计算机体系结构
实验4 层次存储系统分析

江仲鸣

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

实验目的

- 加深对Cache基本结构及工作原理的理解
- 掌握使用程序测量Cache结构参数的基本方法

实验内容

1. 编写C程序测量L1 DCache的容量、块大小、相联度等结构参数
2. 编写C程序测量TLB命中与缺失时的平均访存时间，测量TLB的entry数

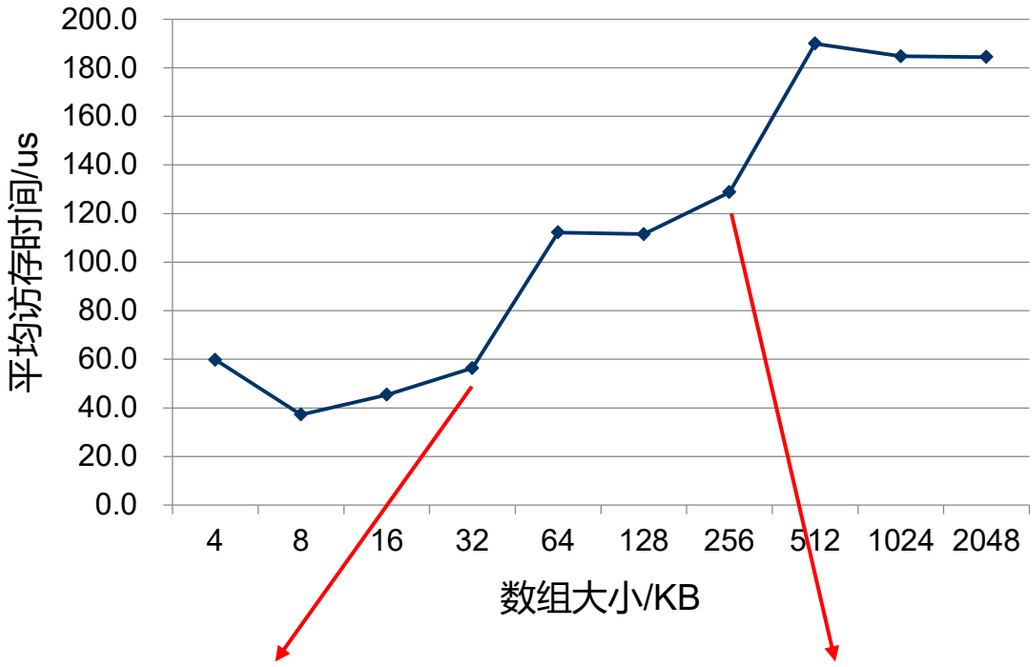
实验原理

- **1、Cache容量测量**
  - 测量原理：不同层次的存储器存在访问速度差异
    - $\overline{t_{访存}}$ = Cache命中率*命中访存时间 + Cache缺失率*缺失访存时间
  - 测量方法：连续访问不同大小的数组，记录平均访存时间
    - 数组大小超过L1 DCache，缺失率明显增加，平均访存时间增加
    - 数组大小超过L2 Cache，缺失率再次增加，平均访存时间增加
    - 平均访存时间呈阶梯状

实验原理

- 2、测量Cache容量
  - 某PC机的某次实验结果示例（L1 DCache为32 KB，L2 Cache为256 KB）

```
  4KB]      Average access time: 59.8us
  8KB]      Average access time: 37.2us
 16KB]      Average access time: 45.4us
 32KB]      Average access time: 56.4us
 64KB]      Average access time: 112.2us
128KB]      Average access time: 111.5us
256KB]      Average access time: 128.8us
512KB]      Average access time: 189.9us
1024KB]     Average access time: 184.7us
2048KB]     Average access time: 184.4us
```



数组大于 32KB 后
平均访存时间明显增加

数组大于 256KB 后
平均访存时间也明显增加

实验原理

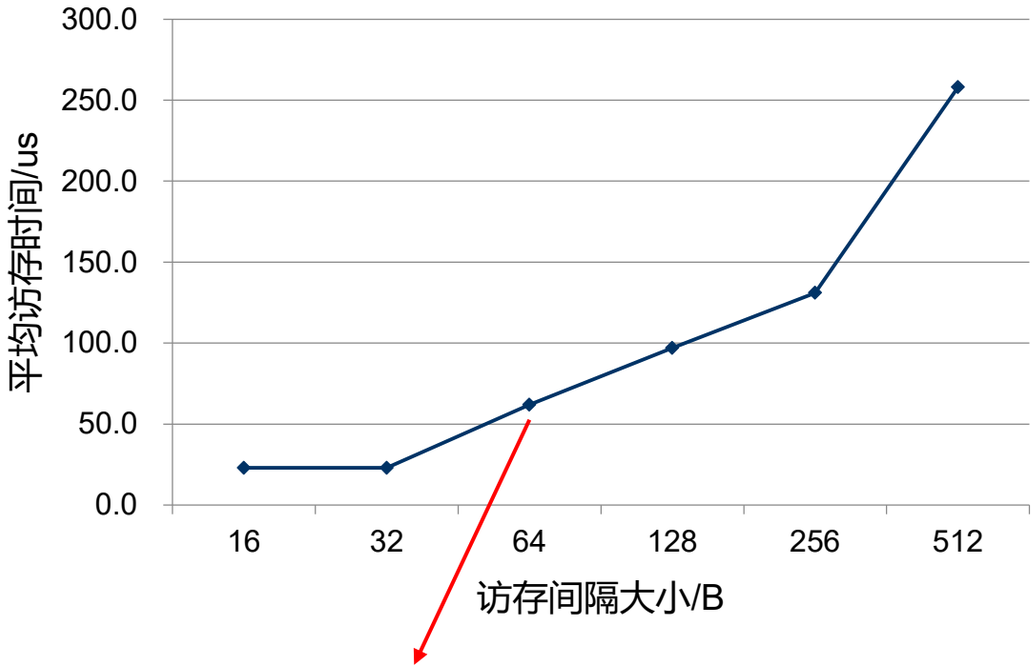
- **3、测量Cache块大小**
  - 测量原理：CPU以字节/字访问Cache，Cache以数据块访问主存
  - 测量方法：以一定间隔访问连续数组
    - 间隔小于块大小时，命中率较高，平均访存时间较短
    - 间隔大于等于块大小时，理论上命中率为0
      - 平均访存时间 ≈ 下一级存储器的平均访存时间

