哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2024 春季

课程名称：计算机组成原理（实验）

实验名称：高速缓存器设计

实验性质：设计型

实验学时：4 地点：T2506

学生班级：22 级 4 班

学生学号：220110430

学生姓名：吴梓滔

作业成绩：

实验与创新实践教育中心制

2024 年 5 月

1、Cache 详细设计

要求：绘制 ICache 的状态转换图，并详细描述状态转移关系、转移条件、各状态的输入输出信号以及需要完成的操作。*若完成了附加题，则分别绘制 DCache 的读、写状态转换图，并配以文字详细描述相应的内容。

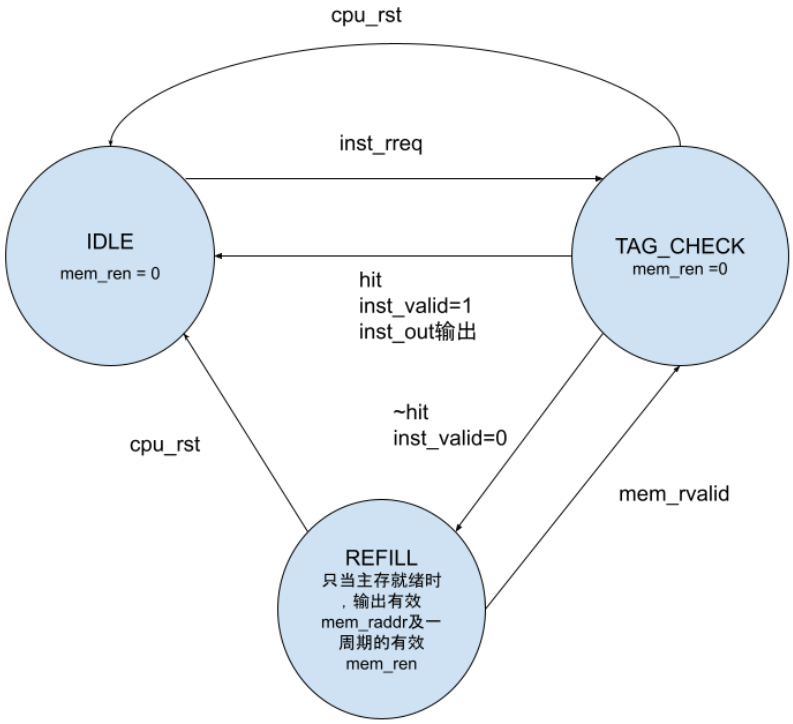

图 1 ICache 状态图

状态机的状态转移逻辑如图，一共定义三个状态：IDLE 为复位状态，TAG_CHECK 为检查是否命中的状态，REFILL 为未命中的处理状态。

IDLE 为复位状态，任何时候输入复位信号，或者一次完整的读取完成后，等待下一次读取时，都处于复位状态。该状态下，当接收到 inst_rreq 信号有效，即转入 TAG_CHECK 状态。

TAG_CHECK 状态为检查是否命中的状态。该状态停留一个周期，通过组合逻辑生成 hit 信号，根据 hit 信号的值立即转移到其他状态。若 hit 信号为 1，说明读 Cache 命中，立即转移至 IDLE 状态，同时 inst_out 和 inst_valid 信号将给出有效输出，完成一轮读取。若 hit 信号为 0，则判断未命中立即转移至 REFILL 状态。inst_valid 输出 0。

REFILL 状态为未命中时，从主存读取目标信息的状态。该状态会等待主存就绪信号 mem_rrdy 为 1 时，输出 1 个周期的 mem_ren=f,同时输出本次读取对应的有效 mem_raddr 信号。之后，该状态等待 mem_rvalid 信号，一旦 mem_rvalid=1，将转移至 TAG_CHECK 信号。

2、调试报告

要求：结合仿真波形截图对 ICache 作详细的时序分析，要求包含读命中、读缺失 2 种情形，且每种情形列举 2 个测试用例。*若完成了附加题，则需额外给出 DCache 的仿真波形截图及其详细文字分析，要求包含写命中、写缺失和 Uncached 访问 3 种情形。

