实验报告

cache block存储空间的大小为64B
cache存储空间的大小为64KB
8-way set associative　8路组相连映射（一组8个）

1024 = 2^10 blocks
2^7= 128 组 （每组2^3 = 8 块） * 每块2^6 = 64 Byte

标志位只需要valid bit即可

替换算法采用随机方式　cache满之后随机选一行换掉

write through 全写法　写cache同时写主存
not write allocate 非写分配法　写缺失不导致cache装载

cache.c:

```
#include "memory/mmu/cache.h"
#include "memory/memory.h"

typedef struct {
    bool valid;
    uint32_t tag;
    uint8_t data[64];   // Byte 要用uint_8t
}CacheLine;
static CacheLine cache[128][8];

// init the cache
void init_cache()
{
    for(int i=0; i<128; i++){
        for(int j=0; j<8; j++){
            cache[i][j].valid = false;
        }
    }
}

// read data from cache
uint32_t cache_read(paddr_t paddr, size_t len)
{
    uint32_t set = (paddr / 64) % 128;          //第几组
    uint32_t tag = paddr / (64 * 128);          //标志位
    uint32_t begin_addr = (paddr / 64) * 64;    //所在行的起始地址
    uint32_t offset = paddr % 64;               //在该行的偏移量

    uint32_t ret = 0;

    if (offset + len > 64)  //跨行就一个一个找
    {
        for (int i = 0; i < len; ++i)
            ((uint8_t*)&ret)[i] = (uint8_t)cache_read(paddr + i, 1);
        return ret;
    }
```

```
    for (int i = 0; i < 8; ++i){
        if (cache[set][i].valid && cache[set][i].tag == tag) //命中 HIT
        {
            memcpy(&ret, cache[set][i].data + offset, len);
            return ret;
        }
    }

     //没有命中 MISS
    int line = -1;      //本组第几行
    for (int i = 0; i < 8; ++i) //找空位
    {
        if (!cache[set][i].valid) {
            line = i; break;
        }
    }
    if(line == -1) line = tag % 8;    // 没有空位 随机选一行

    //填入
    cache[set][line].valid = true;
    cache[set][line].tag = tag;
    memcpy(cache[set][line].data, hw_mem + begin_addr, 64);

    memcpy(&ret, cache[set][line].data + offset, len);
    return ret;
}


// write data to cache
void cache_write(paddr_t paddr, size_t len, uint32_t data)
{
    int set = (paddr / 64) % 128;
    int tag = paddr / (64 * 128);
    //int begin_addr = paddr / 64 * 64;
    int offset = paddr % 64;

    if(offset + len > 64){
        for(int i=0; i<len; i++){
            cache_write(paddr+i, 1, (uint32_t)(((uint8_t*)&data)[i]));
        }
        return;
    }
    for(int i=0; i<8; i++){
        if(cache[set][i].valid && cache[set][i].tag == tag) //HIT
        {
            memcpy(cache[set][i].data + offset, &data, len);
            hw_mem_write(paddr, len, data);
            return;
        }
    }
    hw_mem_write(paddr, len, data);  //MISS
}
```