计 算 机 体 系 结 构（研 讨 课）实 验 报 告

实验项目 prj6 小组编号 28 组员姓名 刘景平、张钰堃、付博宇

一、 实验目标

（一）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。

（二）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。

（三）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 DMW CSR 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、 逻辑电路结构与仿真波形相关说明

（一）tlb 模块设计

• tlb 模块的输入输出端口和内部寄存器设置：

按照讲义 216-218 页给出的定义，设置两个查找端口（分别用于取指和访存）、一个读端口和一个写端口，另外添加专门用于执行 invtlb 指令的两个输入信号。

按照 loongarch 架构对于 tlb 页表项的定义设置寄存器堆。

```
77  );
78
79  reg [TLBNUM-1:0] tlb_e;
80  reg [TLBNUM-1:0] tlb_ps4MB;   //pagesize 1:4MB, 0:4KB
81
82  reg [18:0] tlb_vppn [TLBNUM-1:0];  // 19 bits = 32 - 12(4KB) - 1(od
83  reg [ 9:0] tlb_asid [TLBNUM-1:0];
84  reg        tlb_g    [TLBNUM-1:0];
85
86  reg [19:0] tlb_ppn0 [TLBNUM-1:0];
87  reg [ 1:0] tlb_plv0 [TLBNUM-1:0];
88  reg [ 1:0] tlb_mat0 [TLBNUM-1:0];
89  reg        tlb_d0   [TLBNUM-1:0];
90  reg        tlb_v0   [TLBNUM-1:0];
91
92  reg [19:0] tlb_ppn1 [TLBNUM-1:0];
93  reg [ 1:0] tlb_plv1 [TLBNUM-1:0];
94  reg [ 1:0] tlb_mat1 [TLBNUM-1:0];
95  reg        tlb_d1   [TLBNUM-1:0];
96  reg        tlb_v1   [TLBNUM-1:0];
97
98  // read port
99  assign r_e    = tlb_e    [r_index];
100 assign r_vppn = tlb_vppn [r_index];
101 assign r_ps   = tlb_ps4MB[r_index] ? 6'd21 : 6'd12;
102 assign r_asid = tlb_asid [r_index];
```

• invtlb 的处理：

```
// invtlb
wire [3:0] cond [TLBNUM - 1:0];
wire [TLBNUM - 1:0] invtlb_mask [31:0];
generate for(i = 0; i < TLBNUM; i = i + 1) begin
        assign cond[i][0] = ~tlb_g[i];
        assign cond[i][1] = tlb_g[i];
        assign cond[i][2] = s1_asid == tlb_asid[i];
        assign cond[i][3] = (s1_vppn[18:9] == tlb_vppn[i][18:9]) && (tlb_ps4MB[i] || s1_vppn[8:0] == tlb_vppn[i][8:0]);

end
endgenerate

assign invtlb_mask[0] = 16'hffff;
assign invtlb_mask[1] = 16'hffff;
assign invtlb_mask[2] = cond[1];
assign invtlb_mask[3] = cond[0];
assign invtlb_mask[4] = cond[0] & cond[2];
assign invtlb_mask[5] = cond[0] & cond[2] & cond[3];
assign invtlb_mask[6] = (cond[1] | cond[2]) & cond[3];

generate for (i = 7; i < 32; i = i + 1) begin
    assign invtlb_mask[i] = 16'b0;
end
endgenerate
```

按照讲义上介绍的方法，处理 invtlb 指令的时候为 32 个 tlb 页表项增加一个四位的 cond 向量，每一位分别表示该页表项是否符合四种 condition 的某一个；并为每个页表项增加一个 7 位长的 mask 向量，其中每一位对应该页表项是否符合 invtlb—op 的一种情况。在需要进行 invtlb 擦除操作时，根据 invtlb-op 进行对符合条件的页表项进行擦除。

（二）tlbsrch、tlbrd、tlbwr、invtlb 指令

• 四条指令与 tlb 的交互

在 EX 阶段完成 tlbsrch 和 invtlb 指令与 tlb 模块的交互，在 ID 阶段译码之后，在 EX 模块向 tlb 发出 tlbsrch 和 invtlb 指令的信号。

```
//port with tlb.v
output [18:0] s1_vppn,
output        s1_va_bit12,
output [9:0]  s1_asid,

input         s1_found,
input [3:0]   s1_index,
```

```
//input s1_found,
input [19:0] s1_ppn,
input [1:0] s1_plv,
input s1_d,
input s1_v,

output invtlb_valid,
output [4:0] invtlb_op
```

