLBELO0_PLV];
        csr_tlbelo0_mat <= tlbelo0_wdata[`CSR_TLBELO0_MAT];
        csr_tlbelo0_g <= tlbelo0_wdata[`CSR_TLBELO0_G];
        csr_tlbelo0_ppn <= tlbelo0_wdata[`CSR_TLBELO0_PPN];
    end
    else if (csr_we && csr_waddr==`CSR_TLBELO0) begin
        csr_tlbelo0_v <= csr_wmask[`CSR_TLBELO0_V] & csr_wdata[`CSR_TLBELO0_V]
                      | ~csr_wmask[`CSR_TLBELO0_V] & csr_tlbelo0_v;
        csr_tlbelo0_d <= csr_wmask[`CSR_TLBELO0_D] & csr_wdata[`CSR_TLBELO0_D]
                      | ~csr_wmask[`CSR_TLBELO0_D] & csr_tlbelo0_d;
        csr_tlbelo0_plv <= csr_wmask[`CSR_TLBELO0_PLV] & csr_wdata[`CSR_TLBELO0_PLV]
                        | ~csr_wmask[`CSR_TLBELO0_PLV] & csr_tlbelo0_plv;
        csr_tlbelo0_mat <= csr_wmask[`CSR_TLBELO0_MAT] & csr_wdata[`CSR_TLBELO0_MAT]
                        | ~csr_wmask[`CSR_TLBELO0_MAT] & csr_tlbelo0_mat;
        csr_tlbelo0_g <= csr_wmask[`CSR_TLBELO0_G] & csr_wdata[`CSR_TLBELO0_G]
                        | ~csr_wmask[`CSR_TLBELO0_G] & csr_tlbelo0_g;
        csr_tlbelo0_ppn <= csr_wmask[`CSR_TLBELO0_PPN] & csr_wdata[`CSR_TLBELO0_PPN]
                        | ~csr_wmask[`CSR_TLBELO0_PPN] & csr_tlbelo0_ppn;
    end
end

// TLBELO1
always @(posedge clk) begin
    if (tlbe_we) begin
        csr_tlbelo1_v <= tlbelo1_wdata[`CSR_TLBELO1_V];
        csr_tlbelo1_d <= tlbelo1_wdata[`CSR_TLBELO1_D];
        csr_tlbelo1_plv <= tlbelo1_wdata[`CSR_TLBELO1_PLV];
```

```
        csr_tlbelo1_mat <= tlbelo1_wdata[`CSR_TLBELO1_MAT];
        csr_tlbelo1_g <= tlbelo1_wdata[`CSR_TLBELO1_G];
        csr_tlbelo1_ppn <= tlbelo1_wdata[`CSR_TLBELO1_PPN];
    end
    else if (csr_we && csr_waddr==`CSR_TLBELO1) begin
        csr_tlbelo1_v <= csr_wmask[`CSR_TLBELO1_V] & csr_wdata[`CSR_TLBELO1_V]
                      | ~csr_wmask[`CSR_TLBELO1_V] & csr_tlbelo1_v;
        csr_tlbelo1_d <= csr_wmask[`CSR_TLBELO1_D] & csr_wdata[`CSR_TLBELO1_D]
                      | ~csr_wmask[`CSR_TLBELO1_D] & csr_tlbelo1_d;
        csr_tlbelo1_plv <= csr_wmask[`CSR_TLBELO1_PLV] & csr_wdata[`CSR_TLBELO1_PLV]
                        | ~csr_wmask[`CSR_TLBELO1_PLV] & csr_tlbelo1_plv;
        csr_tlbelo1_mat <= csr_wmask[`CSR_TLBELO1_MAT] & csr_wdata[`CSR_TLBELO1_MAT]
                        | ~csr_wmask[`CSR_TLBELO1_MAT] & csr_tlbelo1_mat;
        csr_tlbelo1_g <= csr_wmask[`CSR_TLBELO1_G] & csr_wdata[`CSR_TLBELO1_G]
                      | ~csr_wmask[`CSR_TLBELO1_G] & csr_tlbelo1_g;
        csr_tlbelo1_ppn <= csr_wmask[`CSR_TLBELO1_PPN] & csr_wdata[`CSR_TLBELO1_PPN]
                        | ~csr_wmask[`CSR_TLBELO1_PPN] & csr_tlbelo1_ppn;
    end
end

