实验 13 报告

学号：2017K8009922027 2017K8009929011
姓名：张磊、郭豪
箱子号：39

一、实验任务（10%）

任务：a) CPU 增加 TLBR、TLBWI、TLBP 指令；

b) CPU 增加 index、entryhi、entrylo0、entrylo1 寄存器；

c) CPU 增加 16 项 TLB 结构支持，支持页的大小为 4KB；

d) 运行专用功能测试 tlb_func，要求通过前 6 项测试；

二、实验设计（40%）

（一）总体设计思路

1. 将 lab12 设计好的 TLB 模块添加到 CPU 中，为了方便 CP0 寄存器的读写，选择放在写回级；
2. 在之前的 CP0 寄存器中补上 ENTRYHI、ENTRYLO0、ENTRYLO1、INDEX 寄存器；
3. 利用 eret 指令刷新流水线的方法，将 tlbwi、tlbr 指令后的所有指令刷空，解决读写相关；

（三）重要模块 1 设计：CP0 寄存器模块；

1、工作原理

采用之前的设计方法，增加本次实验需要的几个 CP0 寄存器（以 Entrylo1 为例）；

```
// CP0_ENTRYLO1+++++++++++++++++++++++++++++++++++++
reg [31:26] cp0_entrylo1_zero;
always @(posedge clk)
begin
    if(rst)
        cp0_entrylo1_zero <= 6'b0;
end

reg [25:6] cp0_entrylo1_pfn;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO1_ADDR)
        cp0_entrylo1_pfn <= cp0_wdata[25:6];
    else if(tlbp & s0_found)
        cp0_entrylo1_pfn <= s0_pfn;
    else if(tlbp & s1_found)
        cp0_entrylo1_pfn <= s1_pfn;
    else if(tlbr)
        cp0_entrylo1_pfn <= r_pfn1;
end
```

```
reg [5:3] cp0_entrylo1_c;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO1_ADDR)
        cp0_entrylo1_c <= cp0_wdata[5:3];
    else if(tlbp & s0_found)
        cp0_entrylo1_c <= s0_c;
    else if(tlbp & s1_found)
        cp0_entrylo1_c <= s1_c;
    else if(tlbr)
        cp0_entrylo1_c <= r_c1;
end

reg cp0_entrylo1_d;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO1_ADDR)
        cp0_entrylo1_d <= cp0_wdata[2];
    else if(tlbp & s0_found)
        cp0_entrylo1_d <= s0_d;
    else if(tlbp & s1_found)
        cp0_entrylo1_d <= s1_d;
    else if(tlbr)
        cp0_entrylo1_d <= r_d1;
end
```

```
reg cp0_entrylo1_v;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO1_ADDR)
        cp0_entrylo1_v <= cp0_wdata[1];
    else if(tlbp & s0_found)
        cp0_entrylo1_v <= s0_v;
    else if(tlbp & s1_found)
        cp0_entrylo1_v <= s1_v;
    else if(tlbr)
        cp0_entrylo1_v <= r_v1;
end

reg cp0_entrylo1_g;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO1_ADDR)
        cp0_entrylo1_g <= cp0_wdata[0];
    else if(tlbr)
        cp0_entrylo1_g <= r_g;
end

assign cp0_entrylo1 =
{
        cp0_entrylo1_zero,
        cp0_entrylo1_pfn,
        cp0_entrylo1_c,
        cp0_entrylo1_v,
        cp0_entrylo1_d,
        cp0_entrylo1_g
};
```

2、功能描述

TLB 配套 CP0 寄存器；

（四）重要模块 2 设计：TLB 模块；

1、工作原理

将 lab12 设计好的 TLB 加入 CPU 中；

```
tlb u_tlb(
    .clk                (clk),
    // search port 0
    .s0_vpn2            (s0_vpn2),
    .s0_odd_page        (s0_odd_page),
    .s0_asid            (s0_asid),
    .s0_found           (s0_found),
    .s0_index           (s0_index),
    .s0_pfn             (s0_pfn),
    .s0_c               (s0_c),
    .s0_d               (s0_d),
    .s0_v               (s0_v),

    // search port 1
    .s1_vpn2            (s1_vpn2),
    .s1_odd_page        (s1_odd_page),
    .s1_asid            (s1_asid),
    .s1_found           (s1_found),
    .s1_index           (s1_index),
    .s1_pfn             (s1_pfn),
    .s1_c               (s1_c),
    .s1_d               (s1_d),
    .s1_v               (s1_v),

    // write port
    .we                 (we),
    //w(rite) e(nable)
    .w_index            (w_index),
    .w_vpn2             (w_vpn2),
    .w_asid             (w_asid),
    .w_g                (w_g),
    .w_pfn0             (w_pfn0),
    .w_c0               (w_c0),
    .w_d0               (w_d0),
    .w_v0               (w_v0),
    .w_pfn1             (w_pfn1),
    .w_c1               (w_c1),
    .w_d1               (w_d1),
    .w_v1               (w_v1),
```

```
    // read port
    .r_index            (r_index),
    .r_vpn2             (r_vpn2),
    .r_asid             (r_asid),
    .r_g                (r_g),
    .r_pfn0             (r_pfn0),
    .r_c0               (r_c0),
    .r_d0               (r_d0),
    .r_v0               (r_v0),
    .r_pfn1             (r_pfn1),
    .r_c1               (r_c1),
    .r_d1               (r_d1),
    .r_v1               (r_v1)
);
```