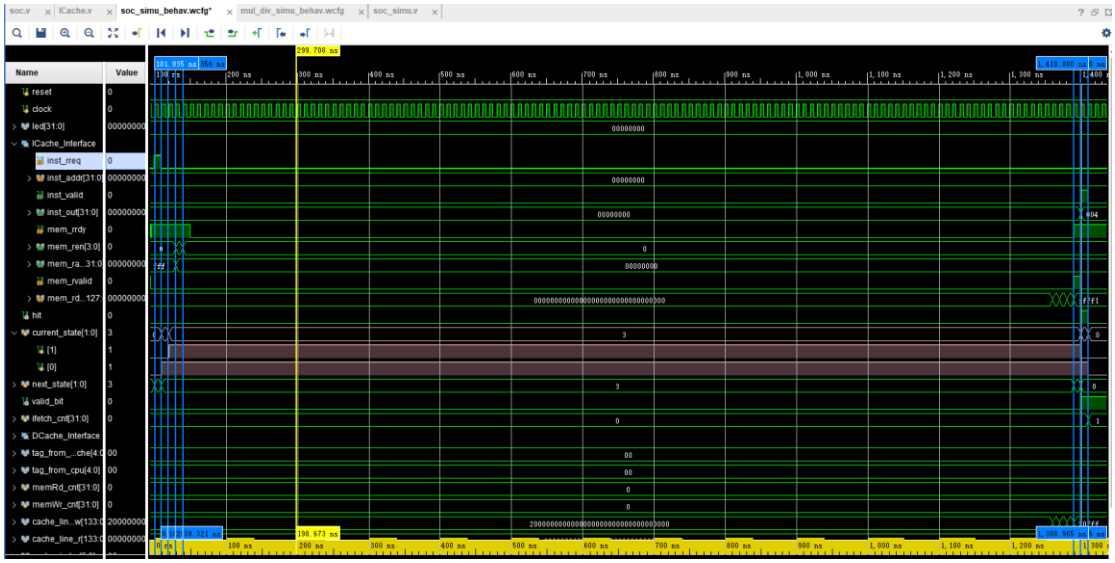

图 2-1 读缺失用例 1

图 2-2 读缺失用例 1（前半部分）

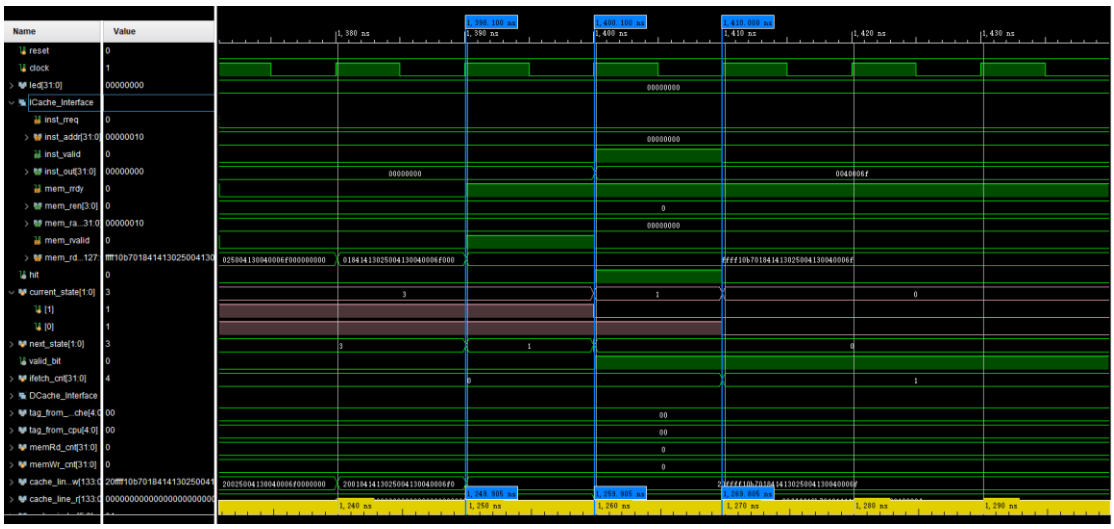

图 2-3 读缺失用例 1（后半部分）

图 2 展示了一个完整的读缺失的情形。

波形图信号说明：

1) inst_rreq 是由 cpu 输入到 ICache 模块的读取请求信号，inst_addr 是 cpu 输入到 ICache 模块的读取地址。Cpu 在请求读取时，会同步地给出一周期 inst_rreq 信号并更新 inst_addr 为目标地址。
2) inst_valid 为 ICache 模块输出给 cpu 的输出有效信号，inst_out 为 ICache 模块输出给 cpu 的数据内容。inst_valid 置 1 时，说明 ICache 向主存返回了有效的数据。
3) mem_rrdy 为主存输入到 ICache 的主存就绪指示，mem_rvalid 为主存输入 ICache 模块的数据有效信号，mem_rdata（128 位）为主存输入 ICache 的数据信号。mem_rvalid 信号为 1 时，说明主存向 ICache 输入了目标数据。
4) mem_ren 为 ICache 模块输出给主存的读使能信号，有 0 和 f 两个取值。mem_raddr 为 ICache 模块输出给主存的访问地址。当 mem_ren 给出 1 周期 f 信号，同时 mem_raddr 给出指定输出，主存就会开始读取 mem_raddr 地址所存的目标数据。
5) current_state 为当前状态信号。状态信号 0 对于 IDLE 状态，1 对于 TAG_CHECK 状态，3 对应 REFILL 状态。在 cache 的正常工作中，该信号有且只有两种变化模式：一是从 0 变成 1，1 保持 1 周期后变成 0，该模式说明读取命中。二是从 0 变成 1，1 保持 1 周期后变成 3，在 3 状态保持较长时间，再变成 1，1 保持 1 个周期后变成 0，该模式说明读缺失。本用例为第二种变化模式，说明读缺失。
6) hit 信号为读命中的判断信号。hit 是一些信号通过组合逻辑生成的。一旦 hit 为 1，说明可以正确地向 cpu 提供读取的数据。hit 信号用于辅助判断是否读命中，如果 inst_rreq 信号和 hit 信号相邻，说明读命中。如果二者的脉冲相隔较长时间，说明读缺失。同时，hit 信号也标志着一次完整的读取过程结束。
7) cache_line_w 信号为写入 cache 存储体的 cache 行信号，cache_line_r 信号为从 cache 存储体读出的 cache 行信号，这两个信号包括 128 位的 cache 数据块，和 5 位 tag 和 1 位 valid_bit 信号。
