实验 7-2 报告

第 28 小组
付琳晴

一、实验任务（10%）

增加 Index、EntryHi、EntryLo0、EntryLo1、PageMask 5 个 cp0 寄存器。增加一个 32 位的 TLB 模块，实现 TLBR、TLBWI、TLBP 三条指令。将存储器寻址映射方式改为 TLB 映射。

二、实验设计（30%）

（一）TLB 接口

在 TLB 中需要存储的每一项 TLB 结构如图 1.

EntryHi		**PageMask**		**EntryLo0**		**EntryLo1**	
VPN2	ASID	PageMask	G	PFN0	C、D、V	PFN1	C、D、V
19bit	8bit	12bit	1bit	20bit	5bit	20bit	5bit

图 1. TLB 表项结构

根据表项结构，定义 8 个寄存器数组来分别存各个域的内容，定义如图 2.

```
25   reg [18:0] VPN2 [31:0];
26   reg [ 7:0] ASID [31:0];
27   reg [11:0] MASK [31:0];
28
29   reg        G    [31:0];
30   //phy0
31   reg [19:0] PFN0 [31:0];
32   reg [ 4:0] CDV0 [31:0];
33   //phy1
34   reg [19:0] PFN1 [31:0];
35   reg [ 4:0] CDV1 [31:0];
```

图 2. 代码中数组定义

TLB 中目前需要支持四种功能：虚实地址转换、TLBWI、TLBR、TLBP。在我的设计中，虚实地址转换采用组合逻辑，三条指令功能采用时序逻辑。为实现四种功能，模块的接口定义如下图。

```
module mytlb(
    input         clk,

    input  [ 4:0] index,
    input  [31:0] Entryhi,
    input  [31:0] Pagemask,
    input  [31:0] Entrylo0,
    input  [31:0] Entrylo1,
    input  [31:0] vaddr_inst,
    input  [31:0] vaddr_data,
    input         TLBWI,
    input         TLBR,
    input         TLBP,

    output reg        index_find,
    output reg [ 4:0] index_out,
    output reg [31:0] Entryhi_out,
    output reg [11:0] mask,
    output reg [25:0] Entrylo0_out,
    output reg [25:0] Entrylo1_out,
    output     [31:0] paddr_inst,
    output     [31:0] paddr_data
    );
```

图 3. TLB 模块接口

（二）TLB 中虚实地址转换

虚实地址转换，发生在取指、存数、取数时，即 inst_sram_addr 和 data_sram_addr 的地址若在 mapped 区域内（本实验中即地址小于 0x80000000）则需要在 TLB 中查找进行虚实地址转换。为了防止对数据和指令地址同时进行查找产生冲突，因此设置了两套查找线路，分别查找。查找时，以虚地址高 19 位同时与 VPN2 的每一项比较，以 cp0 寄存器 Entryhi 第 8 位（PID 域）域 ASID 的每一项比较，若比对正确则命中，每一项的查找结果赋值 hit。这里要注意，若 G 对应项为 1，则不用比对 ASID。对于一项的查找代码如图 4.

```
/*look up for paddr_inst*/
assign hit_inst[0] = (vaddr_inst[31:13] == VPN2[0] && (G[0] || (Entryhi[7:0] == ASID[0])))?1:0;
```

图 4. TLB 查找

由于每一项 TLB 表项将映射到两个连续的物理地址，因此还需要根据虚地址的第 12 位决定选取奇地址或偶地址。每一项选出物理地址高 20 位的代码如图 5.

```
assign PFN_inst[0] = (vaddr_inst[12])?PFN1[0]:PFN0[0];
```

