# Lab 8 报告

**学号: 2021K8009910001、2021K8009908024、2021K8009908004**

**姓名: 徐畅杰、张修梁、王致力**

**箱号: 34**

## 一、实验任务

在完成单独的 TLB 模块设计的基础上，将其集成到已有的 CPU 中，添加相关指令和 CSR、异常处理。

## 二、实验设计

### （一）总体设计思路

#### 1、TLB 模块

TLB 的总体设计思路类似于寄存器堆，按照相关约定进行写入与读出即可，详见后续 TLB 模块。

#### 2、MMU 模块

为了实现虚实地址转换与相关异常处理，需要增加 MMU 模块与 TLB 模块，同时也需要对 CSR 寄存器堆进行相关的改造。

首先是虚实地址的转换过程，首先判断当前处于直接地址翻译模式还是映射地址翻译模式，若是前者则直接使用虚拟地址作为物理地址，若为后者则需要先查看是否落在了直接映射窗口上，并且有符合的权限，若是则按住奥直接映射窗口的规则进行翻译，否则查找页表，并且按照相关的情况进行例外的产生：1. 没找到对应的页表，则产生 TLB 重填例外。2. 找到但是无效，则按照执行操作的类型，报出对应的操作页无效例外。3. 特权不足。4. 需要执行写操作但是没有写检查功能，报出页修改例外。若均没有例外则返回对应的物理地址用于进行相关的处理。而 TLB 相关的异常只会发生在 IF 级或者 EXE 级。而相关的异常处理也需要将映射状态改为直接地址翻译状态，并且将。

其次 MMU 就是按照这一基本认识设计的，其不仅仅需要负责流水线与 TLB 相关的地址翻译，而且为了设计的简洁，将 CSR 相关的 TLBRD，TLBSRCH，TLBWR，TLBFILL，INVTLB 等连线也经过此处。

#### 3、TLB 相关冲突的解决

同时部分 TLB 相关的指令需要考虑相关的冲突，这里的 TLB 指令不仅仅是 TLBSRCH，TLBRD，TLBWR，TLBFILL，INVTLB，也有改写相关寄存器的指令，冲突不仅仅会导致相同类型的指令被阻塞，而且可能处于其下一条指令也会因为相关寄存器的改写导致无效，故而可能需要刷掉整个流水线。这些指令直到译码级才知道该指令是否为 TLB 相关的指令。现在来看具体的 TLB 指令：

1.TLBSRCH：TLBSRCH 需要与访存指令共用一套 TLB 查找端口，所以需要安排在执行级进行，但是其依赖的寄存器 ASID，TLBEHI 可能被 TLBRD，CSRWR，CSRXCHG 改写，但是实际上执行这些操

作都应该清空流水线，因为若这些寄存器被改写，则取值时该指令就可能无效，因为取值级的 TLB 查找实际上也依赖于这些寄存器，那么直接刷掉流水线即可。而其查找的 TLB 表项可能被 TLBWR，TLBFILL，INVTLB 改写，但是后续的分析可知这些指令会刷掉流水线，所以不需要考虑。

2.TLBRD：TLBRD 依赖于 TLBIDX，其会更新 ASID，TLBEHI，TLBELO0，TLBELO1，其中前者会被 CSRWR，CSRXCHG，TLBSRCH 更新，所以将其放在写回级较为合适，同时由于其会改写 ASID，所以之后的指令需要重取。

3.TLBWR 与 TLBFILL：二者都依赖于 ASID，TLBEHI，TLBELO0，TLBELO1，TLBIDX，这些寄存器会被 TLBRD，CSRWR，CSRXCHG 改写，所以放在写回级较为合适，同时由于其会更改 TLB 值，所以也需要重取。

4.INVTLB：由于其需要复用 TLBSRCH 的端口，所以应该在 EX 级进行，同时其依赖的除了源操作数的 CSR 寄存器都会因为可能导致取值错误，所以都会直接清空流水线，所以不需要考虑出现问题。INVTLB 会更新 TLB，所以其之后的指令也需要重取。

综上 TLBSRCH 与 INVTLB 都在执行级进行，而 TLBRD，TLBWR，TLBFILL 在写回级进行。同时 TLBSRCH 不需要刷新流水线，而 TLBWR，TLBFILL，TLBRD，INVTLB 以及部分的 CSRWR，CSRXCHG 都需要刷新流水线。

