实验 7 报告 2

第 32 小组
袁峥

一、实验任务（10%）

1、CPU 增加 TLBR、 TLBWI、 TLBP 指令。

2、CPU 增加 Index、 EntryHi、 EntryLo0、 EntryLo1 、 PageMask CP0 寄存器。

3、CPU 增加 32 项 TLB 结构， 支持的页大小位 4KB。

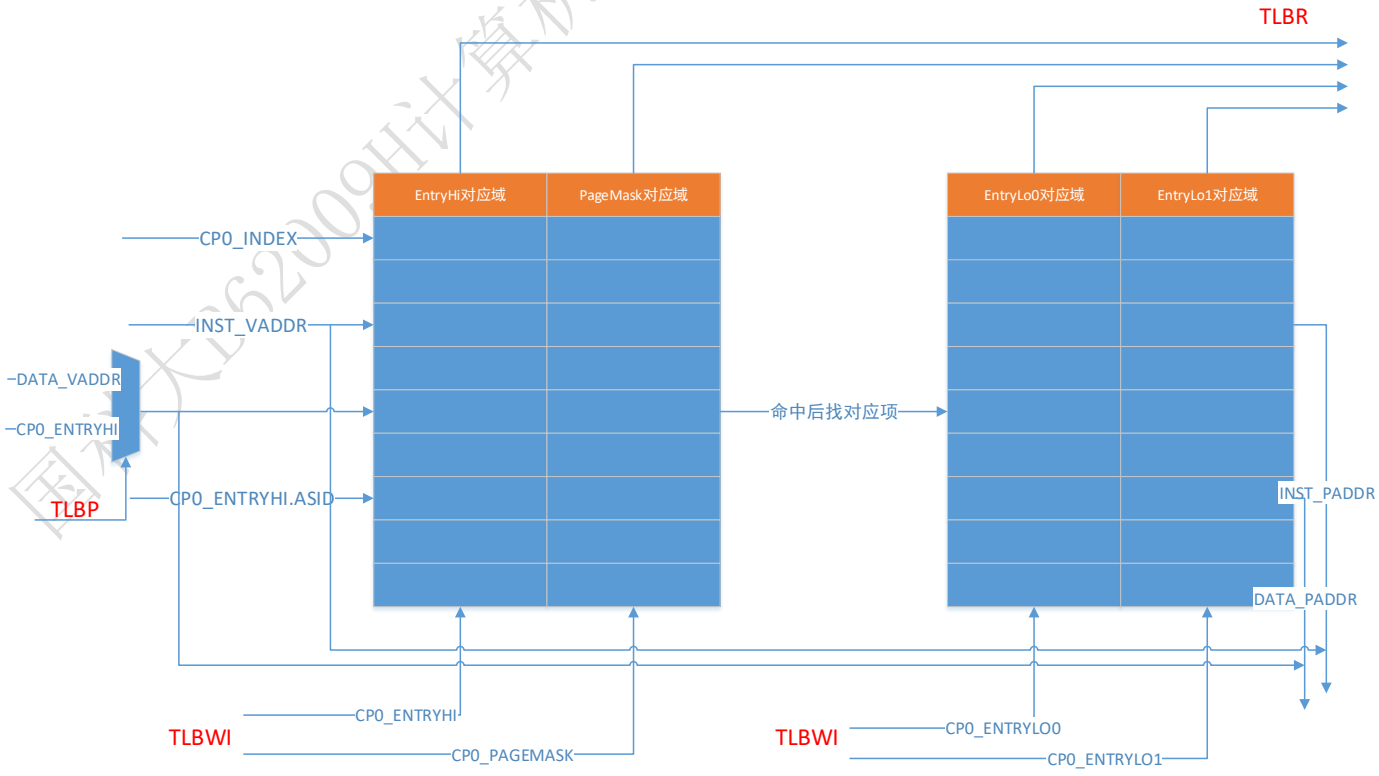
4、CPU 增加 TLB 相关例外： Refill、 Invalid、 Modified。

