# lab2 Cache 模拟实验

PB15000195 邢宇

## 1. 实验目的

- 加深对 Cache 的基本概念、基本组织结构以及基本工作原理的理解；
- 掌握 Cache 容量、相联度、块大小对 Cache 性能的影响；
- 掌握降低 Cache 不命中率的各种方法以及这些方法对提高 Cache 性能的好处；
- 理解 LRU 与随机法的基本思想以及它们对 Cache 性能的影响。

## 2. 实验要求

设计与实现一个 Cache 模拟器，能模拟处理器中 Cache 的行为。处理器访存有三种类型：读指令、读数据和写数据，给出访存的地址和类型，我们的 Cache 的模拟器能够进行模拟这种带有 Cache 的访存行为，并能给出统计信息，如**访存次数、Cache 命中次数、命中率**等。

基本要求：模拟器中必须具备下列配置项

a. 能够设置 Cache 总的大小
b. 能够设置 Cache 块的大小
c. 能够设置 Cache 的映射机制：直接映射、n-路组相联
d. 能够设置 Cache 的替换策略：LRU、FIFO，随机
e. 能够设置 Cache 的写策略：写回法、写直达法

较高要求：模拟器中可以选择支持下列配置

f. 能够设置将 Cache 分为数据 Cache 和 指令 Cache
g. 能够设置预取策略
h. 能够设置写不命中的调块策略
i. 有友好的操作界面，如使用界面来配置 Cache

## 3. 实验设计

将整个执行过程分为下列流程

读取文件到缓冲区 —> 每次读取一条字符串作指令并获取 type 和 address —> 解码条指令，获得块号，索引号，偏移地址等信息 —> 建立这条地址对应组的引用 —> 根据选择的算法（是 LRU/FIFO/random）来维护 tag、valid 等基本标记域 —> 根据命中的结果更新相应的统计量 —> 刷新显示

设计思想：

对于 Cache 的每个组都需要维护自己的优先队列，valid 域和 tag 域，每次获得访问地址后先定位是哪个组，建立其各个域的引用，之后遍历组内的块查找是否命中即可。

用到的成员和方法：

- GUI 所需的 label 和 button
- valid，tag 域，dirty 域（未用到）

- 各种统计量如总缺失数，读取缺失数
- 对地址分析后的得到的各个量比如块号，索引号
- 对各个域建立具体组的引用的变量，如 gvalid，gtag

关键方法：

LRU：

维护一个队列 gRU，这个队列代表 address 对应组所维护的优先队列。

当队列未满时如果有块未命中需调入，则首插设置 valid；若未命中但队列已满，删除队尾元素后插入队首，修改队尾元素作为索引对应的 tag 域表示调入。

若命中了队列中的某一个块，将此命中的块移到队首，修改以其作为索引对应的 tag

FIFO：

维护一个队列 gRU ,这个队列代表 address 对应的组维护的优先队列

当队列未满时如果有块未命中需调入，则首插设置 valid，若需调入但队列已满，删除队尾元素插入队首；修改队尾元素作为索引对应的 tag 域表示调入。

若命中了队列中的某一个块，什么也不做。

random：

若未满则首插，若已满，生成一个随机数，将其对应的 tag 域进行修改

高级功能实现策略：

**复位**：

添加相应事件，复位按钮被按下时清空所有全局索引，全局标记变量和各个统计量，刷新显示

**预取**：

含义：当读数据不命中时取读的块及其下一个块，并将下一个块也作为访存一次，也计入缺失次数。

实现：设置 boolean 表示是否处于预取下一条的状态，防止继续读取文件指令。如果符合了预取的情境：读访问且不命中且不属于预取状态，置预取状态；在预取状态下直接取上一个地址+块大小对应的地址，刷新显示

**写不分配**：

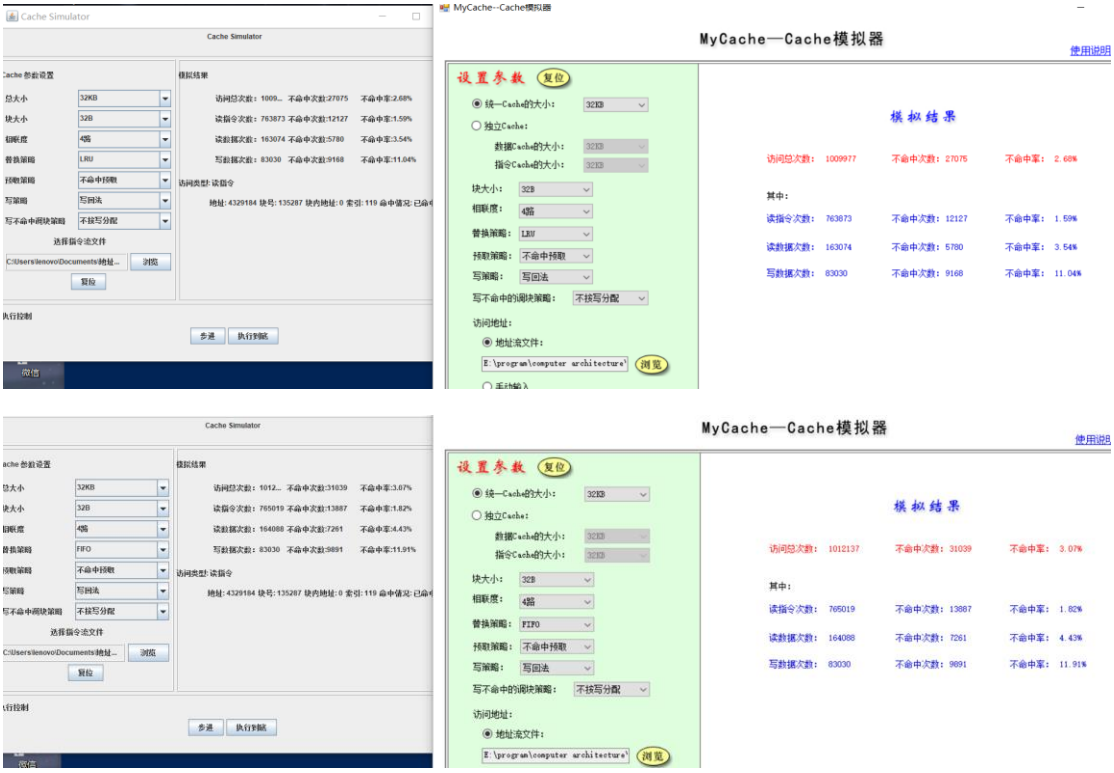
含义：当写不命中时不写 Cache 而是写更低级的缓存。意味着只计算缺失不更新 tag

实现：符合写不命中的情境时计算缺失后直接跳出

实验结果分析

与给定的标准 Cache 数据完全一致（包括块号，索引号等实时统计量）

典型配置下的结果比对：




## 4. 实验感悟

第一次写 Java 觉得还是挺好用的，就是容器的使用和变量的初始化比较繁琐，没有 C++那么间接

由于只需要写一级缓存所以写策略完全不影响，所以 dirty 域在本实验中也没用到。