### 4、TLB 相关异常的报出

取值级的需要复用 INE 的通路进行，同时执行级报出的可以复用 ALE 的。但是需要在改写 TLBEHUI 的 vppn 域上按照相关的约定进行设计，参考手册即可，在此不再赘述。

## （二）重要模块 1 设计：TLB 模块

本模块的设计在讲义中有较多的介绍，本节主要介绍如何将 TLB 表项无效化。

### 1、工作原理

在判断是否需要无效化时，参考讲义上给出的 4 个条件，使用一个 4*16 的数组来记录每一个 TLB 表项对应的条件。

最后定义一个 32 项的数组 `invtlb_masks`，来存储 `INVTLB` 所对应的操作，下标为 `INVTLB` 指令对应的操作码，由于 `cond[x]` 是 16 位的，故 `invtlb_masks` 可以直接作用于 TLB 所有表项的有效位。

### 2、接口定义/具体代码实现

```
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[18:10] == tlb_vppn[i][18:10] &&
                        (s1_vppn[ 9: 0] == tlb_vppn[i][ 9: 0] || tlb_ps4MB[i]);
end
endgenerate

assign invtlb_masks[0] = cond[0] || cond[1];
assign invtlb_masks[1] = cond[0] || cond[1];
assign invtlb_masks[2] = cond[1];
assign invtlb_masks[3] = cond[0];
assign invtlb_masks[4] = cond[0] && cond[2];
assign invtlb_masks[5] = cond[0] && cond[2] && cond[3];
assign invtlb_masks[6] = (cond[0] || cond[2]) && cond[3];
generate for (i = 7; i < 32; i = i + 1) begin
    assign invtlb_masks[i] = 16'b0;
    // TIPS: 未在表中出现的 op 将触发保留指令例外，此处不做处理
end
endgenerate
```

### 3、功能描述

内部具体是怎么设计的，描述要简洁明了，直中要害。

## （三）重要模块 2 设计：MMU 模块

基本上仅仅用于连接，具体的实现通路分布在处理器于 CSR 寄存器中，且之前已经进行过叙述。

### 1、工作原理

将取值级与执行级需要的虚拟地址翻译为物理地址，同时将 CSR 与 TLB 连接，从而实现 TLB 相关指令。

### 2、接口定义

接口实在太多了... 在此部分使用线束进行简述，线束即为为了实现相关操作所需要的所有线。

表 1: 接口定义表

| 接口名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|
| clk | IN | 1 | TLB 时钟 |
| pre_if_vaddr | IN | 32 | 预取指虚地址 |
| pre_if_addr | OUT | 32 | 预取指实地址 |
| s0_exc | OUT | 3 | 预取指异常 |
| exe_vaddr | IN | 32 | 执行虚地址 |
| exe_addr | OUT | 32 | 执行实地址 |
| s1_exc | OUT | 5 | 执行异常 |
| asid | IN | 10 | 当前 asid |
| plv | IN | 2 | 当前特权级 |
| da | IN | 1 | 实地址模式 |
| pg | IN | 1 | 虚地址翻译模式 |
| dmw0 | IN | 32 | 0 号直接映射窗口相关信息 |
| dmw1 | IN | 32 | 1 号直接映射窗口相关信息 |
| tlb_srch | IN | ... | TLBSRCH 相关输入 |
| tlb_r_out | OUT | ... | TLBSRCH 相关输出 |
| tlb_wr | IN | ... | TLBWR 相关输入 |
| tlb_r | IN | 4 | TLBRD 的 index 值 |
| tlb_r_out | OUT | ... | TLBRD 相关输出 |
| tlb_inv | IN | ... | INVTLB 相关输入 |

### 3、代码分析

仅仅从单个端口分析地址翻译工作：

```
assign s0_dmw0_hit = (pre_if_vaddr[31:29] == dmw0_vseg) & ((plv == 2'b0) & dmw0_plv0 | (plv == 2'b1
assign s0_dmw1_hit = (pre_if_vaddr[31:29] == dmw1_vseg) & ((plv == 2'b0) & dmw1_plv0 | (plv == 2'b1
assign s0_dmw_ppn = {{3{s0_dmw0_hit}} & dmw0_pseg | {3{s0_dmw1_hit}} & dmw1_pseg ,pre_if_vaddr[28:1
assign s0_dmw_hit = s0_dmw0_hit | s0_dmw1_hit;
```

```
//tlb translate of s0(pre if)
assign s0_vppn = pre_if_vaddr[31:13];
assign s0_va_bit12 = pre_if_vaddr[12];
//final s0
assign s0_translated_ppn = s0_dmw_hit ? s0_dmw_ppn : s0_ppn;
assign pre_if_addr = {{20{da}} & pre_if_vaddr[31:12] | {20{pg}} & s0_translated_ppn,pre_if_vaddr[11
//handle s0 exception
assign s0_tlbr = ~s0_found;
assign s0_pif  = ~s0_v;
assign s0_ppi  = (s0_plv == 2'b0) & (plv == 2'b11);
assign s0_exc = {3{pg & ~s0_dmw_hit}} & {s0_pif,s0_ppi,s0_tlbr};//only in pg mode can generate prob
```