5、运行专用功能测试 tlb_func， 要求全部通过， 共 10 项测试。

二、实验设计（30%）

（一）对于支持 TLB 的整体设计

本次实验需要在 CPU 的设计中增加一个 32 项的 TLB，同时支持相关的指令和 CP0 寄存器。TLB 模块的设计图如下：



对于原来的 CPU 设计，首先需要增加一个 TLB 模块来进行取指和访存地址的虚实转换，并对 TLB 表项进行读写。

同时在 mycpu_top 中需要增加 5 个 CP0 寄存器，分别为 CP0_Index、 CP0_EntryHi、 CP0_EntryLo0、CP0_EntryLo1 和 CP0_PageMask。其中 CP0_Index 在 TLBP 指令时需要自动更新，CP0_EntryHi、CP0_EntryLo0、 CP0_EntryLo1 和 CP0_PageMask 在 TLBR 指令时需要自动更新。所有的 CP0 寄存器的修改统一放在执行级。

在中断和例外处理上，延续原来的想法，所有异常在执行级统一处理。inst_refill 和 inst_invalid 在取指级触发，data_refill、data_invalid 和 data_modified 在执行级触发。同时区分不同的异常处理入口地址，对于 TLB_refill 异常处理入口地址为 0xbfc00200，其余异常处理入口地址仍为 0xbfc00380。

在流水级的阻塞和清空上，如果在取指级或执行级为 TLBWI、TBLR 或者为修改 CP0_ENTRYHI 的 MTC0 指令，则此时取指级经过虚实转换后取来的指令需要重做，因为上述指令会修改 TLB 和虚实转换时需要使用到的寄存器，可能会导致后续的取指转换的地址不正确。

（二）TLB 模块

（1）基本概述

该模块主要用来存储 TLB 表的内容，同时支持取指虚地址和访存虚地址同时转换为对应物理地址。此外，需要支持 TLBWI、TLBR 和 TLBP 三条指令。

对于 TLBWI 指令，在时钟上升沿根据 Index_in 的索引将对应的 TLB 表项更新为输入值。

对于 TLBP 指令，首先在输入时对于真正要查找 TLB 的 data_vaddr 通过 TLBP 信号进行二选一，如果是 TLBP 指令，则查找 EntryHi_in 对应的虚地址，否则查到 data_vaddr_in 对应的虚地址。对于查找结果，如果找到了对应项，需要将查找信号进行编码，如果没有找到，则把 Index_out 的最高位置 1。

对于 TLBR 指令，根据 Index_in 将 TLB 表中的对应项输出。

对于虚实地址转换，首先将虚地址的高 19 位与 TLB 表中每项的 EntryHi 域的高 19 位结合 PageMask 域进行比较，同时如果 EntryHi 域的第 12 位为 0，还需将 CP0_EntryHi 寄存器的 ASID 域与 TLB 中每项的 ASID 域进行比较，如果全部满足则说明命中。如果 32 项没有一项命中，则 found 信号置 0。对于命中的情况，根据虚地址的第 12 位分别选取 EntryLo0 域或 EntryLo1 域的物理页框号，结合虚地址的低 12 位组成物理地址，同时将 V_flag 和 D_flag 置为对应 EntryLo 域中的 V 位和 D 位。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
TLBWI	IN	1	TLBWI 指令信号

名称	方向	位宽	功能描述
TLBR	IN	1	TLBR 指令信号
TLBP	IN	1	TLBP 指令信号
inst_vaddr	IN	32	取指虚地址
data_vaddr_in	IN	32	访存虚地址
EntryHi_in	IN	32	CP0_EntryHi 寄存器的输入信号
PageMask_in	IN	32	CP0_PageMask 寄存器的输入信号
EntryLo0_in	IN	32	CP0_EntryLo0 寄存器的输入信号
EntryLo1_in	IN	32	CP0_EntryLo1 寄存器的输入信号
Index_in	IN	32	CP0_Index 寄存器的输入信号
EntryHi_out	OUT	32	需要修改的 CP0_EntryHi 寄存器的值
PageMask_out	OUT	32	需要修改的 CP0_PageMask 寄存器的值
EntryLo0_out	OUT	32	需要修改的 CP0_EntryLo0 寄存器的值
EntryLo1_out	OUT	32	需要修改的 CP0_EntryLo1 寄存器的值
Index_out	OUT	32	需要修改的 CP0_Index 寄存器的值
inst_V_flag	OUT	1	取指地址虚页号在 TLB 查找到对应物理页是否有效
data_V_flag	OUT	1	访存地址虚页号在 TLB 查找到对应物理页是否有效
data_D_flag	OUT	1	访存地址虚页号在 TLB 查找到对应物理页是否可写
inst_paddr	OUT	32	取指虚地址经过 TLB 映射后的物理地址
data_paddr	OUT	32	访存虚地址经过 TLB 映射后的物理地址
inst_found	OUT	1	取指地址虚页号是否在 TLB 查找到对应物理页
data_found	OUT	1	访存地址虚页号是否在 TLB 查找到对应物理页

