COA2021-programming11

Good luck and have fun!

1 实验要求

1.1 MMU

将 cache 与 TLB 融合到 MMU 中。

1.2 TLB

在 TLB 中添加你需要使用到的方法。

2 实验攻略

2.1 实验概述

本次实验为虚拟存储器的模拟。在上次作业中，大家已经实现了地址转换与数据加载的功能，本次作业主要的任务将 cache、TLB、memory、disk 融合成一个完整的存储器系统。

请认真阅读课本和 PPT，确保你已经理解存储器系统中的每个部件是如何产生联系的。接下来的工作将会把编程作业 7 到 11 统统联系起来，相信你完成之后将会非常有成就感！

2.2 代码导读

2.2.1 代码结构

```
.
|  .gitignore
|  2021.iml
|  pom.xml
|  README.md
|
└─src
    ├─main
    |  └─java
    |      ├─cpu
    |      |  ├─alu
    |      |  |
    |      |  ├─fpu
    |      |  |
    |      |  ├─mmu
    |      |  |      MMU.java   // need to write
    |      |  |
    |      |  └─nbcdu
    |      |
    |      ├─memory
    |      |  |  Memory.java    // need to write
    |      |  |
    |      |  ├─cache
    |      |  |
    |      |  ├─disk
    |      |  |
    |      |  └─tlb
    |      |          TLB.java  // need to write
    |      |
    |      └─util
    |
    └─test
        └─java
            └─memory
                    CacheTest.java
                    EasterEgg.java
                    MemTestHelper.java
                    PSTest.java
                    TLBTest.java
```

2.2.2 TLB 的模拟

我们模拟了一个大小为 256 行、使用全相联映射、FIFO 替换策略的 TLB。TLB 类的源码比较简单，相关注释也非常详细，大家可以自行阅读。

需要特别提醒的是，TLB 只是页表的缓存，因此 TLB 行号并不等于虚页号。

2.2.3 MMU 的改动

在上次作业中，我们默认 cache 与 TLB 都是关闭的，MMU 的所有访存操作都是直接对主存进行操作。

而本次作业中，我们在 MMU 中加入了对全局变量 `Cache.isAvailable` 和 `TLB.isAvailable` 的判断。这两个全局变量分别表示是否启用 cache 与 TLB。我们提供的代码（即上次作业的框架代码）里面已经处理好 cache 和 TLB 都关闭的情况，至于 cache 和 TLB 都开启的情况就需要大家动手去实现啦！

2.3 实现指导

2.3.1 cache 的融合

第一步，你需要将 cache 融合进 MMU 中。

这一步相对简单。需要注意，由于 cache 是 memory 的缓存，所以任何涉及到访问主存数据的地方都要添加对 cache 的调用。

2.3.2 TLB 的融合

第二步，你需要将 TLB 融合进 MMU 中。

这一步相对麻烦。即使填写 TLB 的方法我们已经帮助大家实现好了，但从 TLB 中读取数据的方法仍然需要大家自行设计并进行调用。由于 TLB 是页表的缓存，所以任何涉及到访问页表的地方都要添加对 TLB 的调用。下面作一点提示。

在 addressTranslation 方法中判断是否缺页并进行加载页的时候，原代码如下：

```
if (!memory.isValidPage(i)) {
  // 缺页中断，该页不在内存中，内存从磁盘加载该页的数据
  memory.page_load(i);
}
```

在启用 TLB 之后，需要改动的地方有：

1. 启用 TLB 之后，判断是否缺页的工作应该首先交给 TLB 来完成，这时候就需要用到你自己设计的 TLB 的方法。
2. 如果发生缺页，page_load 方法会进行填页表，填页表之后不要忘记填 TLB。

3. 如果发生了虚页不在 TLB 但在页表中的情况，即 TLB 缺失但页表命中的情况，这个时候需要把页表中的该虚页项填到 TLB 中，我们有相关的测试用例来专门测试这种特殊情况。

同时，在上次作业中你自己实现的 toPagePhysicalAddr 方法里面，从页表中取物理页框号的时候，也应该改成从 TLB 中取物理页框号，这个时候也需要调用你自己设计的 TLB 的方法。参考结构如下

```
if (TLB.isAvailable) {
访问 TLB 取物理页框号;
} else {
访问页表取物理页框号;
}
```

2.4 总结

所有需要大家完成的部分都已经用 TODO 标出。为了减轻大家的负担，我们归纳了本次作业中你需要完成的小任务以及步骤

1. 将上次作业的代码复制过来，然后你应该可以通过 PSTest（关闭 cache 和 TLB 的情况）的 4 个用例。（粘贴代码的时候注意不要整个文件粘贴过来，否则会破坏框架代码的改动）
2. 正确实现好 cache 的融合，然后你应该可以通过 CacheTest（开启 cache 关闭 TLB 的情况）的 1 个用例。
3. 正确实现好 TLB 的融合，然后你应该可以通过剩余 5 个用例（开启 cache 和 TLB 的情况）。

至此，你已经完成了全部工作(・ω・)ノ

2.5 彩蛋

作为存储模块的最终章，我们设计了一个彩蛋供大家体会各个内存部件之间的关系。

细心观察的同学可以发现，我们在 Memory 类中新增了一个 timer 字段，在这个字段开启设为 true 后，所有访问内存以及访问页表的操作都将产生 10 毫秒的延时，而访问 cache 与 TLB 的操作则不会产生延时。由于在真实的计算机系统中，访问 cache 和 TLB 的速度要比访问主存的速度快得多。因此，我们设计了这个 10 毫秒的延时，来模拟这个速度上的差异。

在我们提供的测试用例中有一个 EasterEgg 类，这个类将不断的访问同一段数据，来模拟计算机的时间局部性原理。在 EasterEgg 类中的 init 方法里面有如下代码：

```
Memory.timer = true;

Cache.isAvailable = true;
TLB.isAvailable = true;
```

大家可以修改两个 isAvailable 字段，自主设置 cache 和 TLB 是否启用，来观察他们速度上的差异。以下是我自己的运行结果，仅供参考，我的处理器型号是 Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz。

运行时间	启用 cache
启用 TLB	约 100ms
不启用 TLB	约 9.5s

大家可以尽情修改彩蛋，比如修改延时的时间、修改 cache 和 tlb 是否启用等。也可以在彩蛋里面单步进入，看看从 mmu 开始是怎么一步步访问 disk、memory、TLB 和 cache 的。

最后，留给大家一个问题，为什么在我们的模拟下，单独关闭 cache 和单独关闭 TLB 的运行时间差那么远？

(・ω・)ノ