计算机体系结构lab5

19281030-张云鹏

cache模拟器

数据结构

```
int cachesize;
    int blocksize;
    int assoc;
    int blockinbyte;
    int NOofblock;
    int NOofset;
    int choice;
    float misscount, accesscount, hitcount;
    int index, byte, tag, ii;
    int i = 0, j, x, y, z, cc, c, m;
    int bytearray[20005], wordaddress[20005]; //代表所需数据的字节地址与字地址

    // newarray中存储cache中各块的valid位与tag位；lru中存储最近使用情况，用来实现LRU替换策略（对应位数越大，代表越久未被使用）
    int newarray[300][300] = {0}, lru[300][300] = {0};
```

direct-mapped和set-associative

1. 计算字节地址的低地址, 块地址,set地址和tag

```
wordaddress[j] = bytearray[j] / 4;             //计算数据的字地址
blockaddress[j] = wordaddress[j] / blocksize; //计算数据的块地址
index = blockaddress[j] % NOofset;             //计算数据所在的set地址
tag = blockaddress[j] / NOofset;               //计算数据所在的tag
```

2. 判断是否miss

- 检验valid位

```
if (newarray[index][z] == 0) // 1.1、当cache 中对应位置上的有效位无效，数据不在cache中，失效
//此时能直接判断失效的原因：在多路组相联中，是从左到右依次存数据的，则遇到的第一个valid为0的数据时，后面就不可能再有数据了。
//此时，能够确定是失效，则将该块存入该cache块中（填写tag与valid位）、判断失效类型、更新LRU数组。
{
    newarray[index][z + 1] = tag; //填写tag位
    newarray[index][z] = 1;       //填写有效位为1
```

```
    misscount++;
    c = misstype(blockaddress[j], NOofblock, j); //判断失效类型
    cc = 1;                                      //该cc值其实没用

    //以下为替换策略（近期最少使用策略LRU）的实现方案，主要通过lru[index][]数组实现（代表该块多久未使用，越大代表近期越少使用）。
    //先将index这一行的所有块的lru[index][m]都加1，代表此次未被访问，然后把此次用到的这一个块（z块）的lru[index][z]至0，代表刚刚被访问。
    for (m = 0; m < (assoc * 2); m = m + 2)
        lru[index][m]++; // increase the value of lru[index][m] to 1 in the same index
    lru[index][z] = 0;   // set the  recent use value to 0

    z = (assoc * 2); //退出循环
}
```

- 检验tag位

```
else // 1.2、当cache 中对应位置上的有效位有效，继续判断tag位是否一致
{
    if (newarray[index][z + 1] == tag) // 1.2.1、并且tag位一致，则数据在cache中，命中
    {
        hitcount++; // hit counter increase

        //同上面的原因，为了实现LRU的替换策略，更新lru[index][]数组
        for (m = 0; m < (assoc * 2); m = m + 2)
            lru[index][m]++;
        lru[index][z] = 0;

        z = (assoc * 2); //退出循环
    }
    else // 1.2.2、但是tag位不一致
    {
        if (assoc < 2) // 1.2.2.1、直接映射时，则肯定是失效了（不存在其他可放的块了）
        {
            newarray[index][z + 1] = tag; //直接替换（替换为当前所需要的数据块）
            misscount++;
            c = misstype(blockaddress[j], NOofblock, j); // decide which miss type this miss belong to
            cc = 1;
            z = (assoc * 2); //退出循环
        }
        else // 1.2.2.2、组相联映射时，则可能还有其他块（其他路），循环继续检测
        {
            if (x < lru[index][z]) //该if语句块，保证了y中存放的是当前index这一行中最久未被使用的块的路数（组数），也就是当全部块都被放满，将要被替换的那个块。
            {
                x = lru[index][z];
                y = z;
            }
```

```
            if (z == ((assoc * 2) - 2)) //如果该index中的所有块都已经循环检测过，均无所需的块，则失效，要进行LRU替换，直接替换到y的位置即可。
            {
                newarray[index][y + 1] = tag; // y处即为近期最久未被使用的块，因为其lru[index][]的值最大
                misscount++;
                c = misstype(blockaddress[j], NOofblock, j);
                cc = 1;

                for (m = 0; m < (assoc * 2); m = m + 2) //更新lru[index][]数组
                    lru[index][m]++;
                lru[index][y] = 0;
            }
            z = z + 2; //继续循环，去检测该index的下一组数据（下一个块）
        }
    }
}
```

实验数据分析

实验数据

kind	cache size	block size	n-way	miss rate	hit rate	capacity miss	conflict miss
Direct mapped	64	2	1	0.0001	0.9999	0	0
Direct mapped	64	2	1	0.0001	0.9999	0	0
Set Associative	64	2	2	0.0001	0.9999	0	0
Fully	64	2	2	0.0001	0.9999	0	0

数据分析

1. Block size对miss rate的影响: 当Cache容量一定的时候，若增大Cache块大小，Cache的不命中率先是下降，然后反升。这是因为增加块大小会产生双重作增加了空间局部性，减少了强制性不命中;(2)减少了Cache中块的数目，所以有可能增加冲突不命中。当块比较小时，会超过第二种作用，使不命中率下降:当块比较大时，第(2)种作用超过第(1)种作用，反而使不命中率上上升。
2. Cache size 对miss rate的影响: 根据实验结果可以看出，Cache的不命中率随着Cache容量的增大而降低，但当容量增大到一定程度后(256KB)，再增大Cache的容量变化就不明显了

实验心得

通过本次实验了解了cache映射的三种方式, 从理论分析的角度, 结合芯片设计的物理限制,cache技术有效兼容了高效对低效设备的访问.