lab4

姓名：张浩南
学号：181860134

编程任务

设计4路组相连,4*64*64B=16KB的cache
内存共1MB=2^14块*2^6B/块，而cache有64组，因此20位地址中，中间6位为组号，低6位为块内地址，高8位为标记tag

```
typedef struct {
  uint8_t data[64];//一块cache为64B
  uint32_t tag;
  bool valid;//1合法，0非法
  bool dirty;//1修改，0未修改
}cache_block;
cache_block Cache[64][4];
```

实现`cache_read`时分成三部分，即
1.cache命中时
直接在cache中读数据并返回
2.cache未命中，但组中有空行(valid=0)
从内存读入一块到cache中来，并设置相应tag，valid，dirty信息，最后返回需要的数据
3.cache未命中且无空行
随机编号`repl`，检查是否需要写回（若需要则调用`mem_write`），然后从内存中读入并覆盖，操作类似于2

而`cache_write`同样分为三部分，处理方法与read相似，但不同的是需要选择写入数据的位（即掩码处理），故有以下函数：

```
//块内地址为block_addr，在cache的第grp_id组，组中第i行写入数据data，掩码为wmask
void write_it(uintptr_t grp_id,int i,uintptr_t block_addr,uint32_t data,uint32_t wmask)
{
  uint32_t *p = (void *)(Cache[grp_id][i].data) + (block_addr & ~0x3);
  *p = (*p & ~wmask) | (data & wmask);
}
```

最后实现`init_cache`，完成初始化

```
void init_cache(int total_size_width, int associativity_width) {
  int grp_tot=(1<<associativity_width),//组数
      block_tot=(1<<total_size_width)/BLOCK_SIZE/grp_tot;//块数
  for (int i=0;i<block_tot;++i)
  for (int j=0;j<grp_tot;++j)
    {
      Cache[i][j].dirty=false;
      Cache[i][j].valid=false;
    }
}
```

评估任务

每次测试指定三种随机种子0,10000,1576854509，观察访问cache的总次数total，击中次数hit，未击中次数miss以及集中率hit rate
除此以外还考察cache击中时的读/写平均时间与未击中时的读/写平均时间
设计了三种不同的cache，分别是
1.4路组相连，cache大小16KB
2.2路组相连，cache大小16KB

3.4路组相连，cache大小256B

第一种cache（4路组相连，cache大小16KB）

```
random seed = 0
Random test pass!
Statistic:
cycle:50484343
total:9388608
hit:7864725
miss:1523883
hit rate=0.837688
cache miss read averge time:53.050424 ns
cache miss write averge time:37.872569 ns
cache hit read averge time:21.039571 ns
cache hit write averge time:0.001464 ns
```

```
random seed = 10000
Random test pass!
Statistic:
cycle:50476977
total:9388608
hit:7864704
miss:1523904
hit rate=0.837686
cache miss read averge time:53.371582 ns
cache miss write averge time:38.297681 ns
cache hit read averge time:21.030501 ns
cache hit write averge time:0.001408 ns
```

```
random seed = 1576854509
Random test pass!
Statistic:
cycle:50485613
total:9388608
hit:7864669
miss:1523939
hit rate=0.837682
cache miss read averge time:55.069820 ns
cache miss write averge time:39.190140 ns
cache hit read averge time:24.232678 ns
cache hit write averge time:0.001498 ns
```

第二种cache（2路组相连，cache大小16KB）

```
random seed = 0
Random test pass!
Statistic:
cycle:50484791
total:9388608
hit:7864727
miss:1523881
hit rate=0.837688
cache miss read averge time:56.145353 ns
cache miss write averge time:40.403119 ns
cache hit read averge time:21.547322 ns
cache hit write averge time:0.003655 ns
```

```
random seed = 10000
Random test pass!
Statistic:
cycle:50476927
```

```
total:9388608
hit:7864697
miss:1523911
hit rate=0.837685
cache miss read averge time:62.003686 ns
cache miss write averge time:44.312005 ns
cache hit read averge time:21.992612 ns
cache hit write averge time:0.003438 ns
```

```
random seed = 1576854509
Random test pass!
Statistic:
cycle:50485379
total:9388608
hit:7864670
miss:1523938
hit rate=0.837682
cache miss read averge time:62.376066 ns
cache miss write averge time:43.901865 ns
cache hit read averge time:25.217198 ns
cache hit write averge time:0.003296 ns
```

第三种cache（4路组相连，cache大小256B）

```
random seed = 0
Random test pass!
Statistic:
cycle:50498059
total:9388608
hit:7864328
miss:1524280
hit rate=0.837646
cache miss read averge time:65.240549 ns
cache miss write averge time:41.032573 ns
cache hit read averge time:21.738649 ns
cache hit write averge time:0.000035 ns
```

```
random seed = 10000
Random test pass!
Statistic:
cycle:50488922
total:9388608
hit:7864327
miss:1524281
hit rate=0.837646
cache miss read averge time:66.380546 ns
cache miss write averge time:40.339517 ns
cache hit read averge time:21.021543 ns
cache hit write averge time:0.000034 ns
```

```
random seed = 1576854509
Random test pass!
Statistic:
cycle:50496467
total:9388608
hit:7864328
miss:1524280
hit rate=0.837646
cache miss read averge time:64.312405 ns
cache miss write averge time:41.083600 ns
cache hit read averge time:20.955733 ns
cache hit write averge time:0.000014 ns
```

从击中率上看，不同随机种子和不同cache下相差很小，基本维持在83.76%~83.77%的区间内。

而从复杂性（击中时的用时）来看，在读方面第三种cache的表现更高，能稳定在21ns且浮动很小；第一种cache存在数据偏好性，对较大的随机种子表现较差(24ns)，而在其他数据上和第三种表现相当；而第二种的表现均落后于其他两种，不够稳定且在同种子下也慢1ns左右。

而在写方面呈现了极为明显的阶梯分布，即第三种好于第一种好于第二种，不过相对于读代价来看，写代价的耗时极小，在读写次数平衡情况下，计算总代价时几乎可以将其忽略。

从miss时的代价来看，读方面，是第一种好于第二种好于第三种，第一种一次read大概需要53~55ns的耗时，而第二种则不够稳定，且全面劣于第一种（56>53,62>55），第三种较稳定但也最慢，约需要65ns。
而在写方面同样是第一种（38~39ns）好于第三种（40~41ns）好于第二种（44ns左右）

而考虑平均读时间，第一种大约是22*0.8376+54*(1-0.8376)=27.1968ns；第二种大约是23*0.8376+60*(1-0.8376)=29.0088ns；第三种21*0.8376+65*(1-0.8376)=28.1456ns，第一种好于第三种好于第二种。
平均读时间上，第一种大约是38*0.8376+0.0015*(1-0.8376)=31.829044ns；第二种大约是41*0.8376+0.0034*(1-0.8376)=34.342152ns；第三种大约是40*0.8376=33.504000ns，第一种好于第三种好于第二种。

总的来说，第一种的表现略优于其他两种，且第三种的cache大小较小，miss时的代价较高；而第二种的2路组相连在实验中表现较差，第一种的分配较为平衡，因此**第一种**应是比较优的cache设计。