实验 13 报告

学号：2017K80099290013　2017K8009929008

姓名：

箱子号：72

一、实验任务（10%）

CPU 增加 TLB 结构；

CPU 增加 TLBR，TLBWI，TLBP 指令；

CPU 增加 Index、EntryHi、EntryLo0、EntryLo1 寄存器；

二、实验设计（40%）

（一）总体设计思路

1、 模块连接：

* EXE 阶段连接 TLB 查找端口；

* WB 阶段连接 TLB 读写端口；

2、 TLBP 指令：

* 在 EXE 阶段接收 EntryHi 寄存器的内容，进行 TLB 查找；

* 查找结果随流水线传入 WB 阶段，写入 index 寄存器；

3、 TLBWI，TLBR 指令：

* 当 WB 阶段为 tlbwi 指令时，TLB 写使能置高，向 TLB 中写入数据；

* 当 WB 阶段为 tlbr 指令时，向 EntryHi、EntryLo0、EntryLo1 寄存器中写入 TLB 读数结果；

* 当流水线中出现 tlbr、tlbwi 指令时，给后续指令标记重取标志，带有重取标志的指令到达 WB 阶段时，刷新流水线，并重取此指令。

（二）重要模块设计：IF 阶段

1、部分代码

```
wire re_fe;
assign re_fe=id_to_if_tlb || exe_to_if_tlb || mem_to_if_tlb || wb_to_if_tlb;
```

图一：重取标志

2、功能描述：

* 当流水线中出现 tlbwi、tlbr 指令后，从 IF 阶段取出的指令将带有重取标记 re_fe；

（三）重要模块设计：ID 阶段：

1、部分代码：

```
assign inst_tlbp = op_d[6'h10] & func_d[6'h08] & rs_d[5'h10];
assign inst_tlbr = op_d[6'h10] & func_d[6'h01] & rs_d[5'h10];
assign inst_tlbwi  = op_d[6'h10] & func_d[6'h02] & rs_d[5'h10];
```

图二：译码逻辑

```
assign id_to_if_tlb=ds_valid&&(inst_tlbr||inst_tlbwi);
```

图三：传向 IF 阶段的重取标记判断

2、功能描述：

* 增加 tlbp、tlbr、tlbwi 指令译码逻辑；

* 以 ID 阶段为例（后续阶段的重取标记判断与之相同），若出现 tlbr、tlbwi 指令则要在 IF 阶段进行重取标记；

（四）重要模块设计：EXE 阶段

1、部分代码：

```
assign es_ready_go    =
            (
                ( (es_div_op && s_dout_tvalid==0)
                || (es_divu_op==1 && u_dout_tvalid==0)
                || block_mfhi_lo
                ||(data_req&&~data_addr_ok)
                ||(inst_tlbp&&mem_to_exe_mtc0_hi)
                )
            && (~es_valid || write_reg)
            )? 0 : 1;
```

图四：EXE 阶段 ready_go 判断逻辑

```
assign  s0_vpn2 = c0_EntryHi_vpn2;
assign  s0_odd_page = c0_EntryHi_vpn2x[1];
assign  s0_asid = c0_EntryHi_asid;
```

图五：TLB 查找输入

2、功能描述：

* 若 EXE 阶段为 tlbp 指令且 MEM 阶段的指令会更新 EntryHi 寄存器，则阻塞，等待 EntryHi 寄存器更新；

* 连接 TLB 查找端口，得到查询结果 s0_index、s0_found，并沿流水线向后传递；

（五）重要模块设计：WB 阶段

1、部分代码：

```
assign c0_EntryHi={c0_EntryHi_vpn2,c0_EntryHi_vpn2x,c0_EntryHi_0,c0_EntryHi_asid};
assign c0_Index = {c0_Index_p,c0_Index_0,c0_Index_index};
assign c0_EntryLo0 = { c0_EntryLo0_fill,
                       c0_EntryLo0_pfn,
                       c0_EntryLo0_c,
                       c0_EntryLo0_d,
                       c0_EntryLo0_v,
                       c0_EntryLo0_g};
assign c0_EntryLo1 = { c0_EntryLo1_fill,
                       c0_EntryLo1_pfn,
                       c0_EntryLo1_c,
                       c0_EntryLo1_d,
                       c0_EntryLo1_v,
                       c0_EntryLo1_g};
```