8) tag_from_cache 为从 cache 行信号读出的 tag 字段，tag_from_cpu 为从 cpu 输入的访

问地址中读出的 tag 字段。这两个字段相同是判断命中的条件之一。

9) valid_bit 为 cache_line_r 中的最高位。

时序分析：

在图 2-1 中，从左至右标注了 8 个时钟上升沿。前 5 个上升沿的细节如图 2-2 所示，后 3 个上升沿的细节如图 2-3 所示。中间持续较长时间的部分，为等待主存返回数据的时间。

1) 在第 1 个上升沿时，current_state=0,此时 ICache 等待 cpu 的读取请求。此时，inst_rreq 信号从 0 变成 1，inst_addr 保持 0，说明 cpu 发出了读取请求，读取地址为 0 的数据。在接下来一个周期中，inst_rreq=1,之后归 0。在此期间，由于 inst_rreq=1 且 current_satte=0,状态机应作出更新，next_state=1。
2) 在第 2 个上升沿时，状态机进行更新，current_state 变为 1，同时经过组合逻辑判断后，此时刻 hit=0，inst_valid=0,说明读缺失，next_state=3,状态机下一时刻应更新到状态 3，即 REFILL 状态。
3) 在第 3 个上升沿时，状态机更新，current_state 变为 3。
4) 在第 4 个上升沿时，由于 current_state=3,且此时 mem_rrdy=1，可以向主存读取数据。因此，mem_ren 从 0 更新为 f，mem_raddr 变为此时要读取的目标地址，为 cpu 发送到 Icache 的取指地址所在的块的基地址。在第 5 个上升沿，mem_rrdy 变为 0，高电平持续 1 个周期。此后，current_state 停留在 3，等待主存返回的信号。
5) 在第 6 个上升沿时，经过等待，current_state=3。此时刻，mem_rvalid 从 0 更新为 1，表示此刻开始 mem_rdata 也是有效的数据信号。主存将目标数据给到了 ICache 模块。
6) 在第 7 个上升沿时，由于 mem_rvalid=1,数据已经存入 ICache，此时可以更新状态至 TAG_CHECK 状态。current_state 由 3 变成 1，同时 hit 由 0 变成 1，inst_valid 由 0 变成 1，并且 inst_out 的值作出了更新。说明目标数据信号读到并且输出给了 cpu。
7) 在第 8 个上升沿时，inst_valid 持续一周期，此时 ICache 模块已经向 cpu 输出了目标数据，因此可以回到 IDLE 状态，并且撤去 inst_valid 等信号。current_state 变为 0，ICache 模块完成了一次的读取操作。

