

一、原型一

```
void updateLruNumber(Sim_Cache *sim_cache,int setBits,int hitIndex) {
    sim_cache->sets[setBits].lines[hitIndex].LruNumber=MAX_NUM;
    int i;
    for (i=0;i<sim_cache->line_num;i++) {
        if (i!=hitIndex)
            sim_cache->sets[setBits].lines[i].LruNumber-=1;
    }
    return;
}
```

更新 LruNumber 就按照 LRU 的策略, 指定块 setBits 的对应的下标为 hitIndex 的行 Number 赋值为最大值, 这里设置最大值为 1, 同组内其余行的就减 1

```
int findMinLruNumber(Sim_Cache *sim_cache,int setBits) {
    int i;
    int evi_Index,MinLruNumber;
    evi_Index=0;
    MinLruNumber=sim_cache->sets[setBits].lines[0].LruNumber;
    for (i=0;i<sim_cache->line_num;i++) {
        if (MinLruNumber > sim_cache->sets[setBits].lines[i].LruNumber) {
            evi_Index=i;
            MinLruNumber=sim_cache->sets[setBits].lines[i].LruNumber;
        }
    }
    return evi_Index;
}
```

找到 LruNumber 的最小值以实现之后对它的 eviction, 就是对该块 setBits 进行循环, 找到 LruNumber 最小的行, 返回该行的下标即可。

```
int isMiss(Sim_Cache *sim_cache,int setBits,int tagBits) {
    int i;
    int Miss=1;
    for (i=0;i<sim_cache->line_num;i++) {
        if (sim_cache->sets[setBits].lines[i].vaild == 1 && sim_cache->sets[setBits].lines[i].tag == tagBits) {
            Miss=0;
            updateLruNumber(sim_cache,setBits,i);
            break;
        }
    }
    return Miss;
}
```

这里判断是否 miss, miss 了就返回 1, 在对这一个块进行循环遍历的时候, 来判断有没有命中, 命中有两个条件: 有效位 vaild 为 1, 标记位 tag 和 tagBits 相等。命中了 miss 就为 0 并且要更新 LruNumber, 此时返回即可。

二、原型二

```
int updateCache(Sim_Cache *sim_cache,int setBits,int tagBits) {
    int full;
    int i;
    full=1;
    for (i=0;i<sim_cache->line_num;i++) {
        if (sim_cache->sets[setBits].lines[i].vaild == 0) {
            full=0;
            break;
        }
    }
    if (full==0) {
        sim_cache->sets[setBits].lines[i].vaild=1;
        sim_cache->sets[setBits].lines[i].tag=tagBits;
        updateLruNumber(sim_cache,setBits,i);
    }
    else if (full==1) {
        int evi_Index=findMinLruNumber(sim_cache,setBits);
        sim_cache->sets[setBits].lines[evi_Index].tag=tagBits;
        updateLruNumber(sim_cache,setBits,evi_Index);
    }
    return full;
}
```

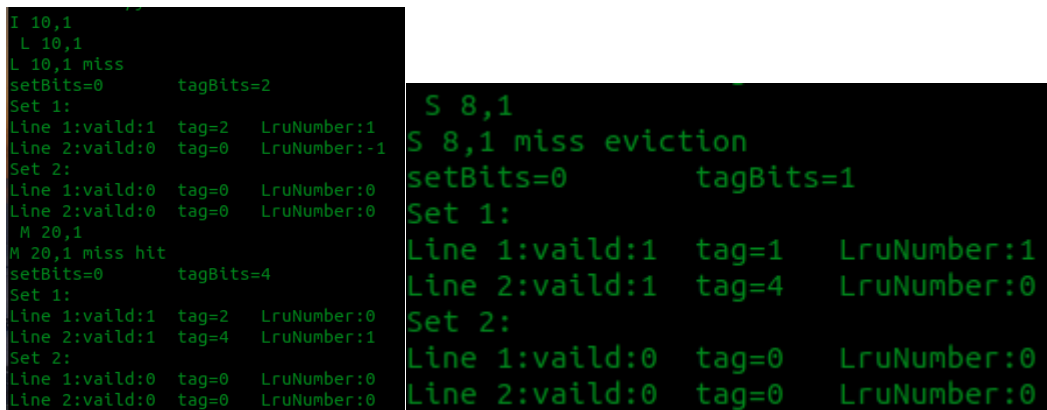
接下来就是依据 LRU 策略来更新 cache, 如果这一个块未滿, 那么就不会发生 eviction, 只需将空的那一行更新其 vaild 和 tagBits, 然后对 LruNumber 进行更新即可; 如果满了, 就要依据 LRU 策略来选择一个块进行 eviction, 此时只需设置它的 tag 就可以, 因为原本的 vaild 就为 1, 最后更新 LruNumber。

