Lab 8 报告

学号 2020K8009908024
姓名 陈卓
箱子号 69

一、实验任务（10%）

TLB 模块设计：TLB 模块需要支持 TLBSRCH 指令的查找操作，要支持 TLBWR 和 TLBFILL 指令的写操作、TLB 模块需要支持 TLBRD 指令的读操作以及要支持 INVTLB 指令的查找、无效操作。

将 TLB 模块集成到 CPU 中，增加 TLB 相关的指令、CSR 寄存器以及相关异常，同时实现虚实地址转换功能。

二、实验设计（40%）

（一）总体设计思路

TLB 模块本身：读写操作非常简单，最主要的设计就是查找操作和无效操作的匹配。

增加的 TLB 指令包括 TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB 等五条指令。这需要在译码级增加相关的译码电路，以及增加相应的 CSR 寄存器。同时，这些指令会在不同流水级读写 TLB 与 CSR 模块，因此需要考虑数据相关。

在取指和 load\store 时，访问内存的虚地址需要经过虚实地址转换。虚实地址转换分为三种情况：直接地址翻译、直接映射窗口地址翻译和 TLB 地址翻译。虚地址会根据 CSR 寄存器中相关的域决定使用何种翻译模式。

（二）重要模块 1 设计：TLB 模块

1、工作原理

主要介绍一下查找和无效操作的匹配设计。

2、功能描述

首先是定义 TLB 的寄存器堆，这里并不采用长位宽的寄存器来表示一整个 TLB 项，而是每一个 TLB 项中具体的信号标识都分别采用一组寄存器。

```
//TLB_regfile
reg [TLBNUM-1:0]     tlb_e;
reg [TLBNUM-1:0]     tlb_ps4MB; //pagesize 1:4MB, 0:4KB
reg [ 18:0]          tlb_vppn [TLBNUM-1:0];
reg [ 9:0]           tlb_asid [TLBNUM-1:0];
reg                  tlb_g [TLBNUM-1:0];

reg [ 19:0]          tlb_ppn0 [TLBNUM-1:0];
reg [ 1:0]           tlb_plv0 [TLBNUM-1:0];
reg [ 1:0]           tlb_mat0 [TLBNUM-1:0];
reg                  tlb_d0 [TLBNUM-1:0];
reg                  tlb_v0 [TLBNUM-1:0];

reg [ 19:0]          tlb_ppn1 [TLBNUM-1:0];
reg [ 1:0]           tlb_plv1 [TLBNUM-1:0];
reg [ 1:0]           tlb_mat1 [TLBNUM-1:0];
reg                  tlb_d1 [TLBNUM-1:0];
reg                  tlb_v1 [TLBNUM-1:0];
```

查找功能的匹配采用并行查找的方法。这里使用了 generate 块来避免重复定义 32 个类似的信号。

```
genvar tlb_i;
generate for (tlb_i = 0; tlb_i < TLBNUM; tlb_i = tlb_i + 1) begin:gen_tlb

   assign match0[tlb_i] = (s0_vppn[18:10]==tlb_vppn[tlb_i][18:10])
                        && (tlb_ps4MB[tlb_i] || s0_vppn[9:0]==tlb_vppn[tlb_i][9:0])
                        && ((s0_asid==tlb_asid[tlb_i]) || tlb_g[tlb_i]);
   assign match1[tlb_i] = (s1_vppn[18:10]==tlb_vppn[tlb_i][18:10])
                        && (tlb_ps4MB[tlb_i] || s1_vppn[9:0]==tlb_vppn[tlb_i][9:0])
                        && ((s1_asid==tlb_asid[tlb_i]) || tlb_g[tlb_i]);
```

其次是无效操作的设计，这里采用讲义上建议的方法，分成了四种情况来组合成不同的 OP 对应的情况。

```
//invtlb:
 assign cond[tlb_i][0] =~tlb_g[tlb_i];
 assign cond[tlb_i][1] = tlb_g[tlb_i];
 assign cond[tlb_i][2] = s1_asid == tlb_asid[tlb_i];
 assign cond[tlb_i][3] = (s1_vppn[18:10]==tlb_vppn[tlb_i][18:10])
                       && (tlb_ps4MB[tlb_i]||s1_vppn[9:0]==tlb_vppn[tlb_i][ 9: 0]);
 assign inv_match[tlb_i] =          ((invtlb_op==0||invtlb_op==1) & (cond[tlb_i][0] || cond[tlb_i][1]))
                                    ||((invtlb_op==2) & (cond[tlb_i][1]))
                                    ||((invtlb_op==3) & (cond[tlb_i][0]))
                                    ||((invtlb_op==4) & (cond[tlb_i][0]) & (cond[tlb_i][2]))
                                    ||((invtlb_op==5) & (cond[tlb_i][0]) & cond[tlb_i][2] & cond[tlb_i][3])
                                    ||((invtlb_op==6) & (cond[tlb_i][1] | cond[tlb_i][2]) & cond[tlb_i][3]);
```