三、实验过程（60%）

（一）实验流水账

12 月 14 日晚上 8 点至 12 点，阅读文档 Lab07-2_3_CPU+TLB+MMU 支持及 MIPS 手册，并编写代码。

12 月 15 日下午 3 点至 10 点，调试程序并上板通过测试。

12 月 15 日下午 11 点至 1 点，完成实验报告。

（二）错误记录

1、错误 1

（1）错误现象

从 CP0_INDEX 寄存器读出的数据不对。

（2）分析定位过程

由于此次实验没有 golden trace 进行跟踪，因此在仿真时将每一条指令的写回级情况进行输出，这样便于观察。逐条指令跟踪，首先发现第一个测试点里的 MFC0 指令结果不对。

（3）错误原因

原来的设计中对于 MTC0 指令的写 CP0 寄存器的编号放在 MTC0_waddr 信号中，默认 0 表示不是 MTC0 语句。在执行级看到 MTC0_waddr 为对应 CP0 寄存器编号时就进行写 CP0 寄存器。但是增加 CP0_INDEX 后，其编号恰好为 0，因此 CP0_INDEX 会被不断更新。

（4）修正效果

将 CP0_INDEX 的写控制进行如下修改，增加判断是否为 MTC0 指令信号。

```
//CP0_INDEX
always @(posedge clk)
begin
    if (!resetn)
        CP0_INDEX <= 32'h00000000;
    else if (EXE_TLBP)
        CP0_INDEX <= Index_out;
    else if (EXE_MTC0 && MTC0_waddr == 5'd0 && !EXE_clear && !EXE_stall)
        CP0_INDEX <= CP0_wdata & 32'h0000001f;
end
```

修改后通过第一个测试点。

（5）归纳总结

这个错误主要是由于在前面的实验时为了偷懒，由于没有用到 CP0 的 0 号寄存器就将其用作默认位，但此次实验中恰好增加了 CP0 的 0 号寄存器，因此出现了错误。这也提醒我以后写代码的时候不能偷懒而用小技巧，所有判断条件要写的更加完整严谨。

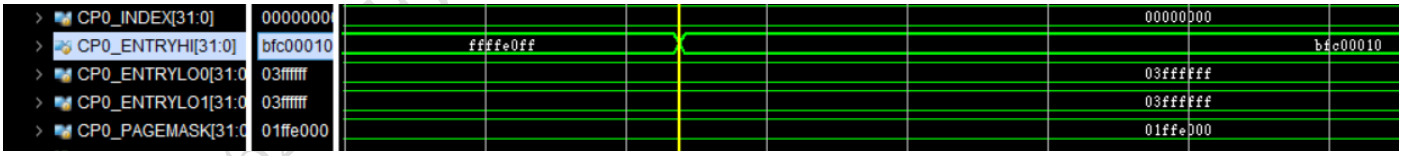
2、错误 2

（1）错误现象

在 TLBWI 后，TLBR 同一个 TLB 表项的内容结果不对。

（2）分析定位过程

根据输出的运行信息，发现在第 6 个测试点中，提前跳到了 inst_error，然后找到出错的指令为 TLBR，查看波形发现在 TLBR 时应该更改的四个 CP0 寄存器中，只有 CP0_ENTRYHI 进行了更新，其余三个均没有更新。波形如下：



（3）错误原因

查看控制 CP0_ENTRYLO0、CP0_ENTRYLO1 和 CP0_PAGEMASK 寄存器读写的代码发现，在写控制条件中将 TLBR 写成了 TLBP，所有导致这三个寄存器没有正常更新。

```
//CP0_PAGEMASK
always @(posedge clk)
begin
    if (!resetn)
        CP0_PAGEMASK <= 32'h00000000;
    else if (EXE_TLBP)
        CP0_PAGEMASK <= PageMask_out;
    else if (MTC0_waddr == 5'd5 && !EXE_clear && !EXE_stall)
        CP0_PAGEMASK <= CP0_wdata & 32'h01ffe000;
end
```