以上过程说明，ICache 能够正确地接受 cpu 的读取请求，判断是否命中，在缺失情况，到主存相应地址取回数据，并输出给 cpu。

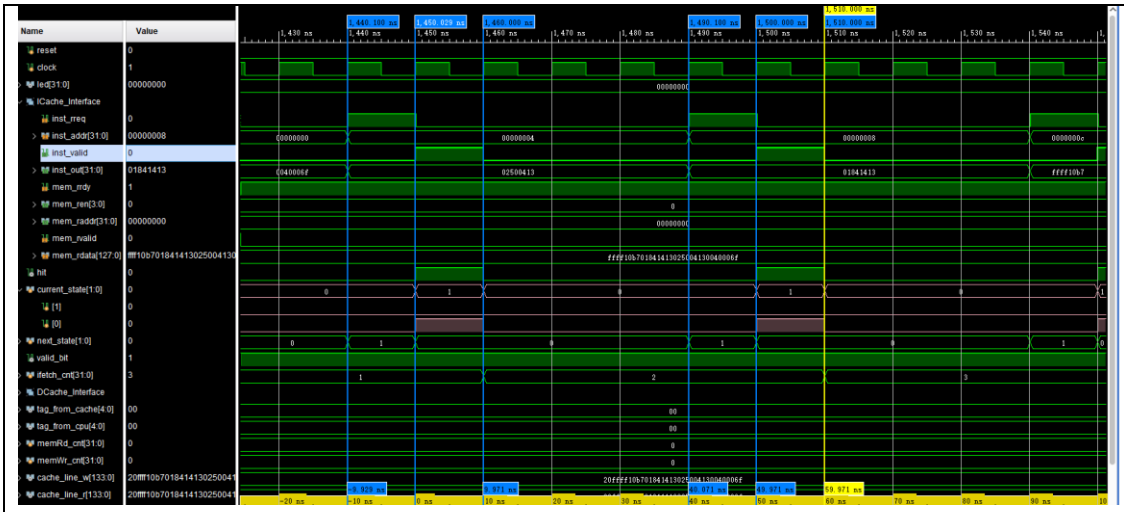

图 3 读命中用例

这两次读命中用例，为上一缺失用例后紧接着读取的。

图中从左至右标记了 6 次时钟上升沿。时序分析:

1) 在第 1 次时钟上升沿时，current_state=0,处于复位状态，等待 cpu 的读取请求。此时 inst_rreq 从 0 变 1，同时读取的地址进行了更新。此时由于 inst_rreq=1,状态机满足更新状态的条件，next_state 同步变为 1。
2) 在第 2 次时钟上升沿时，current_state 更新至 1，此时由于 tag_from_cpu 与 tag_from_cache 均为 00，valid_bit=1,可以说明 cache 命中，hit 信号由组合逻辑生成，进入 TAG_CHECK 状态后，立即同步更新为 1，同时 inst_valid 也更新为 1，此时的 inst_out 即为有效的目标数据，说明读取命中，并立即将数据返回给了 cpu。
3) 在第 3 次时钟上升沿时，由于此前的一个周期已经完成了数据的输出，一次读取结束。因此 current_state 回到 0，并且 inst_valid 置 0。此后状态机处于 IDLE 状态，等待 cpu 的下一次读取请求。
4) 在第 4 次时钟上升沿时，inst_rreq 从 0 变 1，同时 inst_addr 更新，此时 cpu 向 ICache 模块发出了第二次读取请求。
5) 在第 5 次时钟上升沿时，current_state 由 0 更新为 1，同时由于 tag_from_cpu 与 tag_from_cache 均为 00，valid_bit=1,可以说明 cache 命中，hit=1，inst_valid 同步更新为 1，该周期向 cpu 输出了有效的数据。
6) 在第 6 次时钟上升沿时，已经完成了读取，current_state 更新为 0，回到 IDLE 状态，inst_valid 也更新为 0，ICache 开始等待下一次读取请求。

