

Lab3 报告

- 姓名：邵震哲
 - 班级：1620204
 - 学号：162020130
 - 报告阶段：lab3
 - 完成日期：2022.6.12
 - 本次实验，我完成了所有内容。
-

目录

Lab3 报告

目录

init_cache (20分)

cache_read (30分)

cache_write (30分)

最终结果截图 (20分)

备注

init_cache (20分)

- cache结构设计

- 思路

cache有16KB(2^{14})，块大小有 2^6 bit，所以cache行号有 2^8 行，按四路组相联来处理，那么组数是 2^6 ，

主存大小有32KB (2^{15})，tag = 15 - 块内偏移长度 (6) - cache组号 (6) = 3

- 代码

```
typedef struct
{
    bool dirty:1;      //脏位
    bool data[64];     //数据位
    uint32_t tag:3;    //标记
    bool valid:1;      //有效位
}Cache;

Cache *cache;
```

- cache初始化

- 思路

遍历所有的cache行，将每个cache行的有效位和脏位置0

- 代码

```

void init_cache(int total_size_width, int associativity_width) {
    int i;
    int cache_line = exp2(total_size_width - associativity_width);
    cache = malloc(cache_line * sizeof(Cache));
    for(i = 0 ; i < cache_line ; i++)
    {
        cache[i].valid = 0;
        cache[i].dirty = 0;
    }
}

```

cache_read (30分)

- 思路

先按主存地址访问cache，如果命中，返回数据

如果不命中：

在cache中找空闲行，即判断valid是否为0，如果为0，则将此cache行作为新的读入的数据存放的地方

如果该组所有cache的valid为1，那么随便取一行替换，替换时需要判断脏位是否为1，为1的话就将当前的cache行先写回主存，再则将此cache行作为新的读入的数据存放的地方

- 代码

```

uint32_t cache_read(uintptr_t addr) {
    try_increase(1);    //访问cache次数加一
    uint32_t data;      //读取的数据
    addr = addr & 0x7FFF;    //主存地址划分
    uint32_t mem_blocks_num, mem_tag, group_id, block_offset;
    block_offset = addr & 0x3C;
    group_id = (addr >> 6) & 0x3F;
    mem_tag = (addr >> 12) & 0x7;
    mem_blocks_num = (addr >> 6) & 0x1FF;

    uint32_t paddr;
    int i, j, rand_line;
    int group_start = group_id * 4; //在当前cache组中查找
    for(i = group_start ; i < group_start + 4 ; i++)
    {
        if(cache[i].tag == mem_tag && cache[i].valid == 1)
        {
            hit_increase(1);
            break;
        }
    }
    if(i < group_start + 4) //如果命中
    {
        data = cache[i].data[block_offset] + (cache[i].data[block_offset +
1] << 8) + (cache[i].data[block_offset + 2] << 16) +
(cache[i].data[block_offset + 3] << 24); //将该数据返回
    }
    else //如果不命中，则分为寻找空闲行和随机替换两种情况
    {
        for(j = group_start ; j < group_start + 4 ; j++) //寻找空闲行

```

```

{
    if(cache[j].valid == 0)
        break;
}
if(j < group_start + 4 )    //找到空闲行
{
    mem_read(mem_blocks_num , cache[j].data); //先将数据读入
    cache[j].tag = mem_tag;
    cache[j].valid = 1;
    data = cache[j].data[block_offset] + (cache[j].data[block_offset
+ 1] << 8) + (cache[j].data[block_offset + 2] << 16) +
(cache[j].data[block_offset + 3] << 24);
}
else    //该组cache全满，考虑随机替换
{
    rand_line = rand()%4 + group_start; //mod4找到一个0~3的任意数
    if(cache[rand_line].dirty == 1)    //脏位为1先写回
    {
        paddr = ( cache[rand_line].tag << 6 ) | group_id ;
        mem_write(paddr , cache[rand_line].data);
    }
    mem_read(mem_blocks_num , cache[rand_line].data);    //读入主存块
    cache[rand_line].tag = mem_tag;
    cache[rand_line].valid = 1;
    data = cache[rand_line].data[block_offset] +
(cache[rand_line].data[block_offset + 1] << 8) +
(cache[rand_line].data[block_offset + 2] << 16) +
(cache[rand_line].data[block_offset + 3] << 24);
}
}
return data;
}

```

cache_write (30分)

- 思路

先按主存地址访问cache，如果命中，则将掩码对应的偏移处的值修改

若不命中：

在cache中找空闲行，即判断valid是否为0，如果为0，则将此cache行作为新的读入数据的地方，并将掩码对应的偏移处的值修改

如果该cache组所有的valid_bit为1，那么随便取一行，此时先判断脏位是否为1，为1的话就将当前的cache行先写回主存，再则将此cache行作为新的读入的数据存放的地方，并将掩码对应的偏移处的值修改