（4）修正效果

将字母 P 改为 R，重新运行通过该测试点。

（5）归纳总结

写代码太不仔细了，本身指令名称就相近，应该更加细心一点。

下面是三处在调试过程中发现测试程序有问题的地方。

3、错误 3

（1）错误现象

```
bfc00a20:   42000008    tlbp
bfc00a24:   40040000    mfc0    a0,$0
bfc00a28:   2408001f    li  t0,31
bfc00a2c:   1504000f    bne t0,a0,bfc00a6c <inst_erro
```

测试点 7 中指令 bfc00a2c 执行后 t0 和 a0 不相等导致跳转至 inst_error。

（2）分析定位过程

a0 是 TLBP 执行的结果。我的程序得出的 a0 是 29，但是标准的应该是 31。而 TLB 上次更新是在测试点 6 中，于是查看了测试点 6 的代码，发现在更改 TLB 中的表项时，前 29 项是用循环更新的，下标会自动更新，但是最后 3 项是单独更新的，同时更新的时候没有更改下标，导致第 29 项 TLB 被连续更改三次。结果如下：

而测试程序本意应该是后三次更改第 29、30、31 项。所以导致最后应该出现在第 31 项中的值出现在了第 29 项，从而导致了测试点 7 中的错误。

（3）错误原因

测试点 6 中更新最后 3 项 TLB 时，下标没有每次加一。

4、错误 4

（1）错误现象

测试点 10 中，经过 fetch_tlb_invalid 处理后，取指指令仍然触发 TLB_invalid，从而后续程序执行出错。

（2）分析定位过程

查看了 fetch_tlb_invalid 的异常处理函数，如下：

```
214  fetch_tlb_invalid:
215      tlbp
216      mfc0 k0, c0_epc
217      la   k1, 0x33333020
218      bne  k0, k1, tlb_fail
219      nop
220      li    k0, (0xbfcdf<<6)|2  #valid
221      mtc0  k0, c0_entrylo0
222      la    k1, fetch_tlb_pc_2
223      srl   k1, 12
224      sll   k1, 6
225      andi  k1, k1, 2 #valid
226      mtc0  k1, c0_entrylo1
227      tlbwi
228      nop
229      nop
```

导致取指异常的虚地址为 0x33333020，第 13 位为 1，因此对应的 TLB 中应该为 EntryLo1 项，但是按照异常处理函数中处理后，k1 为 0，即写入 TLB 的对应 EntryLo1 的项为 0，从而下次取指仍然触发 TLB_invalid。

（3）错误原因

对比写入 CP0_ENTRYLO0 寄存器的内容，发现第 225 行应该是或操作而不是与。

5、错误 5

（1）错误现象

测试点 10 中，经过 fetch_tlb_invalid 处理后，下次取指时转换后的物理地址为 bfc00020，该地址的指令与该测试点毫无联系，而没能继续后续测试点中的指令。

（2）分析定位过程

导致 TLB_invalid 的虚地址为 0x33333020，在经过异常处理后，在 TLB 中增加了对应项，物理页框号为 0xbfc00，经过映射后对应的物理地址为 0xbfc00020,从而取到了 0xbfc00020 处的指令执行。

（3）错误原因

如果要在异常处理后跳转到测试点 10 中后面的指令继续执行，应该修改虚地址的偏移，使其等于跳转到的指令的偏移。

四、实验总结

（一）组员：袁峥

这次实验在编写代码前花了两个多小时先进行了仔细思考，争取克服以前的毛病，在设计的时候不仔细从而导致后续调试时问题百出。最终发现好像多考虑了一些情况，比如设计中在完成一条 TLBWI、TLBR 或者 MTC0 修改了 CP0_ENTRYHI 后，会让后续指令重新取指。因为上述指令会修改 TLB 和 CP0_ENTRYHI 等内容，如果在完成前让后续指令取指，可能导致查找的 TLB 内容或者 CP0_ENTRYHI 的 ASID 域不正确，从而出现取指错误。但这些情况在最后的测试程序没有出现。

这次实验代码的修改量还是有些大的，前四级流水、阻塞控制模块和 mycpu_top 模块都进行了配套修改，但可能由于在编写代码前考虑的比较仔细，所以在调试过程中发现的错误不多，而且全都是笔误类的小错误，一看

到就能够知道错误的原因。倒是发现测试程序的错误比较费时，需要同时判断是测试程序的问题还是自己的 CPU 的问题，而且由于没有标准的 golden trace 进行比较，许多内容需要逐条手算，找到错误的地方比较慢。

整体来说还算顺利，花了差不多一整天把这次的实验全部完成了。