这两个用例说明，当 Cache 内已经有目标数据时，ICache 模块能够正确地接受 cpu 的读取请求，并判断命中，在命中的情况下直接输出数据，完成一次读取。

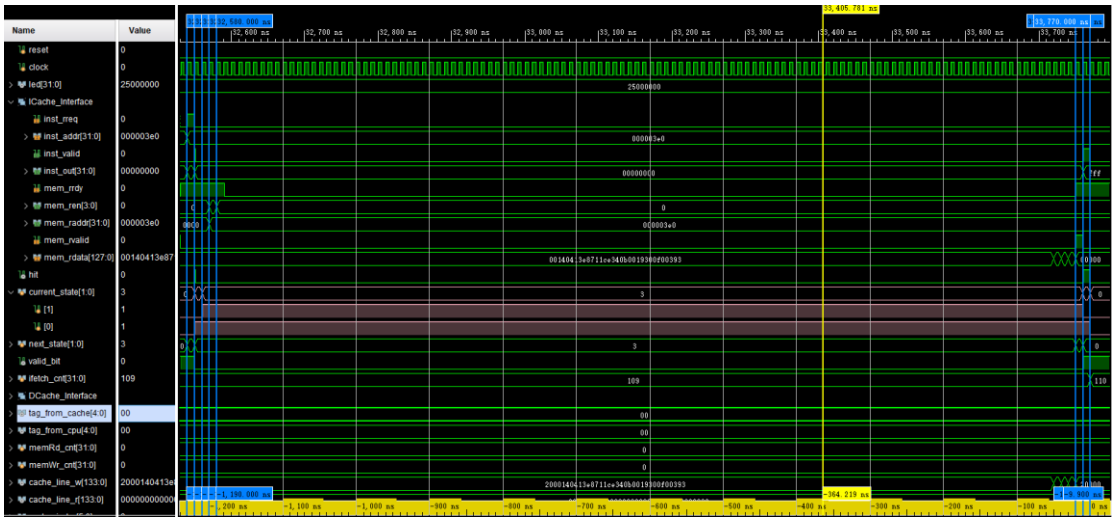

图 4-1 读缺失用例 2

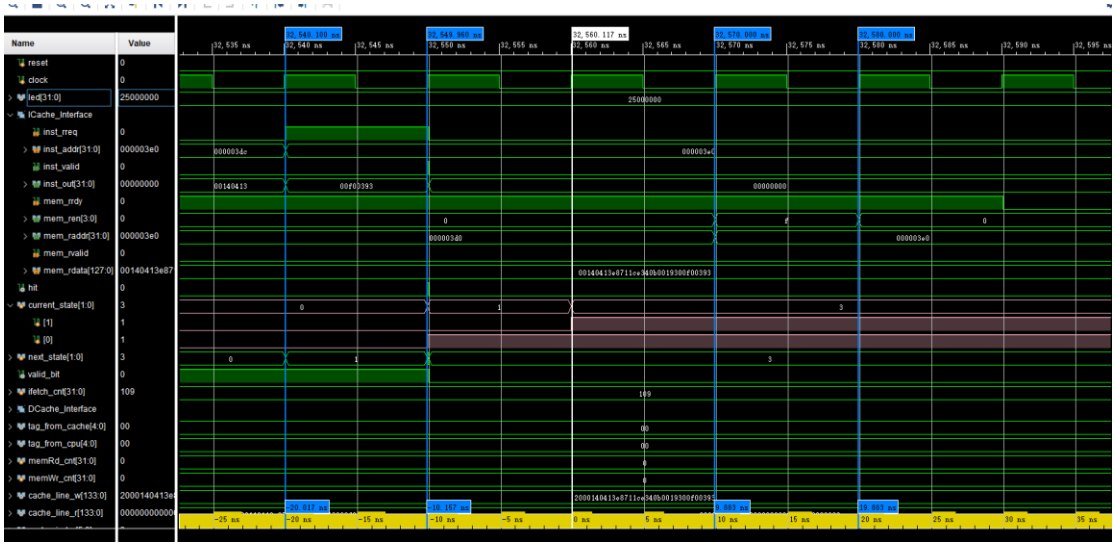

图 4-2 读缺失用例 2(前半部分)