2、功能描述

将 TLB 模块放在写回级；

三、实验过程（50%）

（一）实验流水账

1. 2019 年 12 月 04 日 13:00 - 19:00 完成 rtl 代码编写；
2. 2019 年 12 月 04 日 19:00-21:30 完成了 bug 的处理以及上板验证；

（二）错误记录

1、错误 1：波形停止卡在 PC 等于 0xbfc00bcc 的地方

（1）错误现象

```
----------------------------------------------------------------
[  62000 ns] Test is running, debug_wb_pc = 0xbfc00bc8
[  72000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[  82000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[  92000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 102000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 112000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 122000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 132000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 142000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
[ 152000 ns] Test is running, debug_wb_pc = 0xbfc00bcc
```

（2）分析定位过程

查看控制台显示的信息：

```
============================================================
Test begin!
----[   8745 ns] Number 8'd01 Functional Test Point PASS!!!
----[  14715 ns] Number 8'd02 Functional Test Point PASS!!!
----[  20775 ns] Number 8'd03 Functional Test Point PASS!!!
       [  22000 ns] Test is running, debug_wb_pc = 0xbfc00698
------------------------------------------------------------
[  24405 ns] Error(  0)!!! Occurred in number 8'd04 Functional Test Point!
------------------------------------------------------------
       [  32000 ns] Test is running, debug_wb_pc = 0xbfc009c8
       [  42000 ns] Test is running, debug_wb_pc = 0xbfc009cc
       [  52000 ns] Test is running, debug_wb_pc = 0xbfc009d0
------------------------------------------------------------
[  58725 ns] Error(  1)!!! Occurred in number 8'd05 Functional Test Point!
------------------------------------------------------------
```

发现通过了前 3 个测试点，于是查看 test.s 文件，找到对应出错的指令：

```
bfc00ba8: 3c0a0001  lui t2,0x1
/mnt/shared/tlb_func/inst/n6_tlbp.S:12
bfc00bac: 40800000  mtc0  zero,c0_index
/mnt/shared/tlb_func/inst/n6_tlbp.S:13
bfc00bb0: 3c08bfc0  lui t0,0xbfc0
bfc00bb4: 35084010  ori t0,t0,0x4010
/mnt/shared/tlb_func/inst/n6_tlbp.S:14
bfc00bb8: 40885000  mtc0  t0,c0_entryhi
/mnt/shared/tlb_func/inst/n6_tlbp.S:15
bfc00bbc: 42000008  tlbp
/mnt/shared/tlb_func/inst/n6_tlbp.S:16
bfc00bc0: 40040000  mfc0  a0,c0_index
/mnt/shared/tlb_func/inst/n6_tlbp.S:17
bfc00bc4: 24080002  li  t0,2
/mnt/shared/tlb_func/inst/n6_tlbp.S:18
bfc00bc8: 15040018  bne t0,a0,bfc00c2c <inst_error>
/mnt/shared/tlb_func/inst/n6_tlbp.S:19
bfc00bcc: 00000000  nop
```