值得注意的是，本来想要把写操作写到 generate 外面来更加合理，但是无效操作和写操作都要对 tlb_e 信号赋值，会存在多驱动的问题，因此写在了同一个 always 块里面。

```
//Write
always @(posedge clk) begin
    if (we && w_index == tlb_i) begin
        tlb_vppn[tlb_i] <= w_vppn;
        tlb_asid[tlb_i] <= w_asid;
        tlb_g   [tlb_i] <= w_g;
        tlb_e   [tlb_i] <= w_e;
        tlb_ps4MB[tlb_i]<= (w_ps == 6'd22);

        tlb_ppn0[tlb_i] <= w_ppn0;
        tlb_plv0  [tlb_i] <= w_plv0;
        tlb_mat0  [tlb_i] <= w_mat0;
        tlb_d0  [tlb_i] <= w_d0;
        tlb_v0  [tlb_i] <= w_v0;

        tlb_ppn1[tlb_i] <= w_ppn1;
        tlb_plv1  [tlb_i] <= w_plv1;
        tlb_mat1  [tlb_i] <= w_mat1;
        tlb_d1  [tlb_i] <= w_d1;
        tlb_v1  [tlb_i] <= w_v1;
    end else if(inv_match[tlb_i] & invtlb_valid)begin
        tlb_e[tlb_i] <= 1'b0;
    end
end

end endgenerate
```

（三）重要模块 2 设计：CSR 中与 TLB 相关的寄存器

1、工作原理

此次实验增加了 TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY 和 DMW CSR 寄存器，用于联系 TLB 与 CPU 的联系以及地址转换。

2、功能描述

TLBIDX 寄存器用于存储 TLB 表项索引值、PS 域以及 TLB 表项是否为空。

TLBEHI 寄存器用于存储 TLB 表项高位部分虚页号相关的信息。

TLBELO0、TLBELO1 寄存器用于存储 TLB 表项地位部分物理页号等信息。

ASID 寄存器用于存储访存操作和 TLB 指令的地址空间标识符信息。

TLBRENTRY 寄存器用于配置 TLB 重填例外的入口地址。

DMW 寄存器参与直接映射地址翻译模式。

此外，csr_pc 也做了调整。当例外为 TLB 重填例外时，例外入口地址为 TLBRENTRY。

（四）重要模块 3 设计：TLB 指令

1、工作原理

此次实验增加了 TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTL 等指令。

2、功能描述

TLBSRCH 指令在执行级通过 ASID 和 TLBEHI 信息查找 TLB 并更新 TLBIDX 寄存器。TLBRD 在写回级读 TLB 并更新相关的 CSR 寄存器。TLBWR 和 TLBFILL 在写回级将 CSR 中 TLB 相关信息填入 LB 模块，二者的区别是 TLBWR 指定了写入 TLB 表项的位置，而 TLBFILL 由硬件随机选择 TLB 表项填入。对于硬件随机选择，我们的做法是 reset 后从 0 开始循环，以此作为 TLBFILL 的 TLB 表项位置。INVTLB 指令在执行级读取 TLB 并进行无效 TLB 操作。

此外，还需要处理写后读相关的问题。这里有两种情况。一种是 TLBSRCH 指令在 EX 级会通过 CSR 寄存器的内容查找 TLB，但 CSRWR、CSRXCHG 或 TLBRD 指令会在写回级修改这些寄存器。我们通过阻塞 TLBSRCH 在译码级，直至 CSRWR、CSRXCHG、TLBRD 指令执行完毕。另一种情况比较复杂，我们采用了讲义里建议的标志重取法。

（五）重要模块 4 设计：虚实地址转换

1、工作原理

虚实地址转换分为三种情况：直接地址翻译、直接映射窗口地址翻译和 TLB 地址翻译，将取指级和执行级发送的虚地址转为物理地址，再发送给内存。我们将虚实地址转换单独作为一个模块实现，同时负责判断是否发生 TLB 相关的例外。取指级和执行级分别有一个虚实地址转换模块，分别与 CSR 模块相连。

2、功能描述

模块的接口如下：

```
module vaddr_transfer(
    input  [31:0] va,//虚地址
    input  [ 2:0] inst_op,//{load.store,if}
    output [31:0] pa,//物理地址
    output [ 5:0] tlb_ex_bus,//{PME,PPE,PIS,PIL,PIF,TLBR}
    output wire adem_judge,
    //tlb查询
    output [18:0] s_vppn,
    output        s_va_bit12,
    output [ 9:0] s_asid,
    input         s_found,
    input  [ 3:0] s_index,
    input  [19:0] s_ppn,
    input  [ 5:0] s_ps,
    input  [ 1:0] s_plv,
    input  [ 1:0] s_mat,
    input         s_d,
    input         s_v,
    //crmd
    input  [31:0] csr_asid,
    input  [31:0] csr_crmd,
    //dmw
    output dmw_hit,
    input  [31:0] csr_dmw0,
    input  [31:0] csr_dmw1
```