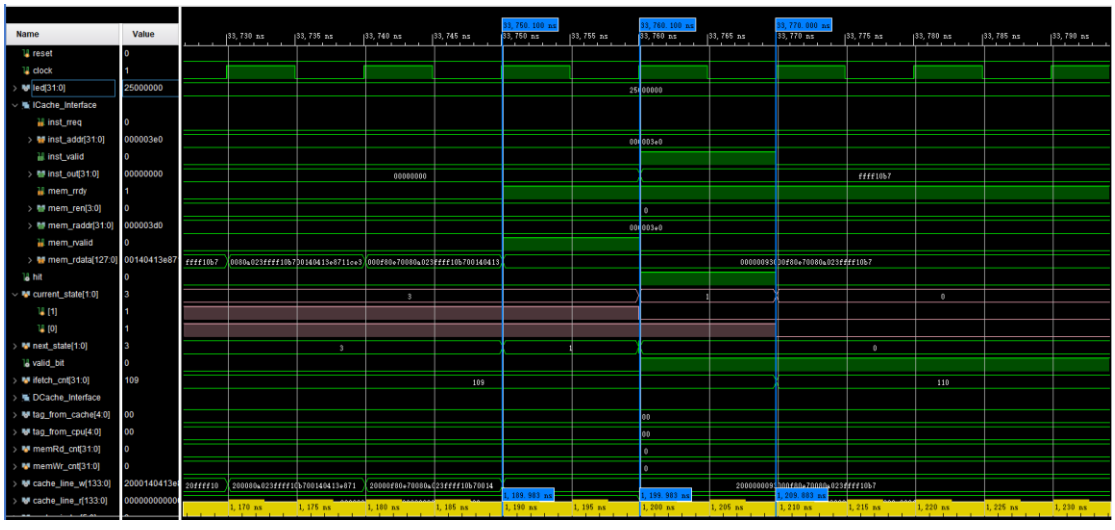

图 4-3 读缺失用例 2（后半部分）

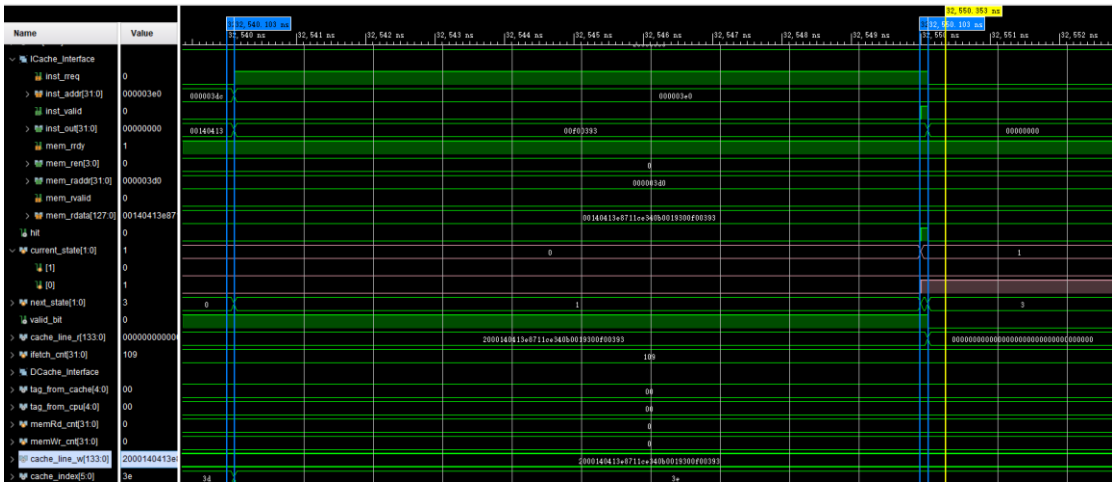

图 4-4 毛刺产生的原因

图 4-1 中从左向右标注了 8 个时钟上升沿。时序分析：

1) 在第 1 个时钟上升沿时，current_state=0,正在等待 cpu 请求。此时 inst_rreq 由 0 变成 1，并且 inst_addr 同时更新，说明 cpu 向 ICache 模块发送读取请求。
2) 在第 2 个时钟上升沿时，经过 1 个周期的信号输入，current_state 获取到了读取请求后，转入 TAG_CHECK 状态，从 0 更新为 1。此时对应地址从 ICache 存储体中读出了 Cache 行，由 cache 行中最高位引出的的 valid_bit=0，说明该 cache 块尚未有数据填充，本次读取不会命中。hit 信号为 0，符合预期。
3) 在第 3 个时钟上升沿时，由于上一个周期已经判断未命中，因此转向 REFILL 状态，current_state 更新为 3。
4) 在第 4 个时钟上升沿时，current_state=3,并且 mem_rrdy=1，因此 ICache 模块将向主存储器发出读取请求。此时 mem_ren 从 0 变成 f，mem_raddr 更新为当前 cpu 所访问的地址所在的块的地址，持续 1 个周期。之后主存储器收到请求后，进入读取操作，mem_rrdy 从 1 变成了 0。
5) 在第 6 个时钟上升沿时，此前 current_state 一直为 3，经过等待到该时刻。mem_rvalid 从 0 变成 1，同时 mem_rdata 更新为有效的数据值，并且写入 Cache 存储体中。
6) 在第 7 个时钟上升沿时，由于上一周期将数据写入了 Cache 存储体，该周期读出的 Cache 行更新，valid_bit 也更新为 1。同时状态转会 TAG_CHECK 状态，current_state 从 3 变成 1，hit 信号也更新为 1，说明读取到了目标数据，并可以向 cpu 输出。inst_valid 同步更新为 1，并且 inst_out 同步更新成正确的数据。
7) 在第 8 个时钟上升沿时，上一周期已经向 cpu 输出了数据，一次读取完成，current_state 从 1 回到 0，inst_valid 从 1 变 0，等待下一次读取请求。
8) 在 cpu 发送请求后，进行命中判断时，会产生一个 hit 信号的毛刺，如图 4-4 所示。因为当时钟上升沿到来时，Cache 存储体读取的输出信号 cache_line_r 不是在时钟上升沿的同时更新，而是比时钟上升沿延迟一小段，而状态机的状态与时钟上升沿同步更新。当时钟上升沿到来，current_state 从 0 变为 1，即进入 TAG_CHECK 状态后，用了还未更新的 cache 行数据通过组合逻辑进行命中判断，由于上一次读取