首先查看是否在直接映射窗口命中，命中的条件为 vseg 相等而且特权级符合。之后查看页表是否命中，并且按照相关约定给出例外。使用直接映射窗口得到的地址与页表翻译地址得到虚地址翻译模式下得到的物理地址。最后查看页表翻译模式为物理地址直接映射还是虚地址翻译得到最终的物理地址。同时页根据当前地址翻译模式与直接映射窗口是否命中决定是否需要报出异常。

### 4、功能描述

将 CPU 与 TLB 模块进行转接，同时封装翻译操作。

# 三、实验过程

## （一）实验流水账

1. Nov. 28 22:00 - Nov. 29 00:30 完成 exp17 并上板测试
2. Nov. 31 11:00 - Nov. 29 17:00 完成 exp18 并上板测试
3. Dec. 6 15:00 - Dec. 7 13:00 完成 exp19 并上板测试

## （二）错误记录

### 1、错误 1：match 与 index 转换错误

#### （1）错误现象

在测试单独的 TLB 模块时，读和查找出现问题。

#### （2）分析定位过程

根据波形查找，match 没有问题，但 index 并不是独热码 match 中 1 所在的位置，而只有 0 和 1 两个值。

#### （3）错误原因

在转换模块（本质是一个对数运算模块）中，所写的选择器缺少位宽 4。

```
    module log(
    input  wire [15:0] in, output wire [ 3:0] out
);
    assign out = in[ 0] &  4'd0 | ... | in[15] & 4'd15;
endmodule
```

#### （4）修正效果

先位扩展再进行与运算即可

```
    module log(
    input  wire [15:0] in, output wire [ 3:0] out
);
    assign out = {4{in[ 0]}} &  4'd0 | ... | {4{in[15]}} & 4'd15;
endmodule
```

## 2、错误 2：BADV 错误

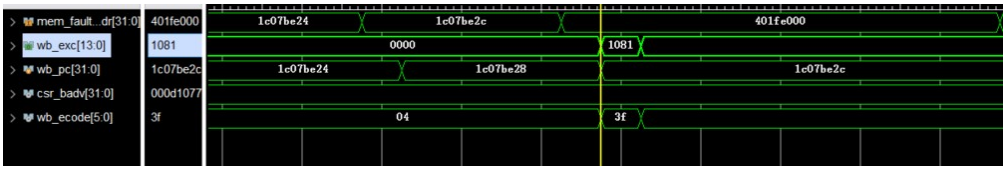
### （1）错误现象

```
[7195617 ns] Error!!!
    reference: PC = 0x1c00f044, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x401fe000
    mycpu    : PC = 0x1c00f044, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000d1077
```

BADV 寄存器的值在发生页重填例外时未能正常设置。

### （2）分析定位过程

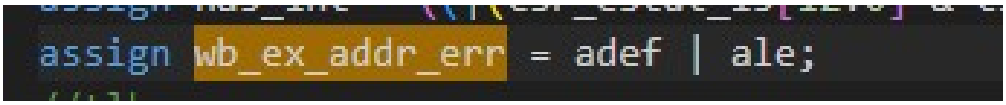
查看对应的错误指令，发现为 CSRRD 指令，推测为之前例外时未能正确设置对用的寄存器，于是找到上一条例外：




按照报出的 ecode 得知发生了页充填例外，基本确定未能正确按照对应指令手册进行相关例外发生时 CSR 寄存器的设置。

### （3）错误原因

忘记修改 badv 对应的页表例外处理。




### （4）修正效果

按照对应的约定改写：

```
assign wb_ex_addr_err = pil | pis | pif | pme | if_ppi | exe_ppi | adef | ale | if_tlbr | exe_tlbr;

always @(posedge clk ) begin
    if(wb_ex & wb_ex_addr_err)
        csr_badv_pc <= (adef | pif | if_ppi | if_tlbr)? wb_pc: wb_fault_vaddr;//now support virtual memory
end
```