当 CRMD 的 DA 域为 1，PG 域为 0 时，为直接地址翻译模式，物理地址就是虚拟地址。

当不处于直接地址翻译模式时，若在两个 DMW 寄存器中查询到相关信息时，为直接地址映射翻译，物理地址由 DMW 转化而来。若未在 DMW 中查到相关信息，则在 TLB 中查询，物理地址为 TLB 中查询到的物理地址。

此外，还需要处理 TLB 相关的例外。需要注意的有两点：第一，模块在判断例外时可能同时判断出多种例外，此时就需分“优先级”（实质上是判断异常发生的条件错误）。这里，重填例外的优先级最高，PIS，PIL，PIF 次之，PPE 再次之，PME 优先级最低。第二，当处于直接地址映射或直接地址翻译时，不应产生 TLB 例外。

```
assign tlb_ex_bus = {direct ? 1'b0 : ~dmw_hit & inst_op[1] & s_found & s_v & csr_crmd[1:0] <= s_plv & ~s_d,//PME
                     direct ? 1'b0 : ~dmw_hit & {|inst_op} & s_found & s_v & csr_crmd[1:0] > s_plv,//PPE
                     direct ? 1'b0 : ~dmw_hit & inst_op[1] & s_found & ~s_v,//PIS
                     direct ? 1'b0 : ~dmw_hit & inst_op[2] & s_found & ~s_v,//PIL
                     direct ? 1'b0 : ~dmw_hit & inst_op[0] & s_found & ~s_v,//PIF
                     direct ? 1'b0 : ~dmw_hit & {|inst_op} & ~s_found//TLBR
                     };
```

传给 CSR 模块的 Ecode 和 Esubcode 也要增加这几个例外的情况。

三、实验过程（50%）

（一）实验流水账

第一周：TLB 模块设计。用了周三一晚上的时间写完并通过了仿真和上板。

第二周：添加 TLB 指令、CSR 相关寄存器、虚实地址转换。周三周四周五完成

第三周：增加 TLB 相关例外。周日周一完成。

（二）错误记录

1、错误 1：重取标志法错误

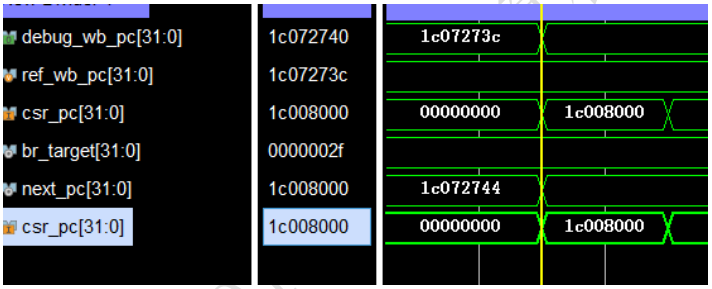
（1）错误现象

```
  reference: PC = 0x1c072740, wb_rf_wnum = 0x1e, wb_rf_wdata = 0x00000000
  mycpu    : PC = 0x1c008000, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00000000
------------------------------------------------------------------------
```

PC 出错

（2）分析定位过程

查看汇编指令，发现其上一条指令为 CSRXCHG，会发生标志重取。但重取后跳转到的地址为例外入口地址而不是写回级的 PC。



（3）错误原因

标志重取复用了例外的 wb_ex 以清空流水级，但却忽略了 next_pc 应该赋值成 wb_pc。

（4）修正效果

```
assign fake_ex = !wb_ex_r & wb_ex;

assign csr_pc = ertn_flush ? csr_era_pc :
                wb_ex_r ? (wb_ecode == 6'h3f ? csr_tlbrentry_rvalue : csr_eentry_rvalue) :
                fake_ex ? wb_pc : 32'h00000000;
```

增加了 wb_ex_r 和 fake_ex。当发生标志重取时，wb_ex 拉高以清空流水级，但 wb_ex_r 不拉高，意思是真正的异常而不是标志重取这个“假异常”。此时 fake_ex 也会拉高，最后 csr_pc 的值为 wb_pc。