最后将数据存入了 cache 中，并且 tag 比较也为真，因此这个判断会得到高电平，因此 hit 在状态更新的同时更新成高电平。但短暂延迟后，cache 行读出的数据更新，就做出了正确的判断，hit 重新变成 0。这个毛刺现象对 ICache 模块的功能没有影响。

以上过程说明，ICache 能够正确地接受 cpu 的读取请求，判断是否命中，在缺失情况，到主存相应地址取回数据，并输出给 cpu。

3、思考与讨论

（1）分别给出无 ICache 时和有 ICache 时，SoC 运行测试程序的总时间的截图，并谈谈你对该测试结果的理解。

```
本次取指耗费的时钟数：1
———— test_24 ————
[615.420us] test_24 passed!
****************************************
* Congratulations! All tests passed!    *
****************************************
Total: 1967 instruction fetching, 26 memory access (18 RDs and 8 WRs)
ICache hit rate: 79.004%
$finish called at time : 616420 ns : File "C:/Users/Administrator/Desktop/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 179
run: Time (s): cpu = 00:00:05 ; elapsed = 00:00:13 . Memory (MB): peak = 933.988 ; gain = 0.129
```

图 5 有 ICache 时的运行时间

```
[1783.920us] test_19 passed!
[1880.320us] test_20 passed!
[2022.050us] test_21 passed!
[2178.540us] test_22 passed!
[2310.280us] test_23 passed!
[2396.080us] test_24 passed!
****************************************
* Congratulations! All tests passed!    *
****************************************
Total: 1967 instruction fetching, 26 memory access (18 RDs and 8 WRs)
$finish called at time : 2397080 ns : File "C:/Users/Administrator/Desktop/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 179
run: Time (s): cpu = 00:00:09 ; elapsed = 00:00:45 . Memory (MB): peak = 933.988 ; gain = 0.000
```