发现时由于前一条 bne 指令执行出了问题，于是先前查找 a0,t0 对应的值，最后发现是 entrylo1 寄存器的 g 位写条件错了

（3）错误原因

Entrylo1 寄存器 g 位写条件写错

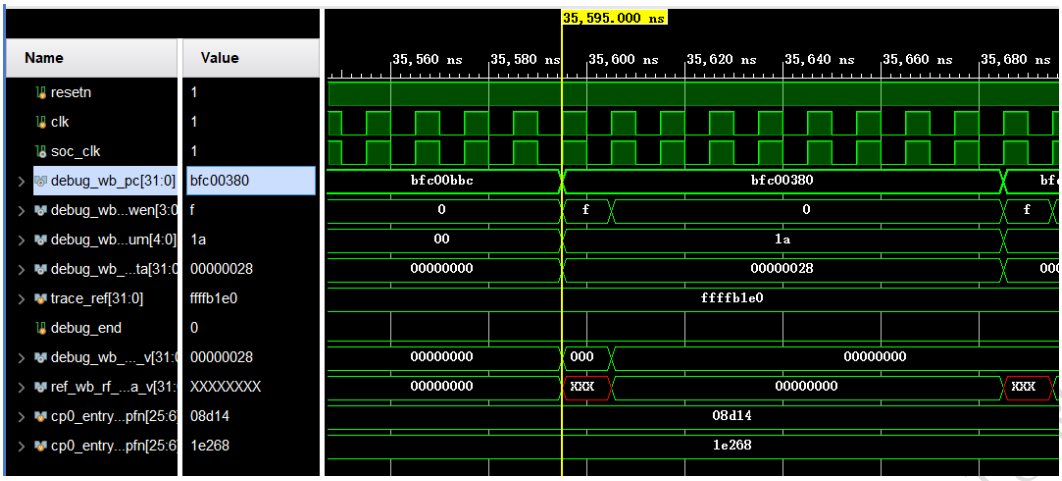
```
reg cp0_entrylo1_g;
always @(posedge clk)
begin
    if(mtc0_we && cp0_addr == `CP0_ENTRYLO0_ADDR)
        cp0_entrylo1_g <= cp0_wdata[0];
    else if(tlbr)
        cp0_entrylo1_g <= r_g;
end
```

（4）修正效果

修正后，顺利通过该测试点。

2、错误 2：触发了例外

（1）错误现象



（2）分析定位过程

查看 test.s 文件后发现，这个测试点不应该出现例外，于是向前查找，到底是哪一级触发了例外，最后在译码级发现，由于没有将新添加的指令加入保留指令例外的判断的逻辑中，所以触发了保留指令例外；

（3）错误原因

没有将新添加的指令加入保留指令例外的判断的逻辑中，所以触发了保留指令例外：

```
assign remd_inst = !(inst_addu | inst_subu | inst_slt | inst_sltu | inst_and | inst_or | inst_xor | inst_nor
| inst_sll | inst_srl | inst_sra | inst_addiu | inst_lui | inst_lw | inst_sw | inst_beq | inst_bne | inst_jal
| inst_jr | inst_add | inst_addi | inst_sub | inst_slti | inst_sltiu | inst_andi | inst_ori | inst_xori | inst_sllv
| inst_srlv | inst_srav | inst_mult | inst_multu | inst_div | inst_divu | inst_mfhi | inst_mflo | inst_mthi | inst_mtlo
| inst_bgez | inst_bgtz | inst_blez | inst_bltz | inst_j | inst_bltzal | inst_bgezal | inst_jalr | inst_lb | inst_lbu
| inst_lh | inst_lhu | inst_lwl | inst_lwr | inst_sb | inst_sh | inst_swl | inst_swr | inst_sys | inst_eret | inst_mfc0
| inst_mtc0 | inst_break);
```

（4）修正效果

修正后，顺利通过所有测试点。

```
assign remd_inst = !(inst_addu | inst_subu | inst_slt | inst_sltu | inst_and | inst_or | inst_xor | inst_nor
| inst_sll | inst_srl | inst_sra | inst_addiu | inst_lui | inst_lw | inst_sw | inst_beq | inst_bne | inst_jal
| inst_jr | inst_add | inst_addi | inst_sub | inst_slti | inst_sltiu | inst_andi | inst_ori | inst_xori | inst_sllv
| inst_srlv | inst_srav | inst_mult | inst_multu | inst_div | inst_divu | inst_mfhi | inst_mflo | inst_mthi | inst_mtlo
| inst_bgez | inst_bgtz | inst_blez | inst_bltz | inst_j | inst_bltzal | inst_bgezal | inst_jalr | inst_lb | inst_lbu
| inst_lh | inst_lhu | inst_lwl | inst_lwr | inst_sb | inst_sh | inst_swl | inst_swr | inst_sys | inst_eret | inst_mfc0
| inst_mtc0 | inst_break | inst_tlbp | inst_tlbwi | inst_tlbr);
```

四、实验总结（可选）

1. 这次实验比较简单，但是代码量还是挺大的，所以要尤其注意代码的编写，尤其是出现大量复制粘贴操作的时候；