2、错误 2：CSR.CRMD DATF 域 DATM 域错误

1）错误现象

```
reference: PC = 0x1c07c7c8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000088
mycpu    : PC = 0x1c07c7c8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000008
```

wdata 出错

（2）分析定位过程

汇编指令为：csrxchg　$r12,$r13,0x0。

0x0 对应的 CSR 寄存器为 CRMD，对比后发现出错的域为 DATM 域。

（3）错误原因

讲义曾提到 DATM 和 DATF 域可以暂时不用考虑，但我误以为是设为常值即可，而没有设置为软件可写的状态。

（4）修正效果

在 CSR 模块中给 CRMD 寄存器增加了 DATF 域和 DATM 域的写操作，使 CSRWR 和 CSRXCHG 对这两个域可写。

3、错误 3：未判断 TLB 例外的“优先级”

1）错误现象

```
reference: PC = 0x1c008380, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000002
mycpu    : PC = 0x1c008380, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000006
```

wdata 出错

（2）分析定位过程

汇编指令为：

```
1c00837c:   0400140c    csrrd   $r12,0x5
1c008380:   0044c18c    srli.w  $r12,$r12,0x10
```

根据上下文猜测是 csrrd 出错。0x5 对应的 CSR 寄存器为 ESTAT，出错的域为 ECODE。

ECODE 域只有在出异常时才会记录写回级传入的 ECODE。但查看后发现目前实现的异常中没有为 6 的 ECODE，但 PIS 和 PME 可以组合出一个为 6 的 ECODE。

（3）错误原因

TLB 异常判断条件错误。

```
assign tlb_ex_bus = {direct ? 1'b0 : ~dmw_hit & inst_op[1] & s_found &  ~s_d,//PME
                     direct ? 1'b0 : ~dmw_hit & {|inst_op} & s_found & csr_crmd[1:0] > s_plv,//PPE
                     direct ? 1'b0 : ~dmw_hit & inst_op[1] & s_found & ~s_v,//PIS
                     direct ? 1'b0 : ~dmw_hit & inst_op[2] & s_found & ~s_v,//PIL
                     direct ? 1'b0 : ~dmw_hit & inst_op[0] & s_found & ~s_v,//PIF
                     direct ? 1'b0 : ~dmw_hit & {|inst_op} & ~s_found//TLBR
                     };
```

（4）修正效果

完善了各 TLB 异常的触发条件。（已在实验设计中说明）

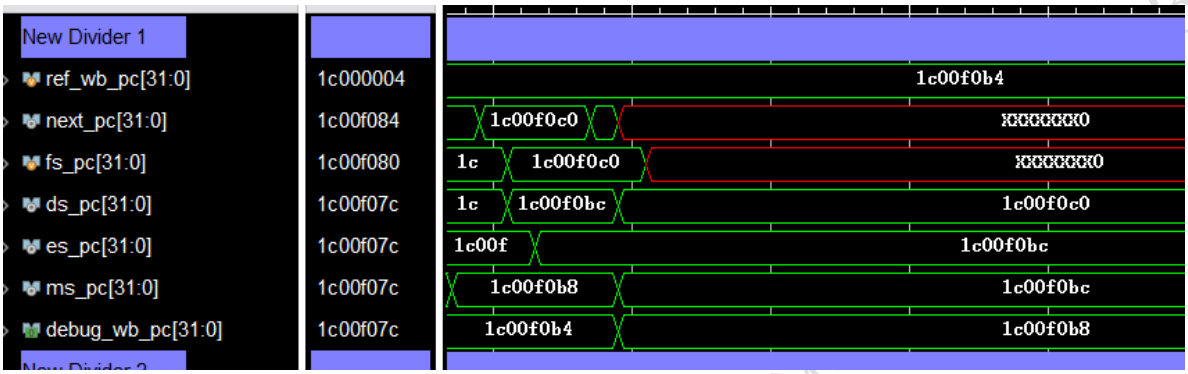
4、错误 4：ADEM 判断条件错误

1）错误现象

```
Test begin!
        [  22000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  32000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  42000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  52000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  62000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  72000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
        [  82000 ns] Test is running, debug_wb_pc = 0x1c00f0b8
```

测试程序在 1c00f0b8 处循环。

（2）分析定位过程



调出各级 PC，发现 fs_pc 和 next_pc 错误。

此时由于 wb_ex 拉高，因此 next_pc 被赋值为 csr_pc。而 CSR 寄存器还未初始化，因此 next_pc 出错。但问题不是 CSR 错误，而应该是异常触发错误。

调出所有异常，发现 adem_ex 被拉高了。但此时访存地址为 bfaff030，确实超出了访存范围。后来询问同学得知 adem_ex 的触发条件还包括不是直接地址翻译模式且直接地址映射未命中。

（3）错误原因

在直接地址翻译和直接地址映射情况下不应触发 adem_ex。

（4）修正效果

增加了是否处于直接地址翻译和直接地址映射的判断条件。

四、实验总结（可选）

无