在 WB 阶段完成 tlbrd、tlbwr 指令与 tlb 的交互，同时完成 tlbrd、tlbwr、tlbsrch 和 csr 的交互。

```
module stage5_WB(
    input                           r_d0,
    input                           r_v0,

    input [19:0]                    r_ppn1,
    input [1:0]                     r_plv1,
    input [1:0]                     r_mat1,
    input                           r_d1,
    input                           r_v1,

    //for tlbrd
    output [19:0]                tlbrd_tlbelo0_ppn,
    output                       tlbrd_tlbelo0_g,
    output [1:0]                 tlbrd_tlbelo0_mat,
    output [1:0]                 tlbrd_tlbelo0_plv,
    output                       tlbrd_tlbelo0_d,
    output                       tlbrd_tlbelo0_v,

    output [19:0]                tlbrd_tlbelo1_ppn,
    output                       tlbrd_tlbelo1_g,
    output [1:0]                 tlbrd_tlbelo1_mat,
    output [1:0]                 tlbrd_tlbelo1_plv,
    output                       tlbrd_tlbelo1_d,
    output                       tlbrd_tlbelo1_v,

    output [5:0]                 tlbrd_tlbidx_ps,
    output [9:0]                 tlbrd_asid_asid,
```

（三）虚实地址转换

分别在 IF 阶段（具体来说是 pre-IF 阶段）和 EX 阶段实现取指、访存的物理地址转换。两处地址转换逻辑基本一致，先判断是直接地址翻译还是映射地址翻译，在映射地址翻译模式时先判断直接映射窗口 DMW0 和 DMW1 是否命中，若未命中则查找 TLB 进行翻译。若仍无有效 TLB 表项命中，则会引发 TLB 重填异常。

```
//choose next_pc
wire if_dt;
assign if_dt = crmd_da & ~crmd_pg;    //da=1, pg=0

wire if_indt;
assign if_indt = ~crmd_da & crmd_pg;    //da=0, pg=1

wire if_dmw0;
assign if_dmw0 = ((plv == 0 && DMW0_PLV0) || (plv == 3 && DMW0_PLV3)) &&
                 (crmd_datf == DMW0_MAT) && (next_pc[31:29] == DMW0_VSEG);

wire if_dmw1;
assign if_dmw1 = ((plv == 0 && DMW1_PLV0) || (plv == 3 && DMW1_PLV3)) &&
                 (crmd_datf == DMW1_MAT) && (next_pc[31:29] == DMW1_VSEG);

wire if_ppt;
assign if_ppt = if_indt && ~(if_dmw0 | if_dmw1);
```

其中 DMW0 和 DMW1 两个 CSR 寄存器结构在参考手册上有详细说明，也没有特殊写入逻辑，以组合逻辑分别连接到 IF 阶段、EX 阶段中进行相关模式判断以及物理地址翻译。

位	名字	读写	描述
0	PLV0	RW	为 1 表示在特权等级 PLV0 下可以使用该窗口的配置进行直接映射地址翻译。
2:1	0	R0	保留域。读返回 0，且软件不允许改变其值。
3	PLV3	RW	为 1 表示在特权等级 PLV3 下可以使用该窗口的配置进行直接映射地址翻译。
5:4	MAT	RW	虚地址落在该映射窗口下访存操作的存储访问类型。
24:6	0	R0	保留域。读返回 0，且软件不允许改变其值。
27:25	PSEG	RW	直接映射窗口的物理地址的[31:29]位。
28	0	R0	保留域。读返回 0，且软件不允许改变其值。
31:29	VSEG	RW	直接映射窗口的虚地址的[31:29]位。

```
begin
    if(reset)
        begin
            DMW0_PLV0 <= 0;
            DMW0_ZERO1 <= 0;
            DMW0_PLV3 <= 0;
            DMW0_MAT <= 0;
            DMW0_ZERO2 <= 0;
            DMW0_PSEG <= 0;
            DMW0_ZERO3 <= 0;
            DMW0_VSEG <= 0;
        end
    else if(csr_we && csr_num == `CSR_DMW0)
        begin
            DMW0_PLV0 <= csr_wmask[`DMW_PLV0] & csr_wvalue[`DMW_PLV0]
                | ~csr_wmask[`DMW_PLV0] & DMW0_PLV0;

            DMW0_PLV3 <= csr_wmask[`DMW_PLV3] & csr_wvalue[`DMW_PLV3]
                | ~csr_wmask[`DMW_PLV3] & DMW0_PLV3;

            DMW0_MAT <= csr_wmask[`DMW_MAT] & csr_wvalue[`DMW_MAT]
                | ~csr_wmask[`DMW_MAT] & DMW0_MAT;

            DMW0_PSEG <= csr_wmask[`DMW_PSEG] & csr_wvalue[`DMW_PSEG]
                | ~csr_wmask[`DMW_PSEG] & DMW0_PSEG;
```

确认地址翻译模式后，利用组合逻辑直接选择对应模式翻译得到的物理地址即可。

```
wire [31:0] next_pc_dt;   //dt --> directly translate
assign next_pc_dt = next_pc;

wire [31:0] next_pc_dmw0; //DMW0
assign next_pc_dmw0 = {DMW0_PSEG , next_pc[28:0]};

wire [31:0] next_pc_dmw1; //DMW1
assign next_pc_dmw1 = {DMW1_PSEG , next_pc[28:0]};

wire [31:0] next_pc_ptt; //ppt --> page table translate
assign next_pc_ptt = {s0_ppn, next_pc[11:0]};
```

```
assign next_pc_p = if_dt ? next_pc_dt : if_indt ?
                (if_dmw0 ? next_pc_dmw0 : if_dmw1 ? next_pc_dmw1 : next_pc_ptt) : 0;
```

