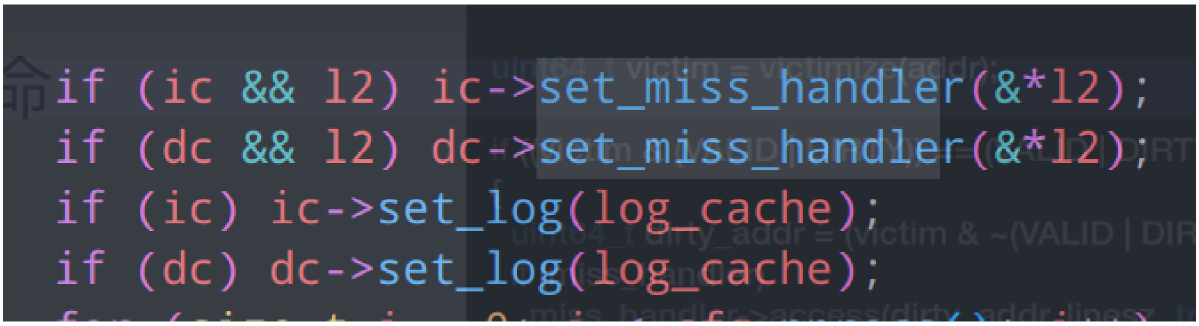
spike 的cache 为什么会出现 l2 miss的次数高于l1 ,按理来说只有l1 miss才有机会去读l2



```
if (ic && l2) ic->set_miss_handler(&*l2);
if (dc && l2) dc->set_miss_handler(&*l2);
if (ic) ic->set_log(log_cache);
if (dc) dc->set_log(log_cache);
```

```
void cache_sim_t::access(uint64_t addr, size_t bytes, bool store)
{
  // access次数的统计
  store ? write_accesses++ : read_accesses++;
  (store ? bytes_written : bytes_read) += bytes;

  // 检查是否命中
  uint64_t* hit_way = check_tag(addr);
  if (likely(hit_way != NULL))
  {
    if (store)
      *hit_way |= DIRTY;
    return;
  }

  // 未命中
  store ? write_misses++ : read_misses++;
  if (log)
  {
    std::cerr << name << " "
              << (store ? "write" : "read") << " miss 0x"
              << std::hex << addr << std::endl;
  }

  // victim 是需要进行替换的tag ,为0说明cache没有满
  uint64_t victim = victimize(addr);


  // dirty && valid
  if ((victim & (VALID | DIRTY)) == (VALID | DIRTY))
  {

    // 把替换的block写入l2
    uint64_t dirty_addr = (victim & ~(VALID | DIRTY)) << idx_shift;
    // 如果有l2的话
    if (miss_handler)
      miss_handler->access(dirty_addr, linesz, true);
    writebacks++;
  }

  // read
  if (miss_handler)
```

```
    miss_handler->access(addr & ~(linesz-1), linesz, false);

  if (store)
    *check_tag(addr) |= DIRTY;
}


// 这种存放的方式应该类似于哈希表,是有键值对的
// idx_shift应该和block的大小有关,即前面几位是一个block
uint64_t fa_cache_sim_t::victimize(uint64_t addr)
{
  uint64_t old_tag = 0;
  if (tags.size() == ways)
  {
    auto it = tags.begin();
    std::advance(it, lfsr.next() % ways);
    old_tag = it->second;
    tags.erase(it);
  }
  tags[addr >> idx_shift] = (addr >> idx_shift) | VALID;
  return old_tag;
}
```

D$ read miss + D$ write miss + I$ write miss = L2 $ read access

L2 $write access = D$ write miss 中 dirty 的一部分

# 附录

## 参考文献

TLCT-Open-Reports

## 版权信息