گزارش تمرین عملی ۸ درس معماری کامپیوتر

متین باقری (۴۰۲۱۰۵۷۲۷)

محمد نوید آتشین بار (۴۰۲۱۰۵۵۸۱)

دکتر اسدی

در این تمرین باید با استفاده از شبیه ساز ChampSim سیاست های مختلف جایگزینی را برای cache و تعداد way مختلف تست کنیم و hit rate را در حالت بدست آورده و نتایج را با یکدیگر مقایسه کنیم.
باید سه سیاست lru، mru و random را تست میکردیم که سیاست های lru و random از قبل در این شبیه ساز وجود داشت اما سیاست mru را با استفاده از دو سیاست موجود نیز اضافه کردیم.
اضافه کردن mru.h :

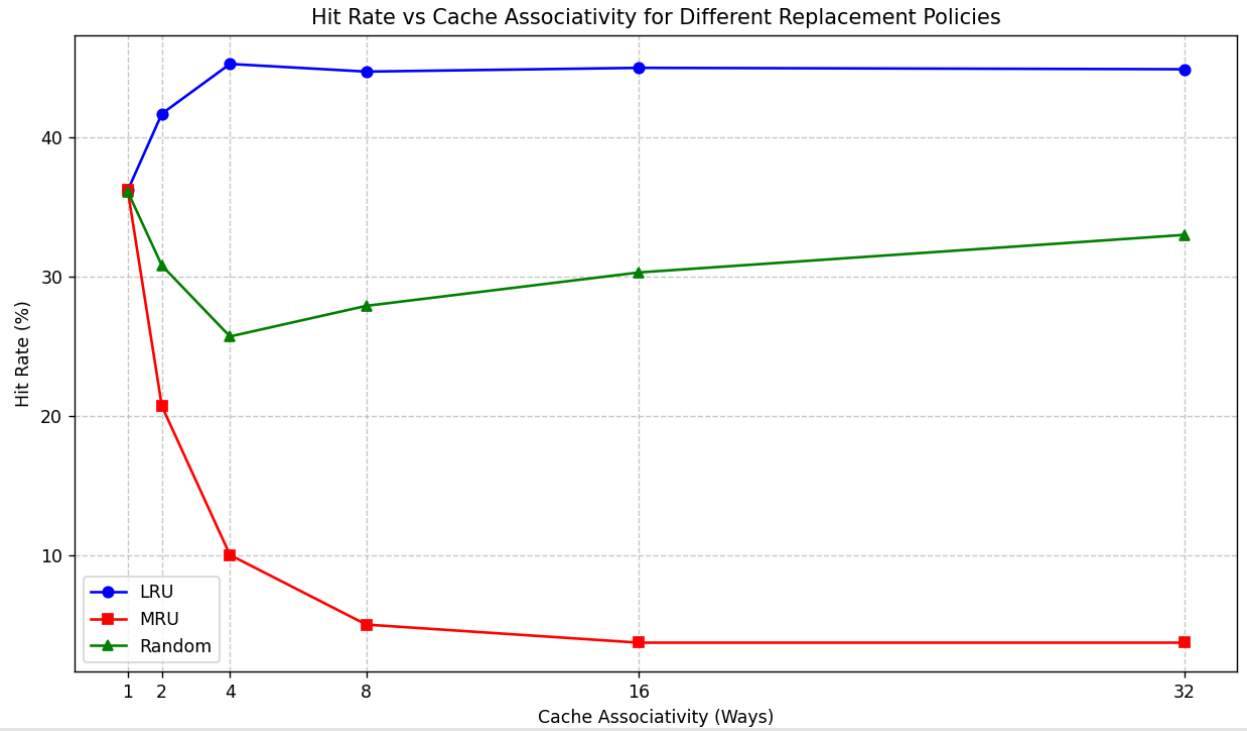
```
#ifndef REPLACEMENT_LRU_H
#define REPLACEMENT_LRU_H

#include <vector>

#include "cache.h"
#include "modules.h"

class mru : public champsim::modules::replacement
{
  long NUM_WAY;
  std::vector<uint64_t> last_used_cycles;
  uint64_t cycle = 0;

public:
  explicit mru(CACHE* cache);
  mru(CACHE* cache, long sets, long ways);

  // void initialize_replacement();
  long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
champsim::cache_block* current_set, champsim::address ip,
                   champsim::address full_addr, access_type type);
  void replacement_cache_fill(uint32_t triggering_cpu, long set, long way,
champsim::address full_addr, champsim::address ip, champsim::address victim_addr,
access_type type);
  void update_replacement_state(uint32_t triggering_cpu, long set, long way,
champsim::address full_addr, champsim::address ip, champsim::address victim_addr,
                                access_type type, uint8_t hit);
  // void replacement_final_stats()
};

#endif
```

