中国科学院大学
《计算机体系结构基础(研讨课)》实验报告

姓名 艾华春,李霄宇,王敬华

学号 2022K8009916011,2022K8009929029,2022K8009925009

实验项目编号 6 实验名称 存储管理单元设计

一、 逻辑电路结构与仿真波形的截图及说明

- **TLB 模块设计。**

1. TLB 总体工作原理

   TLB 采用全相联查找表的组织形式。在 TLB 模块中保存了 TLB 页表的内容,共 16 个页表项。并通过 2 个查找端口,1 个读端口,1 个写端口和一个 INVTLB 指令端口与外部交换数据。实现了对 TLB 页表的 4 种操作:

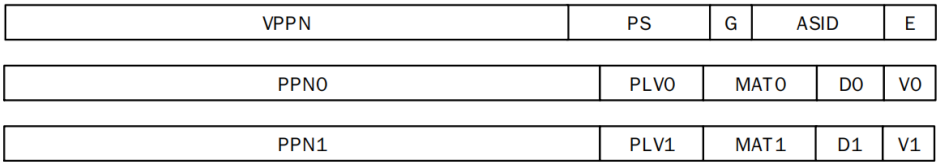
   查找:输入虚双页号 vppn、虚地址 12 位 va_bit12 与地址空间标识 asid,得到对应物理页的内容。

   写:输入页表项数 index,并将输入的页表项内容写到对应的页表项里。

   读:输入页表项数 index,并得到对应的页表项中的内容。

   无效指令 INVTLB:使满足 opcode 对应条件的页表项无效化。

   TLB 页表中每一个页表项为一个双页,包含比较部分和物理转换部分,结构如下图:



   本次实验中实现的 TLB 页表中包含 16 个页表项,能保存 16 对 32 个页。页大小有 2MB 与 4KB 两种,通过 ps4MB 信号指示。使用 15 组寄存器来保存页表内容:

```
reg [TLBNUM-1:0] tlb_e;
reg [TLBNUM-1:0] tlb_ps4MB; //pagesize 1:4MB, 0:4KB

reg [18:0] tlb_vppn [TLBNUM-1:0];
reg [ 9:0] tlb_asid [TLBNUM-1:0];
reg        tlb_g    [TLBNUM-1:0];

reg [19:0] tlb_ppn0 [TLBNUM-1:0];
reg [ 1:0] tlb_plv0 [TLBNUM-1:0];
reg [ 1:0] tlb_mat0 [TLBNUM-1:0];
reg        tlb_d0   [TLBNUM-1:0];
reg        tlb_v0   [TLBNUM-1:0];

reg [19:0] tlb_ppn1 [TLBNUM-1:0];
reg [ 1:0] tlb_plv1 [TLBNUM-1:0];
reg [ 1:0] tlb_mat1 [TLBNUM-1:0];
```

```
    reg       tlb_d1  [TLBNUM-1:0];
    reg       tlb_v1  [TLBNUM-1:0];
```

2. TLB 读写设计

TLB 模块的读写方式与寄存器堆的读写方式十分类似。读 TLB 的方式是根据传入的 index 将对应表项各个域的数据通过组合逻辑读出；写 TLB 的方式是根据传入的 index 和写使能信号，在下一拍将要写的各个域的数据写入对应表项相应位置。

```
    //read
assign r_e  = tlb_e  [r_index];
assign r_vppn = tlb_vppn [r_index];
assign r_ps = tlb_ps4MB[r_index]? 6'b010101 : 6'b001100;
assign r_asid = tlb_asid [r_index];
assign r_g  = tlb_g  [r_index];

assign r_ppn0 = tlb_ppn0 [r_index];
assign r_ppn1 = tlb_ppn1 [r_index];
assign r_plv0 = tlb_plv0 [r_index];
assign r_plv1 = tlb_plv1 [r_index];
assign r_mat0 = tlb_mat0 [r_index];
assign r_mat1 = tlb_mat1 [r_index];
assign r_d0 = tlb_d0  [r_index];
assign r_d1 = tlb_d1  [r_index];
assign r_v0 = tlb_v0  [r_index];
assign r_v1 = tlb_v1  [r_index];

//write
wire  w_ps4MB;
assign w_ps4MB=(w_ps == 6'b010101)?1:0;

always @(posedge clk)
    begin
        if(we)
            begin
                tlb_e[w_index]   <= w_e;
                tlb_vppn[w_index] <= w_vppn;
                tlb_ps4MB[w_index] <= w_ps4MB;
                tlb_asid[w_index] <= w_asid;
                tlb_g[w_index]   <= w_g;

                tlb_ppn0[w_index] <= w_ppn0;
                tlb_ppn1[w_index] <= w_ppn1;
                tlb_plv0[w_index] <= w_plv0;
                tlb_plv1[w_index] <= w_plv1;
                tlb_mat0[w_index] <= w_mat0;
                tlb_mat1[w_index] <= w_mat1;
                tlb_d0[w_index]  <= w_d0;
                tlb_d1[w_index]  <= w_d1;
                tlb_v0[w_index]  <= w_v0;
```

```
            tlb_v1[w_index]   <= w_v1;
        end
```

