计算机系统Lab3实验报告

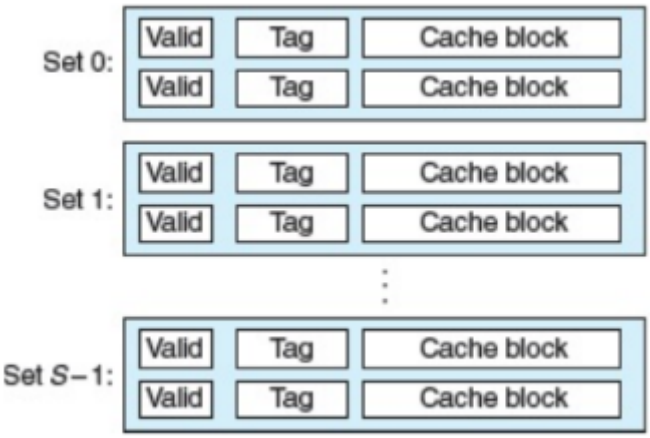
22级JOHN班 涂文良

基于代码展示的实现思路简述

在本次 lab 中，我们需要实现一个 cache 的模拟器。由于我对于 C++ 更为熟悉，所以我采取了 C++ 进行编程，代码文件在 csim.cpp 中。经过测试，我实现的模拟器行为与 csim-ref 一致。

根据实验要求，我们需要设计的 cache 模拟器，只需要在接受输入文件后，输出 miss / hit / eviction 的相关信息。 所以，实际上我们不需要真正实现 block 的部分。我只需要存储 tag 和 valid bit 就行了，不必真正把信息存入 block。另外实验要求中也说明了不必考虑 size 字段，那就更无需考虑 block 了。

如图所示， 我们实现的 cache 实际上是一个 set-associative cache：



每一个 set 中，有 E 个 cacheline。因此，我如下设计 cacheline 与 cache:

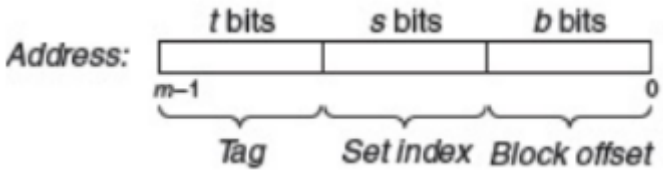
```
typedef struct
{
    int valid;
    size_t tag;
} CacheLine;

typedef struct
{
    int S, E, B;
    std::vector<std::vector<CacheLine>> line;
} Cache;
```

在这里，我的 cacheline 中的 valid bit 不再只是 0 或 1， 而是为了方便实现 LRU 替换策略， 存储了最后一次访问这个 cacheline 的时间。

在模拟阶段。首先我们对命令行参数做分析，初始化 cache， 然后打开源文件接受输入，接着进行最关键的分析 miss / hit环节。

这部分我的实现思路是， 首先根据 address 算出 对应的 set 编号和 tag， 计算方法如图：



实现如下(简单的位运算技巧)：

```
size_t tag = address >> (s + b);
size_t set_index = (address >> b) & ~(~0u << s);
```

接着，我遍历这个 set 里的所有 cacheline， 检查 tag 是否相等。如果相等那就说明 hit。如果没有 hit， 那么出现了 miss。 这时，我们试图寻找到一个 non-valid 的 cacheline。如果没有找到，那么就要替换了。我们采用LRU替换策略， 遍历一遍这个 set， 找到访问时间最小的那个 cacheline, 把它替换掉。这部分实现如下：

```
int min_num = INT32_MAX;
int evit_num = -1;
for(int i = 0; i < E; i++)
{
    if(cache.line[set_index][i].valid < min_num)
    {
        min_num = cache.line[set_index][i].valid;
        evit_num = i;
    }
}
result += "eviction ";
eviction_cnt++;
cache.line[set_index][evit_num].valid = time_spent;
cache.line[set_index][evit_num].tag = tag;
```

另外，如果 operation 是 M， 那么要访问两次， 也就是调用两次这个函数。

用以上思路，我实现了这个 cache 模拟器，并通过了测试。