三、 检验

```
void testbyHand(Sim_Cache *sim_cache,int s,int E,int b,int isVerbose) {
    char ope[2];
    int addr,size;
    while(scanf("%s %x,%d",ope,&addr,&size)!=EOF) {
        //getchar();
        if (ope[0]=='I')
            continue;
        int setBits,tagBits;
        setBits=( addr>>b ) & ( (1<<s)-1 );
        tagBits=addr>>(s+b);
        printf("%s %x,%d ",ope,addr,size);
        if (ope[0]=='L')
            loadData(sim_cache,setBits,tagBits,isVerbose);
        else if (ope[0]=='S')
            storeData(sim_cache,setBits,tagBits,isVerbose);
        else if (ope[0]=='M')
            modifyData(sim_cache,setBits,tagBits,isVerbose);
        printf("\nsetBits=%d\ttagBits=%d\n",setBits,tagBits);
        putSets(sim_cache);
    }
}
```

这是对上面部分代码进行的检验，不过因为既然要检验 LRU 策略，那么必然的是需要对三个操作的代码进行编写的，于是就提前写了 loadData, storeData, modifyData 三个部分。整体的函数就是通过手动输入指令，根据输入的地址来获取 setBits 和 tagBits，之后对 cache 进行处理，输出需要的信息，来检验 LRU 策略是否正确。

原本打算用%c 来接收要处理的行为，通过 getchar 来吸收换行符，但是由于 I 和其余三个不同，前没有空格，比较麻烦，所以还是采用字符串的形式来接收。首先接收后的第一行输出这一行的操作，并在后面显示对应的 hit 或者其他的行为，最后输出 cache 各个组的信息。



```
I 10,1
L 10,1
L 10,1 miss
setBits=0      tagBits=2
Set 1:
Line 1:vailld:1 tag=2   LruNumber:1
Line 2:vailld:0 tag=0   LruNumber:-1
Set 2:
Line 1:vailld:0 tag=0   LruNumber:0
Line 2:vailld:0 tag=0   LruNumber:0
M 20,1
M 20,1 miss hit
setBits=0      tagBits=4
Set 1:
Line 1:vailld:1 tag=2   LruNumber:0
Line 2:vailld:1 tag=4   LruNumber:1
Set 2:
Line 1:vailld:0 tag=0   LruNumber:0
Line 2:vailld:0 tag=0   LruNumber:0
S 8,1
S 8,1 miss eviction
setBits=0      tagBits=1
Set 1:
Line 1:vailld:1 tag=1   LruNumber:1
Line 2:vailld:1 tag=4   LruNumber:0
Set 2:
Line 1:vailld:0 tag=0   LruNumber:0
Line 2:vailld:0 tag=0   LruNumber:0
```

这是输入输出显示，第一行的 I 不进行任何操作，之后输入 L 10, 1 发生 cold miss，可以看出它的 setBits=0 选择 1 组，写入到第一行，更新该行信息，之后 M 操作同样，写入到第一组第二行，更新，并对 LruNumber 有正常的更新。之后进行的 S 操作也在组 1，但是组 1 满并且未命中，发生 eviction，选择 LruNumber 最小的也就是行 1 驱逐，并更新新值，验证完毕，行为正确。