但在读写过程中需要注意的是，由于龙芯架构 32 位精简版只支持 4KB 和 4MB 两种页大小，因此尽管 TLB 表项中的 PS 域是 6bit，但其只有 0 和 1 两种取值：0 表示页大小为 4KB（两个物理页大小均为 4KB），1 表示页大小为 4MB（两个物理页大小均为 2MB）。而 TLB 模块输入和输出中的 PS 端口标记了页的真实大小，取值分别是 6'd21 和 6'd12，因此在对页表项进行读写操作时需要将两者进行转换。

除此之外，TLB 写操作还需要实现 TLB invtlb 指令的 opcode 对应的操作：

各 op 对应的操作如下表所示，未在表中出现的 op 将触发保留指令例外。

op	操作
0x0	清除所有页表项。
0x1	清除所有页表项。此时操作效果与 op=0 完全一致。
0x2	清除所有 G=1 的页表项。
0x3	清除所有 G=0 的页表项。
0x4	清除所有 G=0，且 ASID 等于寄存器指定 ASID 的页表项。
0x5	清除 G=0，且 ASID 等于寄存器指定 ASID，且 VA 等于寄存器指定 VA 的页表项。
0x6	清除所有 G=1 或 ASID 等于寄存器指定 ASID，且 VA 等于寄存器指定 VA 的页表项。

按照讲义上面的提示，可以将各操作的匹配分解成若干“子匹配”的逻辑组合，具体来说，可得到 4 个“子匹配”的判断条件：(1)cond1——G 域是否等于 0；(2)cond2——G 域是否等于 1；(3)cond3——s1_asid 是否等于 ASID 域；(4)cond4——s1_vppn 是否匹配 VPPN 和 PS 域

```
    generate
    for (i = 0; i < TLBNUM; i = i + 1) begin
        assign cond1[i] = ~tlb_g[i];
        assign cond2[i] = tlb_g[i];
        assign cond3[i] = s1_asid == tlb_asid[i];
        assign cond4[i] = (s1_vppn[18:9] == tlb_vppn[i][18:9])&&(tlb_ps4MB[i]||(s1_vppn[8:0] ==
            tlb_vppn[i][8:0]));
    end
endgenerate
```

那么 invtlb op=0、1 的匹配条件就可以表达为 cond1||cond2，op=4 的匹配条件可以表达为 cond1&&cond3，op=5 的匹配条件可以表达为 cond1&&cond3&&cond4，op=6 的匹配条件可以表达为 (cond2||cond3)&&cond4。根据 cond1 cond4 得到 match 信号如下：

```
    assign inv_match = (invtlb_op == 5'h0 || invtlb_op == 5'h1) ? (cond1 | cond2):
    invtlb_op == 5'h2 ? cond2:
    invtlb_op == 5'h3 ? cond1:
    invtlb_op == 5'h4 ? cond1 & cond3:
    invtlb_op == 5'h5 ? cond1 & cond3 & cond4:
    (cond2 | cond3) & cond4;
```