// TLBRENTRY
always @(posedge clk) begin
    if (csr_we && csr_waddr==`CSR_TLBRENTRY)
        csr_tlbrentry_pa <= csr_wmask[`CSR_TLBRENTRY_PA] & csr_wdata[`CSR_TLBRENTRY_PA]
                         | ~csr_wmask[`CSR_TLBRENTRY_PA] & csr_tlbrentry_pa;
end
```

（三）Exp19 设计：

1. 利用 MMU 进行虚实地址转换。

直接地址翻译、直接映射窗口地址翻译和 TLB 地址翻译逻辑均可以使用组合逻辑实现。则虚实地址转换实现就是将请求虚地址——nextPC 同时送往直接地址翻译、直接映射窗口地址翻译和 TLB 地址翻译三处，同时分别进行各自的虚实地址转换，然后再根据当前所处的地址翻译模式决定是否选择直接地址翻译逻辑的转换结果，否则再根据直接映射窗口是否有命中的信息决定是选择直接映射窗口地址翻译逻辑的转换结果还是 TLB 模块的地址转换结果。

```
assign inst_flag_dmw0_hit = csr_crmd_pg && nextpc[31:29] == csr_dmw0[31:29] &&
csr_dmw0[csr_crmd_plv] == 1'b1;
assign inst_flag_dmw1_hit = csr_crmd_pg && nextpc[31:29] == csr_dmw1[31:29] &&
csr_dmw1[csr_crmd_plv] == 1'b1 && !inst_flag_dmw0_hit;
assign inst_flag_tlb_hit  = !inst_flag_dmw0_hit && !inst_flag_dmw1_hit && s0_found &&
csr_crmd_plv <= s0_plv;

assign inst_sram_addr  =  !csr_crmd_pg ? nextpc : (
```

```
                    {32{inst_flag_dmw0_hit}} & {csr_dmw0[27:25], nextpc[28:0]} |
                    {32{inst_flag_dmw1_hit}} & {csr_dmw1[27:25], nextpc[28:0]} |
                    {32{inst_flag_tlb_hit }} & {s0_ppn[19:9], s0_ps == 6'd12 ?
s0_ppn[8:0] : nextpc[20:12], nextpc[11:0]}
                    );
wire [31:0] data_sram_vaddr = alu_result_MEM;

assign data_flag_dmw0_hit = csr_crmd_pg && data_sram_vaddr[31:29] == csr_dmw0[31:29] &&
csr_dmw0[csr_crmd_plv] == 1'b1;
assign data_flag_dmw1_hit = csr_crmd_pg && data_sram_vaddr[31:29] == csr_dmw1[31:29] &&
csr_dmw1[csr_crmd_plv] == 1'b1 && !data_flag_dmw0_hit;
assign data_flag_tlb_hit = !data_flag_dmw0_hit && !data_flag_dmw1_hit && s1_found &&
csr_crmd_plv <= s1_plv;

assign data_sram_addr  = !csr_crmd_pg ? data_sram_vaddr : (
                         {32{data_flag_dmw0_hit}} & {csr_dmw0[27:25],
data_sram_vaddr[28:0]} |
                         {32{data_flag_dmw1_hit}} & {csr_dmw1[27:25],
data_sram_vaddr[28:0]} |
                         {32{data_flag_tlb_hit }} & {s1_ppn[19:9], s1_ps == 6'd12 ?
s1_ppn[8:0] : data_sram_vaddr[20:12], data_sram_vaddr[11:0]}
                        );
```

2. 实现 MMU 相关的异常。

MMU 相关异常的实现与页映射地址翻译模式下 TLB 的查找结果相关，所需的各种结果信息在实现 TLB 模块时都已经设置了相应的输出接口。本阶段所要做的就是在取指和访存两部分利用 TLB 模块的这些输出信息，同时结合访问自身的类型实现这些异常的判断逻辑。余下的设计与第七章介绍的异常的一般性实现方式一样，将前面在取指和访存两部分生成的异常判断结果沿流水线逐级传递至写回级，然后在写回级更新相关的 CSR 寄存器、触发异常、清空流水线并跳转到异常入口处。

```
always @(posedge aclk) begin
    if (reset) begin
        pc <= 32'h1bfffffc;     //trick: to make nextpc be 0x1c000000 during reset
        ecode_MMU_IF <= 6'd0;
    end
    else if (pipe_ready_go_preIF & ~br_taken)begin
        pc <= nextpc;
        ecode_MMU_IF <= ecode_MMU_preIF;
    end
    else if (pipe_ready_go_preIF & br_taken) begin
        pc <= pc;
        ecode_MMU_IF <= ecode_MMU_IF;
    end
end
```