### （5）归纳总结

未能熟悉相关的指令集手册，使用了较为旧的约定。

### 3、错误 3：查询页表时未查看 TLB_e

#### （1）错误现象

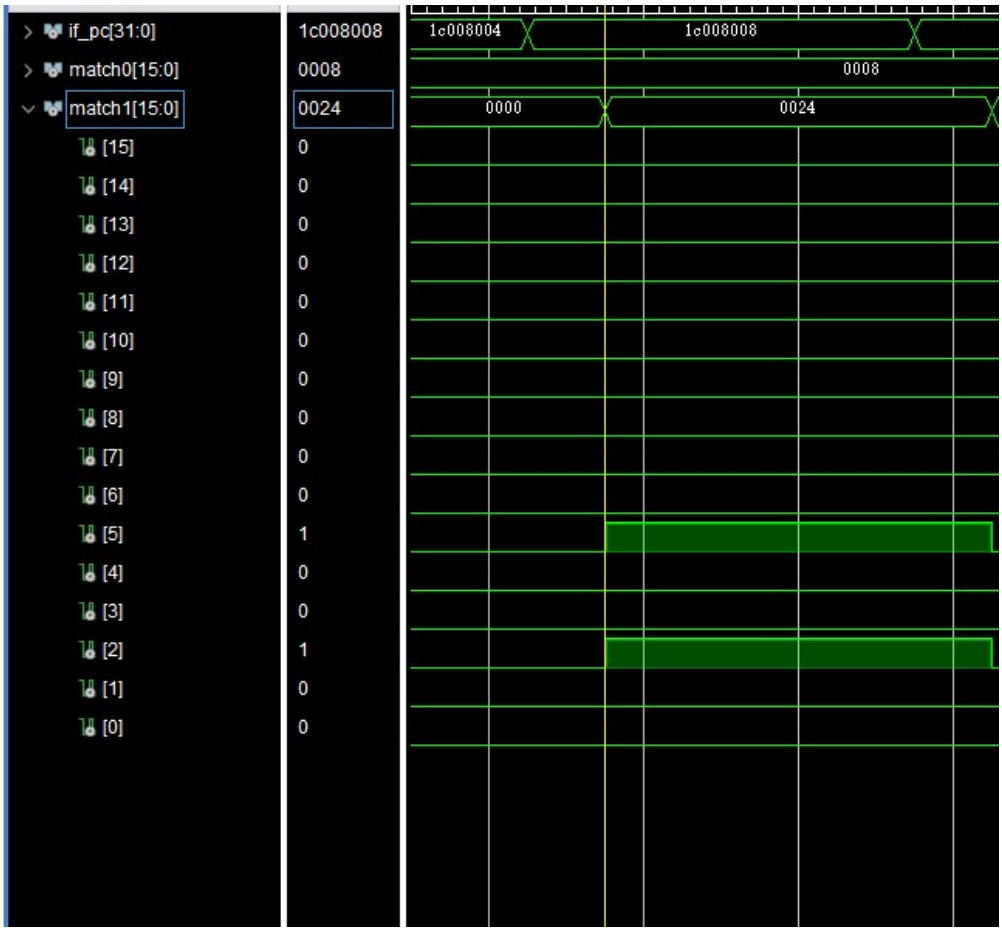
在虚地址翻译模式下 ld 指令出现问题：

```
[7245967 ns] Error!!!
    reference: PC = 0x1c008004, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x0000000f
    mycpu    : PC = 0x1c008004, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x1400000b
```

#### （2）分析定位过程

过程较为曲折，先是查看了 MMU 模块的翻译，发现并没有问题，查看的内容较多，就先不贴图了。

之后查看 TLB 表项，发现 match 居然有多项命中，但是由于优先级问题选中错误的那个：




#### （3）错误原因

后经过思考发现是讲义中的 match 方法有问题，不仅仅需要页号相等并且 asid 对应或页表为 global，而且也需要对应的页表项是有效的。

```
genvar i;
generate
    for (i = 0; i < TLBNUM; i = i + 1) begin: search
        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
```

### （4）修正效果

将每一个结果 and 上对应的 tlbe 即可。

```
genvar i;
generate
    for (i = 0; i < TLBNUM; i = i + 1) begin: search
        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])
                           && tlb_e[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])
                           && tlb_e[i];
    end
endgenerate
```

### （5）归纳总结

不能盲信讲义，需要结合实际各个 mask 的意义与指令集进行设计。

# 四、实验总结（可选）