在 invtlb 指令有效时，根据输入的 opcode 选择 inv_match，将 inv_match 对应位为 1 的页表项无效化（e 值清零），为 0 的页表项不变：

```
    else if(invtlb_valid && invtlb_op < 5'h7)
    begin
```

```
        tlb_e            <= ~inv_match & tlb_e;
    end
```

3. TLB 查找设计

TLB 模块提供了两个通道用于查找，一个通道对应取指阶段的查找，另一个通道对应访存阶段的查找。查找的流程是并行化的，即同时对 16 个页表项的奇偶相邻页表的物理转换信息进行比较和查找，并用两个 16 位宽的 match 信号表示查找结果。

```
    genvar i;
generate
    for (i = 0; i < TLBNUM; i = i + 1) begin
        assign match0[i] = (s0_vppn[18:9]==tlb_vppn[i][18:9]) && (tlb_ps4MB[i] ||
            s0_vppn[8:0]==tlb_vppn[i][8:0]) && ((s0_asid==tlb_asid[i]) || tlb_g[i]);
        assign match1[i] = (s1_vppn[18:9]==tlb_vppn[i][18:9]) && (tlb_ps4MB[i] ||
            s1_vppn[8:0]==tlb_vppn[i][8:0]) && ((s1_asid==tlb_asid[i]) || tlb_g[i]);
    end
endgenerate
```

匹配的逻辑是：对于每一个页表项，一个是要查看比较部分的全局标志位 G 和地址空间表示位 ASID；除此之外，还需要查看比较部分的页大小 PS 位和虚双页号 VPPN 位，如果 PS 位为 1，表示页大小为 4MB（两个物理页大小均为 2MB），此时只需要比较 VPPN 的高 10 位是否与 s_vppn 的高 10 位相等，若相等则查找成功；如果 PS 位为 0，表示页大小为 4KB（两个物理页大小均为 4KB），此时需要比较 VPPN 和 s_vppn 的全部位数。如果以上全部满足则查找成功。

然后得到 found，portindex，port，ps 信号（端口 0 为例），其中含义如下：

found：是否匹配到。

index：匹配到的页表项的项数。

port：双页中的哪一页，页大小为 2MB 时通过 vppn 的第 8 位来判断，4KB 时通过输入的 va_bit12 来判断。

ps：页大小，2MB 为 21，4KB 为 12。

```
assign s0_index = match0[0] ? 4'b0000 :...
assign s1_index = match1[0] ? 4'b0000 :...

assign s0_found = |match0[TLBNUM-1:0];
assign s1_found = |match1[TLBNUM-1:0];

assign s0_ps = tlb_ps4MB[s0_index] ? 6'b010101 : 6'b001100;// 21 or 12
assign s1_ps = tlb_ps4MB[s1_index] ? 6'b010101 : 6'b001100;

assign s0_port = tlb_ps4MB[s0_index] ? s0_vppn[8] : s0_va_bit12;
assign s1_port = tlb_ps4MB[s1_index] ? s1_vppn[8] : s1_va_bit12;
```

最后通过 index 和 port 从页表中得到物理转换信息并输出：

```
    assign s0_ppn = s0_port ? tlb_ppn1[s0_index] : tlb_ppn0[s0_index];
    assign s1_ppn = s1_port ? tlb_ppn1[s1_index] : tlb_ppn0[s1_index];
    assign s0_plv = s0_port ? tlb_plv1[s0_index] : tlb_plv0[s0_index];
```

```
assign s1_plv = s1_port ? tlb_plv1[s1_index] : tlb_plv0[s1_index];
assign s0_mat = s0_port ? tlb_mat1[s0_index] : tlb_mat0[s0_index];
assign s1_mat = s1_port ? tlb_mat1[s1_index] : tlb_mat0[s1_index];
assign s0_d = s0_port ? tlb_d1[s0_index] : tlb_d0[s0_index];
assign s1_d = s1_port ? tlb_d1[s1_index] : tlb_d0[s1_index];
assign s0_v = s0_port ? tlb_v1[s0_index] : tlb_v0[s0_index];
assign s1_v = s1_port ? tlb_v1[s1_index] : tlb_v0[s1_index];
```

- **添加 TLB 相关指令和 CSR 寄存器。**

1. 添加 TLB 相关指令和 CSR 寄存器
2.