```
assign pc_unalign = nextpc[1:0] != 2'b00;
assign ex_IF     = ecode_MMU_IF != 6'h00;
assign csr_ecode = ecode_MMU_IF;
assign ex_from_IF = ex_IF;

assign ecode_MMU_preIF =  pc_unalign ? 6'h8 :
                          {6{csr_crmd_pg && !inst_flag_dmw0_hit && !inst_flag_dmw1_hit}} &
(
                          !s0_found ? 6'h3F          :  // TLBR
                          !s0_v     ? 6'h03          :  // PIF
                          !inst_flag_tlb_hit ? 6'h07 :  // PPI
                          6'h00                         // no MMU exception
                          );

wire [5:0] csr_ecode_MEM_m = inst_need_refetch_MEM ? 6'h00 :
                             csr_ecode_MEM != 6'd0                                    ?
csr_ecode_MEM :
                            (!(inst_ld_w_MEM || inst_ld_b_MEM || inst_ld_h_MEM ||
inst_ld_bu_MEM || inst_ld_hu_MEM
                            || inst_st_w_MEM || inst_st_b_MEM || inst_st_h_MEM)
                            || !csr_crmd_pg || data_flag_dmw0_hit || data_flag_dmw1_hit) ?
6'h00        :
                            ! s1_found                                               ?
6'h3F        :  // TLBR
                            ! s1_v && (inst_ld_w_MEM || inst_ld_b_MEM || inst_ld_h_MEM ||
inst_ld_bu_MEM || inst_ld_hu_MEM)

  ? 6'h01        :  // PIL
                            ! s1_v && (inst_st_w_MEM || inst_st_b_MEM || inst_st_h_MEM)  ?
6'h02        :  // PIH
                            ! data_flag_tlb_hit                                      ?
6'h07          :  // PPI
                            ! s1_d && (inst_st_w_MEM || inst_st_b_MEM || inst_st_h_MEM)  ?
6'h04        :  // PME
                           6'h00
                           ;
```

三、实验过程

（一）错误记录

1、错误 1： Exp18 漏加 TLBRENTRY 例外入口

（1）错误现象

当触发 TLB 重填例外时，跳转入口出现异常。

（2）分析定位过程

通过观察流水线控制信号波形，发现是由于 TLB 重填例外仍然使用例外入口地址（EENTRY）而非 TLB 重填例外入口地址（TLBRENTRY）。

（3）错误原因

TLB 重填例外需使用专用的 TLB 重填例外入口地址（TLBRENTRY）。

（4）修正效果

在添加 TLBRENTRY 后可正常通过测试节点。

2、错误 2： Exp19 加入 TLB 相关异常后，此前的诸多 CSR 寄存器未考虑 TLB 异常，导致错误

（1）错误现象

如 TLB 索引（TLBIDX）寄存器，考虑 TLB 重填异常，则有其第 31 位，若 CSR.ESTAT.Ecode!=0x3F，将该位的值取反后写入到被写 TLB 项的 E 位；若此时 CSR.ESTAT.Ecode=0x3F，那么被写入的 TLB 项的 E 位总是置为 1，与该位的值无关。而此前在 Exp18 中未进行考虑。

（2）错误原因

TLB 相关 CSR 寄存器未考虑 TLB 异常带来的影响。

（3）修正效果

为相关寄存器加入相应的处理逻辑后可正常通过测试节点。

四、实验总结

本次实验的主要内容是为 CPU 添加存储管理单元（MemoryManagement Unit，简称 MMU）。虽然总体实验难度相对不大，但本次实验内容涉及的技术 细节较多，不容易分出主次。通过本次实验的学习与讨论，小组同学在顺利完成实验的同时，也联系理论课内容，掌握了 TLBMMU 的相关知识，理解了 LoongArch 架构中 MMU 相关的控制状态寄存器和指令，并根据对 CPU 中的地址翻译机制的知识，实现了 LoongArch 架构 TLB 相关异常及其处理过程，以及在流水线 CPU 中添加 TLB 支持。

五、实验分工

exp17 代码编写： 武弋

exp18 代码编写：张家玮

exp19 代码编写与 prj6 实验报告撰写：王泽黎

除完成各自的任务外，小组成员之间在遇到问题时相互讨论，共同解决了实验中遇到的问题。