图 5. 选出奇地址或偶地址

最后根据每一项的查找结果结合物理地址高 20 位结合虚拟地址低 12 位的偏移量得到物理地址。物理地址的多路选择如图 6.

```
assign paddr_inst[31:12] =  ({20{hit_inst[0]}}& PFN_inst[0])   | ({20{hit_inst[1]}}& PFN_inst[1]) |
                            ({20{hit_inst[2]}}& PFN_inst[2])   | ({20{hit_inst[3]}}& PFN_inst[3]) |
                            ({20{hit_inst[4]}}& PFN_inst[4])   | ({20{hit_inst[5]}}& PFN_inst[5]) |
                            ({20{hit_inst[6]}}& PFN_inst[6])   | ({20{hit_inst[7]}}& PFN_inst[7]) |
                            ({20{hit_inst[8]}}& PFN_inst[8])   | ({20{hit_inst[9]}}& PFN_inst[9]) |
                            ({20{hit_inst[10]}} & PFN_inst[10]) | ({20{hit_inst[11]}} & PFN_inst[11]) |
                            ({20{hit_inst[12]}} & PFN_inst[12]) | ({20{hit_inst[13]}} & PFN_inst[13]) |
                            ({20{hit_inst[14]}} & PFN_inst[14]) | ({20{hit_inst[15]}} & PFN_inst[15]) |
                            ({20{hit_inst[16]}} & PFN_inst[16]) | ({20{hit_inst[17]}} & PFN_inst[17]) |
                            ({20{hit_inst[18]}} & PFN_inst[18]) | ({20{hit_inst[19]}} & PFN_inst[19]) |
                            ({20{hit_inst[20]}} & PFN_inst[20]) | ({20{hit_inst[21]}} & PFN_inst[21]) |
                            ({20{hit_inst[22]}} & PFN_inst[22]) | ({20{hit_inst[23]}} & PFN_inst[23]) |
                            ({20{hit_inst[24]}} & PFN_inst[24]) | ({20{hit_inst[25]}} & PFN_inst[25]) |
                            ({20{hit_inst[26]}} & PFN_inst[26]) | ({20{hit_inst[27]}} & PFN_inst[27]) |
                            ({20{hit_inst[28]}} & PFN_inst[28]) | ({20{hit_inst[29]}} & PFN_inst[29]) |
                            ({20{hit_inst[30]}} & PFN_inst[30]) | ({20{hit_inst[31]}} & PFN_inst[31]) ;

 assign paddr_inst[11: 0] = vaddr_inst[11:0];
```

图 6. 物理地址多路选择

（三）TLBWI 指令设计

TLBWI 指令将五个 cp0 寄存器的值输入 TLB，将 EntryHi、PageMask、EntryLo0、EntryLo1 的内容写入 TLB 表项。在我的设计中，TLBWI 指令在译码级被识别出，作为控制信号输入 TLB 中，TLB 根据该信号和 Index 寄存器值在对应项中存数。这样在 exe 级就已经成功存入了。赋值代码如图 7.

```
always @(posedge clk) begin
    if (TLBWI) begin
        VPN2[index] <= Entryhi[31:13];
        ASID[index] <= Entryhi[7:0];
        MASK[index] <= Pagemask[24:13];
        G[index]    <= Entrylo0[0] & Entrylo1[0];
        PFN0[index] <= Entrylo0[25:6];
        CDV0[index] <= Entrylo0[5:1];
        PFN1[index] <= Entrylo1[25:6];
        CDV1[index] <= Entrylo1[5:1];
    end
```

图 7. TLBWI 相关代码

（三）TLBR 指令设计

根据 Index 值，读出存在 TLB 中的各个域信息并存入对应的 cp0 寄存器中，整个过程需要两个周期。在我的设计中，decode 级将译码得到的 TLBR 信号和五个 cp0 寄存器的值输入 TLB，TLBR 信号同时要沿流水线传下去。一个周期后查询读出寻址到的 TLB 项并输出，利用沿流水线传下来的 TLBR 信号作为控制信号在 exe 级写四个 cp0 寄存器，这与之前设计的对 cp0 寄存器的赋值流水级一致。再过一个周期，才能成功写入。TLBR 相关代码如图 8.

```
else if (TLBR) begin
    Entryhi_out  <= {VPN2[index], 5'd0, ASID[index]};
    mask     <= MASK[index];
    Entrylo0_out <= {PFN0[index], CDV0[index], G[index]};
    Entrylo1_out <= {PFN1[index], CDV1[index], G[index]};

end
```

图 8. TLBR 相关代码

这样的设计将产生一个问题，即数据相关。若 TLBR 后紧跟一条 mtc0 指令，欲取出 EntryHi 寄存器中的指令，由于在 mtc0 指令在 decode 级就应该得到 cp0 寄存器的值，但此时很明显 TLBR 指令还没将值真正存入 EntryHi 寄存器，会导致出错。解决方法是，以 TLB 模块输出的值作为 EntryHi 寄存器中的值给 mtc0 直接使用。