- **添加 TLB 相关例外支持，添加虚实地址映射功能。**

1. 为 cpu 增加虚实映射功能

   在 if 和 ex 级发出访存请求的虚拟地址，同时送往直接地址翻译，直接窗口映射和 tlb 地址映射翻译三处，同时进行各自的虚实地址转换，然后再根据 CSR 寄存器中的值进行选择。

   以 if 级发出指令访存信号为例，通过组合逻辑，在同一个 clk 内生成三种地址翻译的结果，最后通过多路选择器进行选择。

   访存物理地址 pre_pc_pa 首先根据控制寄存器中的 crmd.pg 中的值选择当前为直接地址地址翻译模式（物理地址 = 虚拟地址 = pre_pc），还是映射地址翻译模式 (物理地址 = pre_pc_map)。

   映射地址翻译（pre_pc_map）首先判断是否命中直接映射窗口，若不是，则使用 tlb 地址映射结构。

   第一行将虚拟地址 pre_pc [31:12] 发送到 tlb 进行 tlb 地址翻译，tlb 中的组合逻辑查找结果 s0_ps, s0_ppn, s0_found

```
assign {s0_vppn, s0_va_bit12} = pre_pc[31:12];// output to tlb

assign hit_dmw0 =       csr_dmw0_plv_met & csr_dmw0_vseg == pre_pc[31:29];
assign hit_dmw1 =       csr_dmw1_plv_met & csr_dmw1_vseg == pre_pc[31:29];

assign pre_pc_pa =      csr_crmd_pg ? pre_pc_map // enable mapping
:pre_pc;                // direct translate

assign pre_pc_map = hit_dmw0 ? {csr_dmw0_pseg, pre_pc[28:0]} // dierct map windows 0
                  :hit_dmw1? {csr_dmw1_pseg, pre_pc[28:0]}  // direct map windows 1
                  :(s0_ps == 6'b010101) ? {s0_ppn[19:9], pre_pc[20:0]} // tlb map: ps 4Mb
                  :{s0_ppn,pre_pc[11:0]};                     // tlb map : ps 4kb
```

2. 为 cpu 添加 tlb 相关例外支持

   在每个流水级中设置 excep_en ，excep_ecode, excep_esubdcode 分别表示当前流水级是否触发异常，和异常的种类。

   以在 ex 流水级中为例：

```
assign ex_excep_en =    ex_excep_ALE | ex_excep_TLBR|ex_excep_PIL| ex_excep_PIS|
    ex_excep_PPI|ex_excep_PME| id_excep_en;

assign ex_excep_TLBR =   //TLB 重填例外
   (ex_res_from_mem | ex_mem_we) &csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1 & ~s1_found;
assign ex_excep_PIL =   // load 操作页无效例外
   (ex_res_from_mem) &csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1 & s1_found & ~s1_v;
assign ex_excep_PIS =   // store 操作页无效例外
   (ex_mem_we) &csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1 & s1_found & ~s1_v;
assign ex_excep_PPI =   // 页特权等级不合规例外
   (ex_res_from_mem | ex_mem_we) & csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1 & s1_found & s1_v &
       (s1_plv < csr_crmd_plv);
assign ex_excep_PME =    //页修改例外
   (ex_res_from_mem | ex_mem_we) & csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1 & s1_found & s1_v &
       (s1_plv >= csr_crmd_plv) & ~s1_d;

assign ex_badv =        (id_excep_en) ? id_badv
                        : ex_alu_result;
assign ex_esubcode =    (id_excep_en) ? id_esubcode
                        :9'b0;
assign ex_ecode =       (id_excep_en) ? id_ecode
                        :ex_excep_ALE ? 6'h9
                        :ex_excep_TLBR ?6'h3f // tlb refill
                        :ex_excep_PIL ?6'h1
                        :ex_excep_PIS ?6'h2
                        :ex_excep_PPI ?6'h7
                        :6'h4; // pme
```