四、 原型一

```
void loadData(Sim_Cache *sim_cache,int setBits,int tagBits,int isVerbose) {
    if (isMiss(sim_cache,setBits,tagBits)) {
        miss++;
        if (isVerbose==1)
            printf("miss ");
        if (updateCache(sim_cache,setBits,tagBits)==1) {
            eviction++;
            if (isVerboost==1)
                printf("eviction ");
        }
    }
    else {
        hit++;
        if (isVerbose==1)
            printf("hit ");
    }
    return;
}
```

首先判断是否命中，未命中则 miss++ 并在 -v 时输出 miss，之后判断需不需要发生 eviction，发生了则 eviction++，在 -v 时输出 evition，如果命中则 hit++，-v 时输出 hit，

这三个 miss, hit, eviction 都定义为全局变量, 初始值为 0

```
void storeData(Sim_Cache *sim_cache,int setBits,int tagBits,int isVerbose) {
    loadData(sim_cache,setBits,tagBits,isVerbose);
}
void modifyData(Sim_Cache *sim_cache,int setBits,int tagBits,int isVerbose) {
    loadData(sim_cache,setBits,tagBits,isVerbose);
    storeData(sim_cache,setBits,tagBits,isVerbose);
}
```

接下来的 S 操作调用了 loadData, M 操作调用了 storeData 和 loadData, 所以直接调用就行。

五、原型二

```
int getSet(int addr,int s,int b){
    return ( addr>>b ) & ( (1<<s)-1 );
}
int getTag(int addr,int s,int b){
    return addr>>(s+b);
}
```

这两个函数也都在之前检验正确性时编写过, Set 就是依据定义, 将 addr 先左移 b 位, 将对应的 s 位移到最右, 并且与 1 左移 s 位-1 构造的 0x0……f 相与得到。Tag 就是直接将 addr 右移 s+b 位即可得到。

六、编写 main 函数及测试

```
FILE *tracefile=fopen(fileName,"r");
char ope[2];
int addr,size;
while(fscanf(tracefile,"%s %x,%d",ope,&addr,&size)!=EOF) {
    if (ope[0]=='I')
        continue;
    int setBits,tagBits;
    setBits=getSet(addr,s,b);
    tagBits=getTag(addr,s,b);
    printf("%s %x,%d ",ope,addr,size);
    if (ope[0]=='L')
        loadData(&cache,setBits,tagBits,v);
    else if (ope[0]=='S')
        storeData(&cache,setBits,tagBits,v);
    else if (ope[0]=='M')
        modifyData(&cache,setBits,tagBits,v);
    //printf("\nsetBits=%d\ttagBits=%d\n",setBits,tagBits);
    //puts(sim_cache);
    if (v==1)
        printf("\n");
}
printSummary(hit,miss,eviction);
```

与之前第三部分测试时相似, 只不过更改为文件输入, 并且替换变量名的调用, 而且最后注意换行符的控制。

```
ubuntu@ubuntu:~/cachelab-handout$ ./csim
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3

ubuntu@ubuntu:~/cachelab-handout$ ./csim-ref
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
```

```
ubuntu@ubuntu:~/cachelab-handout$ ./csim -s 1 -E 2 -b 2 -t traces/yi2.trace
hits:13 misses:4 evictions:0
ubuntu@ubuntu:~/cachelab-handout$ ./csim-ref -s 1 -E 2 -b 2 -t traces/yi2.trace
hits:13 misses:4 evictions:0
```

```
ubuntu@ubuntu:~/cachelab-handout$ ./csim -s 1 -E 2 -b 2 -t traces/long.trace
hits:128161 misses:158803 evictions:158799
ubuntu@ubuntu:~/cachelab-handout$ ./csim-ref -s 1 -E 2 -b 2 -t traces/long.trace
hits:128161 misses:158803 evictions:158799
```

两者行为一致, 正确。