（四）TLBP 指令设计

涉及到两个寄存器 EntryHi 和 Index，根据 EntryHi 查 TLB 并将找到的 Index 输出。查找方式类似于前面虚实地址转换的方法。若未查找到，则将 Index 最高位置 1. 赋值代码如图 9.

```
else if (TLBP) begin
    if (tlbp_hit[31:0] != 32'd0) begin
        index_out <=    ({5{tlbp_hit[0]}}& 5'd0 )  | ({5{tlbp_hit[1]}}& 5'd1 )|
                        ({5{tlbp_hit[2]}}& 5'd2 )  | ({5{tlbp_hit[3]}}& 5'd3 )|
                        ({5{tlbp_hit[4]}}& 5'd4 )  | ({5{tlbp_hit[5]}}& 5'd5 )|
                        ({5{tlbp_hit[6]}}& 5'd6 )  | ({5{tlbp_hit[7]}}& 5'd7 )|
                        ({5{tlbp_hit[8]}}& 5'd8 )  | ({5{tlbp_hit[9]}}& 5'd9 )|
                        ({5{tlbp_hit[10]}} & 5'd10) | ({5{tlbp_hit[11]}} & 5'd11) |
                        ({5{tlbp_hit[12]}} & 5'd12) | ({5{tlbp_hit[13]}} & 5'd13) |
                        ({5{tlbp_hit[14]}} & 5'd14) | ({5{tlbp_hit[15]}} & 5'd15) |
                        ({5{tlbp_hit[16]}} & 5'd16) | ({5{tlbp_hit[17]}} & 5'd17) |
                        ({5{tlbp_hit[18]}} & 5'd18) | ({5{tlbp_hit[19]}} & 5'd19) |
                        ({5{tlbp_hit[20]}} & 5'd20) | ({5{tlbp_hit[21]}} & 5'd21) |
                        ({5{tlbp_hit[22]}} & 5'd22) | ({5{tlbp_hit[23]}} & 5'd23) |
                        ({5{tlbp_hit[24]}} & 5'd24) | ({5{tlbp_hit[25]}} & 5'd25) |
                        ({5{tlbp_hit[26]}} & 5'd26) | ({5{tlbp_hit[27]}} & 5'd27) |
                        ({5{tlbp_hit[28]}} & 5'd28) | ({5{tlbp_hit[29]}} & 5'd29) |
                        ({5{tlbp_hit[30]}} & 5'd30) | ({5{tlbp_hit[31]}} & 5'd31) ;
        index_find <=   1'd0;
    end
    else begin
        index_out  <= 5'd0;
        index_find <= 1'd1;
    end
end
```

图 9. TLBP 相关代码

TLBP 也会产生数据相关，即若有一条 mtc0 在 TLBP 前面，且是向 EntryHi 存数，则 EntryHi 里还没存入数就被拿来进行 TLB 查找了，就会出现问题。解决方法是数据前递，即将还在流水线中的 mtc0 指令欲存入的值直接作

为 TLB 模块输入信号 EntryHi。

三、实验过程（60%）

（一）实验流水账

付琳晴：

12.16 下午 2 点：开始看任务书并写代码。

12.17 凌晨 2 点：debug 完成，睡觉。

12.17 上午：上板成功。

12.19 晚上：写实验报告。

（二）错误记录

1、错误 1

（1）错误现象

程序运行进了例外处理程序。

（2）分析定位过程

查看波形发现居然进了例外处理程序，查看中断、例外判断信号，发现是触发了保留指令例外，意识到新添加了三条指令之后没有更改保留指令的判断条件。

（3）错误原因

新添加指令却忘记修改保留指令判断条件，导致触发保留指令例外。

（4）修正效果

不再触发例外。

2、错误 2

（1）错误现象

测试用例连第一个都没有过。

（2）分析定位过程

会报 inst_error，出错的原因只有可能是相邻 mtc0 和 mfc0 处，存入一个数，但取出来不是存入的值。由于我规划线路时已经考虑了数据前递，因此不大可能是还没存入就取出了值。继续查看数据前递处的代码，发现应该是前递的值有问题。由于 cp0 寄存器中有几位是不可写的，使用 mtc0 的时候欲存入的值其实有几位是写不进 cp0 寄存器的，而我把这个欲存入的值直接未经处理地前递到了 mfc0 处。修改过后前递的数据应使用位拼接，应将不

