实验 13 ～ 15 报告

学号	2019K8009929019	2019K8009929026
姓名	桂庭辉	高梓源
箱子号	44	

1 实验任务

2 实验设计

2.1 重要模块设计：TLB

2.1.1 功能描述

TLB 的功能在于在硬件上记录部分页表项以加快页表查询速度，具体需要记录指定项数的页表项 PTE，根据 LoongArch 32 位精简版 TLB 需支持的指令，设计同步写、异步读的读写端口以及用于取指、访存两处地址转换的查找功能。

TLB 模块的接口与讲义一致，此处不再赘述。

2.1.2 细节实现

TLB 的读写逻辑与通用寄存器堆 regfile 的逻辑类似。读操作根据 `r_index` 异步读取 TLB 记录页表项各域即可，写操作在 `we` 拉高时根据 `w_index` 将页表项各域同步更新记录。

查找过程需要将输入的虚页号与 TLB 记录的虚页号相匹配，生成各表项匹配成功与否的 `match0/1` 信号逻辑与讲义一致，具体而言对每个 TLB 表项需检查存在位 `e`、可见性 `g` 或 `asid` 匹配、虚页号相等与否，其中虚页号对于页大小为 4 MB 时比对高 9 位即可，而为 4 KB 时需匹配所有 19 位。

```
generate for (i = 0; i < TLBNUM; i = i + 1) begin
    assign match0[i] = tlb_e[i] && (tlb_g[i] || tlb_asid[i] == s0_asid) &&
                       // cmp high bits whatever ps is
                       s0_vppn[18:10] == tlb_vppn[i][18:10]            &&
                       // if ps == 4 KB, cmp low bits
                      (s0_vppn[ 9: 0] == tlb_vppn[i][ 9: 0] || tlb_ps[i]);
    assign match1[i] = tlb_e[i] && (tlb_g[i] || tlb_asid[i] == s1_asid) &&
                       s1_vppn[18:10] == tlb_vppn[i][18:10]            &&
                      (s1_vppn[ 9: 0] == tlb_vppn[i][ 9: 0] || tlb_ps[i]);
end
endgenerate
```

是否查找成功信号 `found` 生成逻辑简单，只需将 `match` 按位或（等效于 $\neq 0$）即可。由 `match` 信号获取匹配的 TLB 表项索引 `index` 可以使用多路选择器实现，但在个人设计中未能编写出兼顾代码可读

性与参数化（即选择器路数由参数 TLBNUM 决定）的多路选择器，故而以另一种形式实现该过程，即下文将要讨论的模块 mylog2。

获取到查找索引 s_index 后，由于一个 TLB 表项记录两个物理页，接下来需要判断奇偶页的选择。标记奇偶页的虚地址位为 va[PS]，对于 4 KB 页，其为虚地址的第 12 位，对于 4 MB 页，其为虚地址的第 22 页，即查找端口输入的虚页号第 9 位。根据页大小获取奇偶标志位后即可选出查找结果的物理页。

```
assign s0_ppg_sel = tlb_ps[s0_index] ? s0_vppn[9] : s0_va_bit12;
assign s0_ps      = tlb_ps[s0_index] ? 6'd22 : 6'd12;

assign s0_ppn   = s0_ppg_sel ? tlb_ppn1[s0_index] : tlb_ppn0[s0_index];
assign s0_plv   = s0_ppg_sel ? tlb_plv1[s0_index] : tlb_plv0[s0_index];
assign s0_mat   = s0_ppg_sel ? tlb_mat1[s0_index] : tlb_mat0[s0_index];
assign s0_d     = s0_ppg_sel ? tlb_d1  [s0_index] : tlb_d0  [s0_index];
assign s0_v     = s0_ppg_sel ? tlb_v1  [s0_index] : tlb_v0  [s0_index];
```

接下来考虑 INVTLB 指令功能的实现，如讲义所言，将其查找逻辑拆分为四个子匹配逻辑：

```
generate for (i = 0; i < TLBNUM; i = i + 1) begin
   assign inv_match[0][i] = ~tlb_g[i];
   assign inv_match[1][i] =  tlb_g[i];
   assign inv_match[2][i] = s1_asid == tlb_asid[i];
   assign inv_match[3][i] = s1_vppn[18:10] == tlb_vppn[i][18:10]            &&
                            (s1_vppn[ 9: 0] == tlb_vppn[i][ 9: 0] || tlb_ps[i]);
end
endgenerate
```