图六：增加 Index、EntryHi、EntryLo0、EntryLo1 寄存器

```
assign  ws_flush = ws_valid && (ws_ex_op || eret_flush ||re_fe);
```

图七：重填标记 re_fe

```
assign we = inst_tlbwi;
assign w_index =c0_Index_index;
assign w_vpn2 = c0_EntryHi_vpn2;
assign w_asid = c0_EntryHi_asid;
assign w_g = c0_EntryLo0_g&c0_EntryLo1_g;
assign w_pfn0 = c0_EntryLo0_pfn;
assign w_c0 = c0_EntryLo0_c;
assign w_d0 = c0_EntryLo0_d;
assign w_v0 = c0_EntryLo0_v;
assign w_pfn1 = c0_EntryLo1_pfn;
assign w_c1 = c0_EntryLo1_c;
assign w_d1 = c0_EntryLo1_d;
assign w_v1 = c0_EntryLo1_v;
```

图八：连接 TLB 写端口

```
else if(inst_tlbr)begin
    c0_EntryHi_vpn2<=r_vpn2;
    c0_EntryHi_asid<=r_asid;
    c0_EntryLo0_pfn<=r_pfn0;
    c0_EntryLo0_c<=r_c0;
    c0_EntryLo0_d<=r_d0;
    c0_EntryLo0_v<=r_v0;
    c0_EntryLo0_g<=r_g;
    c0_EntryLo1_pfn<=r_pfn1;
    c0_EntryLo1_c<=r_c1;
    c0_EntryLo1_d<=r_d1;
    c0_EntryLo1_v<=r_v1;
    c0_EntryLo1_g<=r_g;
end
```

图九：连接 TLB 读端口

3、功能描述：

* 增加 Index、EntryHi、EntryLo0、EntryLo1 寄存器，且与相应的 TLB 读写通道相连；

* 当重填标记到达 WB 阶段时，刷新流水线；

三、实验过程（50%）

（一）实验流水账

周一下午：阅读任务书，更新环境，整理实验思路；

周二到周六：逻辑设计，代码实现，仿真调试，上板运行；

周六晚上：小组交流代码实现和报告内容；

周一晚上：写报告；

（二）错误记录

1、错误 1：tlb index 寄存器的 p 位在 mtc0 的时候不能写

错误现象：仿真出错：

```
----------------------------------------------------------------
[   5695 ns] Error(  0)!!! Occurred in number 8'd01 Functional Test Point!
----------------------------------------------------------------
```

图十：错误现象 1

错误查找过程：

图十一：错误波形 1

根据报错时间找到此处的波形，查看对应的指令，发现是一次 index 寄存器的读写出错导致的跳转错误，对比正确值和实际值之后，意识到 p 位不能写。

修改方法：mtc0 更新 tlb index 寄存器的时候 p 位不写。

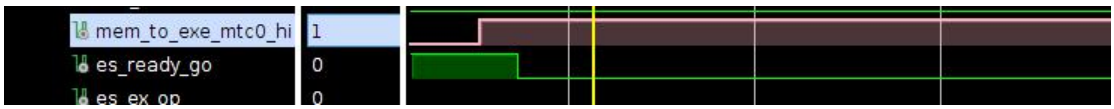
2、错误 2：mem 传递的阻塞信号没有考虑 mem 阶段的 valid

错误现象：仿真出错：

```
----[  38185 ns] Number 8'd05 Functional Test Point PASS!!!
[  42000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[  52000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[  62000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[  72000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[  82000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[  92000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 102000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 112000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 122000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 132000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 142000 ns] Test is running, debug_wb_pc = 0xbfc00a48
[ 152000 ns] Test is running, debug_wb_pc = 0xbfc00a48
```

图十二：错误现象 2

错误查找过程:



图十三：错误波形 2

从仿真情况可以看到一值卡在了一条指令，定位发现卡是因为这里有一个 tlbp 的寄存器相关，即当 exe 阶段位 tlbp 指令时，前一条指令要写 entryhi 寄存器，所以把传给 exe 阶段阻塞信号（上图中的 mem_to_exe_mtc0_hi）拉高，但是后续 ms_valid 因为阻塞拉低的时候，并没有随之拉低阻塞信号，导致一直阻塞。

修改方法：mem_to_exe_mtc0_hi 的拉高条件与上 ms_valid。