图 6 五 ICache 时的运行时间

有 ICache 时测试程序运行的总时间为 615.42us，无 ICache 时测试程序运行的总时间为 2396.08us。体现了有 Cache 时能显著缩小程序运行时间。在有 Cache 的时候，大部分的读取通过 Cache 命中，只需要 1 个时钟周期就完成。而无 Cache 的时候，每次存储访问都相当于 Cache 未命中的情形，因此消耗大量的时间。

（2）给出你的 ICache 命中率的截图，并尝试分析如何提高 ICache 命中率。

```
本次取指耗费的时钟数：1
———— test_24 ————
[615.420us] test_24 passed!
****************************************
* Congratulations! All tests passed!    *
****************************************
Total: 1967 instruction fetching, 26 memory access (18 RDs and 8 WRs)
ICache hit rate: 79.004%
$finish called at time : 616420 ns : File "C:/Users/Administrator/Desktop/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 179
run: Time (s): cpu = 00:00:05 ; elapsed = 00:00:13 . Memory (MB): peak = 933.988 ; gain = 0.129
```

图 7 Cache 命中率

Cache 的命中率为 79.004%。提高 ICache 命中率的方法：

1) 扩大 Cache 数据块的大小。原本一个 Cache 块存 128 位数据，扩大 Cache 块的块长，让一次读缺失的处理将更多的数据存入 Cache 中，减小接下来一段时间需要重新读取的可能性。
2) 改变 Cache 和主存的映射方式，直接映射改为组相联映射，可以提高命中率。
3) 增加 Cache 容量。

4、总结与反思

要求：总结完成本课程实验获得的收获，并给出合理的意见和建议。

通过这个课程实验，我完成了 RISC V 汇编语言的编程，用 Verilog 完成了除法器和乘法器的设计以及 Cache 的设计，对硬件是如何实现除法器的原理以及硬件中访问地址的细节有了更多的认识。

在 RISC V 汇编语言编写字符串匹配程序的过程中，所使用的思维是比较底层的，直接对寄存器及其内容进行构思。不论在高级语言中是什么类型的变量，在 RISC V 体系的硬件里面都是 32 位二进制数。但同时，我发现使用汇编语言编程和高级语言编程的思考方法没有本质差别，只是为了实现同一目标采取的方式有差别。

在除法器设计实验中，我深度思考了完成原码除法需要的位数，最后确定寄存器的位数是多少。使用 verilog 描述的过程中，我发现描述硬件的执行过程和理论课学习的执行过程在操作的定义和划分上有区别，人的思维把移位完成当作一轮，但是硬件中把移位和上商作为一个操作进行。通过乘法器设计实验，我也体会到了硬件上实现 booth 算法计算乘法的原理。同时在实验中积累了调试仿真波形的经验。

通过 Cache 设计实验，我加深了对寻址单位的理解。本来 RV 的寻址单位是 1 个字节，因此 cpu 的地址线最低位代表字节为单位。在 RISC V 中，涉及指令跳转的都是以 2 字节（半字）为单位跳转的，最理想是直接按指令长即 4 字节跳转，但考虑到压缩指令的存在，才按照 2 字节跳转。因此，B 型指令和 J 型指令储存到的是 Imm[12:1]，最低位肯定是 0，这就是按半字寻址。在实验 3 中做 ICache，指令长 32 位，寻址也是按照 32 位来寻址，而一个块是 4 个字，此时块内偏移确实是需要 2 位地址，但是应该把地址的[3:2]这两位接入偏移量，就是因为实际的地址中，按照 4 个字节位单位寻址，最低 2 位都是 0 是不会变化的，会变化的最低位是从 addr[2]开始。此外我明白了地址线在 Cache 中是如何使用的，对实现 Cache 的原理有了更深入的理解。

实验 2 和实验 3 我花了大部分时间 debug，最后发现基本都是我的信号和仿真测试的接口没对上，导致仿真跑不出任何波形，其实上我的计算功能本来是正确的，但输入输出信号时许没有严格按照要求做，所以测试跑错。