访存物理地址翻译与取指类似，区别主要在于从不同端口读取页表相关数据。这一部分按照书上描述编写即可。

注意翻译得到的物理地址只用于传给内存查找对应指令或数据，cpu 内部计算、不同流水级传递的仍然为虚拟地址。

（四）MMU 相关异常实现

新添加六种 MMU 相关异常，按照指令手册上叙述进行判定，然后把相应信号连接到例外相应相关信号，如 csr_ecode 等。

MMU 相关异常的判断主要发生在 IF 级和 EX 级。IF 级需要判断 TLB 重填、页特权等级不合规、取指操作页无效的异常，EX 级除了取指页无效外其他例外都需要判断。

```
wire fs_ex_fetch_tlb_refill;
wire fs_ex_inst_invalid;
wire fs_ex_fetch_plv_invalid;

assign fs_ex_fetch_tlb_refill = if_ppt & ~s0_found;
assign fs_ex_inst_invalid = if_ppt & s0_found & ~s0_v;
assign fs_ex_fetch_plv_invalid = if_ppt & s0_found & s0_v & (plv > s0_plv)
```

图 1 IF 级例外判断

```
assign es_ex_loadstore_tlb_fill = if_ppt & (es_res_from_mem | es_mem_we) & ~s1_found;
assign es_ex_load_invalid = if_ppt & es_res_from_mem & s1_found & ~s1_v;
assign es_ex_store_invalid = if_ppt & es_mem_we & s1_found & ~s1_v;
assign es_ex_loadstore_plv_invalid = if_ppt & (es_res_from_mem | es_mem_we) & s1_found
                                & s1_v & (plv > s1_plv);
assign es_ex_store_dirty = if_ppt & es_mem_we & s1_found & s1_v & ~s1_d & ((plv < s1_plv) | (plv == s1_plv));
```

图 2 EX 级例外判断

此外，MMU 相关例外在相应时还需要额外保存信息到 BADV 和 TLBEHI 中，需要在 csr 中添加相关写入逻辑

```
always @(posedge clk)
    begin
        if(wb_ex && wb_ex_addr_err)
            csr_badv_vaddr <= ((wb_ecode == `ECODE_ADE && wb_esubcode == `ESUBCODE_ADEF) ||
                               (wb_ecode == `ECODE_PIF) ||
                               (wb_ecode == `ECODE_PPI ) && if_fetch_plv_ex || //
                               (wb_ecode == `ECODE_TLBR ) && if_fetch_tlb_refill)//
                                ? wb_pc : wb_vaddr;
    end
```

另外需要注意，TLB 重填异常的入口是通过 TLBRENTRY 单独配置的，因而当发生 TLB 重填异常时，读取的 csr_num 需要设置为 TLBRENTRY 对应序号

```
assign csr_num = ex_tlb_refill ? `CSR_TLBRENTRY : (wb_ex ? `CSR_ERA : ws_csr_num) ;
assign csr_re = 1'b1;
```

三、 实验过程中遇到的问题以及 debug 心得

1. tlb 模块中 cond 和 mask 向量的处理：

```
// invtlb
wire [TLBNUM - 1:0] cond [3:0];
wire [TLBNUM - 1:0] invtlb_mask [6:0];
generate for(i = 0; i < TLBNUM; i = i + 1) begin
        assign cond[0][i] = ~tlb_g[i];
        assign cond[1][i] = tlb_g[i];
        assign cond[2][i] = s1_asid == tlb_asid[i];
        assign cond[3][i] = (s1_vppn[18:9] == tlb_vppn[i][18:9]) &&

end
endgenerate

assign invtlb_mask[0] = 16'hffff;
assign invtlb_mask[1] = 16'hffff;
assign invtlb_mask[2] = cond[1];
assign invtlb_mask[3] = cond[0];
assign invtlb_mask[4] = cond[0] & cond[2];
assign invtlb_mask[5] = cond[0] & cond[2] & cond[3];
assign invtlb_mask[6] = (cond[1] | cond[2]) & cond[3];
```

需要注意两个二维向量的命名方式：如果搞反了 mask 或 cond 向量前后的两个中括号的内容，可能会导致 mask 向量在赋值的时候出错，变成全 X。

2. tlb 异常处理中需要将引发异常的虚地址保存到 CSR.BADV 中，将虚地址的[31:12]位写入 CSR.TLBEHI 的 VPPN 域。需要注意这里所说的是引发异常的虚地址，而 TLB 重填、页特权等级不合规例外无法直接从例外类型判断到底是取指还是访存地址引发异常，故需要额外引入相关接口，方便这里进行判断。

```
//tlb IF exception
output                          if_fetch_plv_ex,
output                          if_fetch_tlb_refill,
```

```
assign if_fetch_plv_ex = ws_ex_fetch_plv_invalid;
assign if_fetch_tlb_refill = ws_ex_fetch_tlb_refill;
```

四、 实验分工

张钰堃负责完成 exp17 和 exp18，刘景平负责完成 exp19。