可写的位置 0。

（3）错误原因

数据前递时忽略了 cp0 寄存器只有部分位可写，直接将欲写入的值进行了前递，导致出错。

（4）修正效果

先 mtc0 后 mfc0 的值正确。

（5）归纳总结

一定要记住 cp0 寄存器的特殊性，部分位可读可写，还有些位不可写，处理数据前递时可能忘记这个条件。

3、错误 3

（1）错误现象

测试用例第 6 个和第 7 个出错。

（2）分析定位过程

报 inst_error，查错的方式是对着波形看什么时候跳到 inst_error 的，由于这里出错极大可能是在 TLBWI 和 TLBR 前后，因此直接找到译码级以上两个信号有效的时候，在该波形附近查看。发现 TLBWI 根本就没存对位置，一定是传入的 Index 错误。Index 怎么可能错呢，只有可能是发生了数据相关。一看汇编代码果然，代码如图 10. Index 的值还没来得及存进去，就被 TLBWI 在 decode 级拿来做寻址了。因此传入给 TLBWI 的 Index 值不应该用 cp0_Index 寄存器的真实值，而是应该用稍后将会存入 cp0_Index 的还在流水线中的数据。这样数据前递后，值就正确了。

```
847   bfc009fc:   40820000    mtc0    v0,$0
848   bfc00a00:   42000002    tlbwi
```

图 10. 数据相关

（3）错误原因

TLBR 还没来得及把值写到 PageMask 里，mfc0 就读了，导致读出的数据出错。

（4）修正效果

使用 mfc0 取出的 cp0 的值正确。

4、错误 4

（1）错误现象

测试用例第 6 个依然出错。

（2）分析定位过程

对着波形继续看，发现在一次 TLBWI 后，使用 TLBR 从 TLB 中取出存过的 PageMask 的值会出错。又查看了 test.s 中的代码发现，TLBR 和 mfc0 指令是相连的，如图 11. 即意味着，按照我之前的设计，必定会出现 TLBR 还没来得及把值写到 PageMask 里，mfc0 就读了，因此读出的值是错误的。

```
854   bfc00a18:   42000001    tlbr
855   bfc00a1c:   40072800    mfc0    a3,$5
```

图 11. 会发生数据相关的代码

为了解决此处数据相关的问题，来分析一下这两条连续指令的流水线的关系。在 mfc0 出于 decode 级时，TLBR 处于 exe 级，此时 mfc0 需要读取 cp0 寄存器里的值作为要写入通用寄存器的值。但此时 TLB 模块仅仅刚输出读到的 PageMask 的 mask 域的值，还没来得及写入 cp0 寄存器。那么我们可以利用 TLB 的输出数据，直接作为 cp0 中的数据给 mfc0 指令使用。

（3）错误原因

TLBR 还没来得及把值写到 PageMask 里，mfc0 就读了，导致读出的数据出错。

（4）修正效果

使用 mfc0 取出的 cp0 的值正确。

（5）归纳总结

在数据相关上栽了多个坑之后，幡然醒悟，添加的这三条 TLB 指令，每一条都有使用 cp0 寄存器，就都可能与前后指令产生 RAW 数据相关。于是总结了一下，如图 12.并按照这些情况进行了相应的数据前递解决问题。

指令	读写情况	产生数据相关的情况
TLBWI	读Index	mtc0 v0, $0
		tlbwi
TLBR	读Index	mtc0 v0, $0
	写EntryHi、EntryLo0、EntryLo1、Pagemask	tlbr
		mfc0
TLBP	读EntryHi	mtc0 v0, $10
	写Index	tlbp
		mfc0 v0, $0

图 12. 会产生数据相关的情况

四、实验总结

（一）组员：付琳晴

现在看来除了 lab6 其他实验好像都挺简单的…毕竟是添个模块稍稍改一下线路的问题，这次代码写了不到一天就写完了。想大谈一下对这学期体系结构实验的感受，但是想了想还是留在第三阶段做完之后吧哈哈哈。感觉自己到了一月份之后可能无心写代码了，这次可能是最后一个认真写的 lab 了。在课程评估里认真的夸了汪老师和邢助教！老师上课讲的内容对代码框架非常有帮助，助教任务书写的也特别详尽，是所有实验课里最负责的了。