- 代码

```

void cache_write(uintptr_t addr, uint32_t data, uint32_t wmask) {
    try_increase(1);    //访问cache次数加一
    addr = addr & 0x7FFF;    //划分主存地址
    uint32_t mem_blocks_num , mem_tag , group_id , block_offset;
    uint32_t * ptr;
    block_offset = addr & 0x3C;
    group_id = (addr >> 6) & 0x3F;

```

```

mem_tag = (addr >> 12) & 0x7;
mem_blocks_num = (addr >> 6) & 0x1FF;

uint32_t paddr;

int i , j , rand_line;
int group_start = group_id * 4; //在当前cache组中查找
for(i = group_start ; i < group_start + 4; i++)
{
    if(cache[i].tag == mem_tag && cache[i].valid == 1)
    {
        hit_increase(1);
        break;
    }
}
if(i < group_start + 4) //如果写命中
{
    ptr = (uint32_t *)&cache[i].data[block_offset]; //修改cache行的对应块内
偏移的数据
    *ptr = (*ptr & ~wmask) | (data & wmask);
    cache[i].dirty = 1;
}
else //如果写不命中
{
    for(j = group_start ; j < group_start + 4 ; j++) //找空闲行
    {
        if(cache[j].valid == 0)
            break;
    }
    if(j < group_start + 4) //找到空闲行
    {
        mem_read(mem_blocks_num , cache[j].data); //读入主存块

        ptr = (uint32_t *)&cache[j].data[block_offset]; //修改cache行的对应
块内偏移的数据
        *ptr = (*ptr & ~wmask) | (data & wmask);

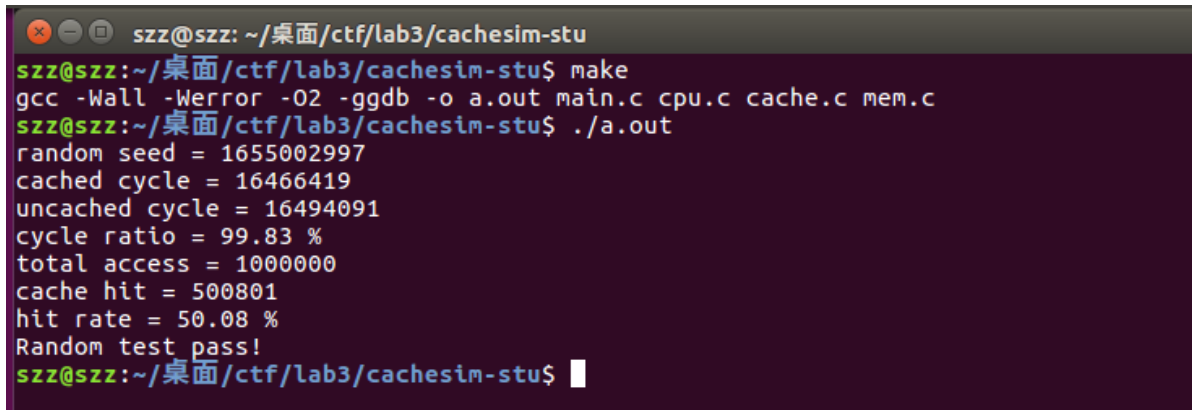
        cache[j].tag = mem_tag; //设置标志位
        cache[j].valid = 1;
        cache[j].dirty = 1;
    }
    else //该组cache全满，考虑随机替换
    {
        rand_line = rand()%4 + group_start;
        if(cache[rand_line].dirty == 1) //脏位为1先写回主存
        {
            paddr = (cache[rand_line].tag << 6 ) | group_id; //拼接写回主存
的主存块号
            mem_write(paddr, cache[rand_line].data); //写回对应的主存块
        }
        mem_read(mem_blocks_num , cache[rand_line].data); //读入主存
块

        ptr = (uint32_t *)&cache[rand_line].data[block_offset]; //修改
cache行的对应块内偏移的数据
        *ptr = (*ptr & ~wmask) | (data & wmask);
    }
}

```

```
        cache[rand_line].tag = mem_tag; //设置标志位
        cache[rand_line].valid = 1;
        cache[rand_line].dirty = 1;
    }
}
```

最终结果截图（20分）



```
szz@szz: ~/桌面/ctf/lab3/cachesim-stu
szz@szz:~/桌面/ctf/lab3/cachesim-stu$ make
gcc -Wall -Werror -O2 -ggdb -o a.out main.c cpu.c cache.c mem.c
szz@szz:~/桌面/ctf/lab3/cachesim-stu$ ./a.out
random seed = 1655002997
cached cycle = 16466419
uncached cycle = 16494091
cycle ratio = 99.83 %
total access = 1000000
cache hit = 500801
hit rate = 50.08 %
Random test pass!
szz@szz:~/桌面/ctf/lab3/cachesim-stu$
```

备注

助教真帅