اضافه کردن فایل mru.cc :

```
#include "mru.h"
#include <algorithm>
#include <cassert>

mru::mru(CACHE* cache) : mru(cache, cache->NUM_SET, cache->NUM_WAY) {}

mru::mru(CACHE* cache, long sets, long ways) : replacement(cache), NUM_WAY(ways),
last_used_cycles(static_cast<std::size_t>(sets * ways), 0) {}

long mru::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
champsim::cache_block* current_set, champsim::address ip,
                      champsim::address full_addr, access_type type)
{
  auto begin = std::next(std::begin(last_used_cycles), set * NUM_WAY);
  auto end = std::next(begin, NUM_WAY);


  auto victim = std::max_element(begin, end);
  assert(begin <= victim);
  assert(victim < end);
  return std::distance(begin, victim);
}

void mru::replacement_cache_fill(uint32_t triggering_cpu, long set, long way,
champsim::address full_addr, champsim::address ip, champsim::address victim_addr,
                                 access_type type)
{
  // Mark the way as being used on the current cycle
  last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}

void mru::update_replacement_state(uint32_t triggering_cpu, long set, long way,
champsim::address full_addr, champsim::address ip,
                                   champsim::address victim_addr, access_type type,
uint8_t hit)
{
  // Mark the way as being used on the current cycle
  if (hit && access_type{type} != access_type::WRITE) // Skip this for writeback hits
    last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}
```

سپس با استفاده از شبیه ساز مقدار hit rate را برای ۱۸ حالت زیر را بدست آوردیم بطوریکه زمان شبیه سازی حدود ۴ دقیقه طول بکشه و نسبت warmup instructions به simulation instructions ۱ به ۴ باشد (دستورات اجرا شده و خروجی هر حالت در فایل practical ضمیمه شده است) :

policy \ way	1way	2way	4way	8way	16way	32way
lru	36.2%	41.7%	45.28%	44.73%	45%	44.9%
mru	36.2%	20.7%	10%	5%	3.7%	3.7%
random	36.1%	30.8%	25.7%	27.9%	30.3%	33%



طبق دیتاهای بدست آمده میبینیم که سیاست LRU در حالت 4 way بیشترین نرخ برخورد را فراهم کرده است. زیرا در این حالت هر set دارای ۴ خط داده است و وقتی چنده داده به یک set نگاشت میشوند هنوز فضای کافی برای ذخیره آنها وجود دارد و برای locality نیز LRU عالی عمل میکند و نرخ برخورد را بالا میبرد اما در مقابل MRU اینجور نیست و داده هایی که به تازگی استفاده شده است رو زودتر حذف میکند. Random نیز چون شانسی یک داده رو حذف میکند با 32 way نیز شانس بیشتری ایجاد میکند که داده های مهم را حذف نکند و عملکردش متوسط است. در نتیجه بهترین ترکیب سیاست جایگزینی LRU با 4-way associativity است با نرخ برخورد ۴۵٫۲۸٪.