由子匹配结果容易获得一套各 invtlb_op 对应的掩码，INVTLB 指令的无效实现只需要将 op 对应的掩码中为 1 处的 tlb_e 域抹零即可。

```
assign inv_op_mask[0] = 16'b0;
assign inv_op_mask[1] = 16'b0;
assign inv_op_mask[2] = inv_match[1];
assign inv_op_mask[3] = inv_match[0];
assign inv_op_mask[4] = inv_match[0] & inv_match[2];
assign inv_op_mask[5] = inv_match[0] & inv_match[2] & inv_match[3];
assign inv_op_mask[6] = (inv_match[0] | inv_match[2]) & inv_match[3];

always @ (posedge clk) begin
    if (we) begin
        // write logic
    end else begin
        tlb_e <= ~inv_op_mask[invtlb_op] & tlb_e;
    end
end
```

2.2 重要模块设计：mylog2

2.2.1 功能描述与设计实现

设计本意是检测输入中的 1，返回其位置。思路可简单概括为折半查找。

折半查找的每一步检索结果与输出结果自高到低每一位一一对应。以 32 位输入为例，输出应有 5 位，折半查找第一次判断 1 落在高 16 位还是低 16 位，若高则将输出最高位置 1，依此类推，每次折半判断的高 (1) 低 (0) 与作为结果的索引一一对应。判断 1 落在哪一半的实现通过判断高半部分是否有 1（是否不等于 0），由此在输入有多个 1 时进行了高位优先，最终获取的结果为最高 1 的索引，在数值上等同于运算 $\log_2$。

在具体实现上，对此前提到的高半部分的选取通过语法 `+:` 实现，由此需要获取到每次判断的基址。以 32 位为例，首次基址应为 16，判断输入的 `[16+:16]` 部分是否有 1。类似的对于任意[①]位宽 `WIDTH` 的输入，其索引位宽 `IDX_WIDTH` 为 `WIDTH` 对 2 取对数，首基址只需将 `IDX_WIDTH` -1 位置 1，其他位置 0 即可（即对 `WIDTH` 数值取半）。

```
localparam IDX_WIDTH = $clog2(WIDTH);

assign base[IDX_WIDTH-1] = {1'b1, {IDX_WIDTH-1{1'b0}}};
```

对于折半查找每层需要做的工作，(1) 根据基址判断高半部分是否有 1，生成结果 `res` 的相应位；(2) 计算下一层应使用的基址。前者的实现较为简单。后者归纳下来即高半部分有 1 则当前层所用基址加上 (1) 中检测宽度的一半，否则减去。此处的加减法具有一定的特殊性，记当前层对应位索引为 `i`，次层对应位为 `i-1`。加数或减数定为在 `i-1` 处为 1，其他位为 0。而被加/减数的 `i` 位以后均为 0，所以计算结果中 `i-1` 处必定为 1，根据这个结论可以发现不会出现连续借位的情况，即 `i` 位是上一层的 `i-1` 位，必定为 1 可以被借位，若发生加法则该位保持 1，若发生减法则被借位置 0。由此可以注意到 `base[i]` 与 `res[i]` 的值是一致的。

```
generate
    for (i = IDX_WIDTH-1; i > 0; i = i - 1) begin
        assign res[i] = src[base[i]+:({{IDX_WIDTH-1{1'b0}}, 1'b1} << i)] != 0;

        for (j = 0; j < IDX_WIDTH; j = j + 1) begin
            if (i == j) begin
                assign base[i - 1][j] = res[i];
            end else if (i - 1 == j) begin
                assign base[i - 1][j] = 1'b1;
            end else begin
                assign base[i - 1][j] = base[i][j];
            end
        end
    end
endgenerate
```

结果的末位与根据最后一次计算出的基址 `base[0]` 索引输入的结果一致。

[①]实际应用上最好指定位宽 `WIDTH` 为 2 的次幂，非 2 次幂时 `+:` 的越界问题我并未检验与解决。

```
assign res[0] = src[base[0]];
```

2.2.2 接口定义

表 1: mylog2 模块接口定义

名称	方向	位宽	描述
src	IN	WIDTH	res $= \log_2$ src
res	OUT	$\log_2$ WIDTH	

3 实验过程

3.1 实验流水账

2021.10.28 10:00 ~ 2021.10.28 10:30 ： 根据讲义内容与 gitee 仓库 tlb_verify[②]完成 TLB 初步设计，由于验证环境问题未完全验证设计。

2021.11.18 16:00 ~ 2021.11.18 16:45 ： 编写设计 mylog2 模块，根据正式实验环境通过验证。

3.2 错误记录

4 实验总结

[②]https://gitee.com/ucas-ca-edu-lab/tlb_verify