所有 tlb 相关的异常，触发的必要条件都有：当前指令是访存指令（load/store），当前虚拟地址翻译模式不是直接地址翻译模式，且不命中直接映射窗口。

即 `(ex_res_from_mem | ex_mem_we) &csr_crmd_pg & ~hit_dmw0 & ~hit_dmw1`

(a) TLB 重填例外

在 tlb 没有找到对应的页表项，即`s1_found == 0`

(b) load 操作页无效例外, store 操作页无效例外

在 tlb 中找到对应的页表项，但是页表项的有效位为 0，即`s1_found & ~s1_v`

(c) 页特权等级不合

在 tlb 中找到对应的页表项，页表项的有效，但当前特权等级为 plv3，但是页表项所需要特权等级为 plv0, 即 `s1_found & s1_v & (s1_plv < csr_crmd_plv)`

(d) 页修改例外在 tlb 中找到对应的页表项，页表项的有效，且特权等级满足要求，但是页表项脏位为 0, 即 `s1_found & s1_v & (s1_plv >= csr_crmd_plv) & ~s1_d`

由上面的分析可知，与 tlb 相关的异常触发条件互斥，不用考虑不同异常间的优先级的问题。

二、 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)

- **if 级错误丢弃异常处理函数的第一条指令。**

要求在发生 mmu 相关异常的时候,不能向总线发出请求,所以在 pre_if 级中识别到 pc 的虚拟地址发生异常,则将 inst_sram_req 拉低。

但是导致了异常传到 wb 级时,发出更新流水线信号 flush 时,将 if 级误判为还在等待指令返回(实际上 pre_if 级没有发出请求),则将 inst_cancel 信号(表示要丢弃接收到的第一条指令)拉高,导致丢弃了异常处理含数的第一条指令。

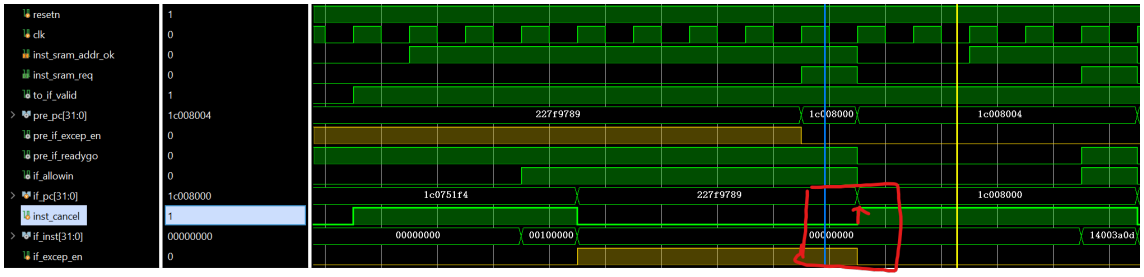


图 1: 错误拉高 inst_cancel

修改:在 inst_cancel 拉高的逻辑中,判断 if 级是否有异常,若有,则说明 if 级没有已经发出请求,但是没有接受到的总线事务,不用拉高 inst_cancel。

```
/* 清空流水线时，第一个指令需要丢弃*/
always @(posedge clk) begin
    if(~resetn)
        inst_cancel <= 1'b0;
    else if ( (if_valid & ~if_ir_valid & ~inst_sram_data_ok & ~if_excep_en // if正在等待指令返回，
        加入if级是否有异常的判断
              |pre_if_reqed_reg & ~inst_sram_data_ok) // pre_if
                  级发出请求，但是数据没有返回，也还没有进入if级
           & (flush | br_taken))
        inst_cancel <= 1'b1;
    else if(inst_sram_data_ok) // 异常后第一个需要被舍弃的指令返回
        inst_cancel <= 1'b0;
end
```

三、 小组成员分工合作情况

王敬华负责 ex 级访存添加类 sram 总线支持。

李霄宇负责部分 debug 工作。

艾华春负责添加 TLB 相关寄存器和 TLB 相关的例外支持,为 cpu 增加虚实映射功能。

实验报告为根据每人负责代码的部分,写相应部分的报告。