实验原理

- **3、测量Cache块大小**
  - 某PC机的某次实验结果示例

    (L1 DCache块大小为64B)

```
 16B]        Average access time = 23.0us
 32B]        Average access time = 23.0us
 64B]        Average access time = 62.0us
128B]        Average access time = 97.0us
256B]        Average access time = 131.0us
512B]        Average access time = 258.0us
```

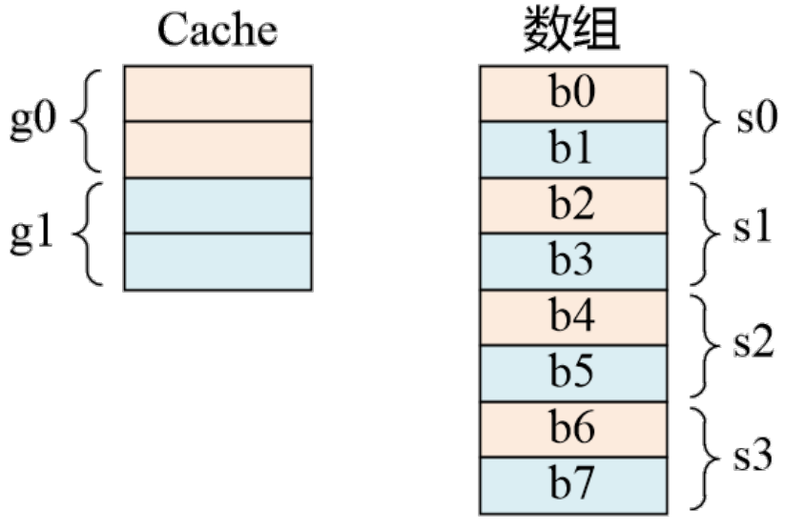


为何访问间隔大于64B后，平均访存时间继续明显增加?

步长为64B时，平均访存时间明显增加，
说明 Cache 缺失率明显增加

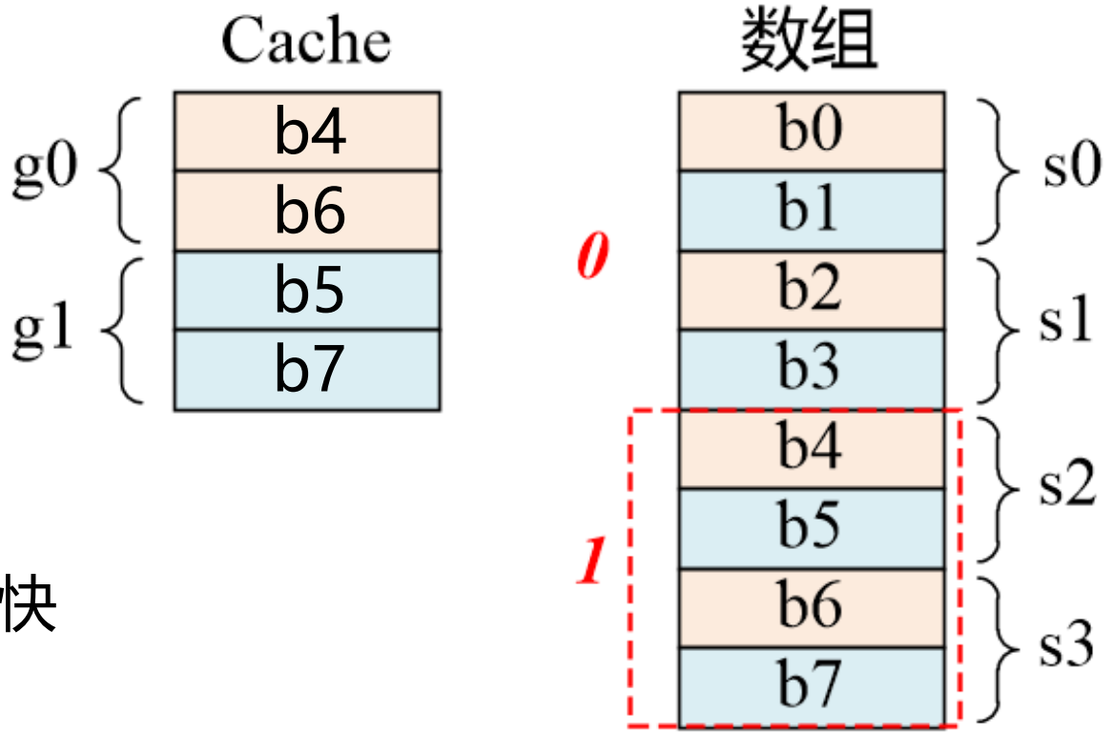
实验原理

- **4、测量Cache相联度**
  - 测量原理：按照组相联映射规则进行访存
  - 测量方法：建立2倍Cache大小的数组，平均分成2^n块，访问奇数块
    - 逐渐增大n，记录平均访存时间
    - 平均访问时间突然增加时，出现临界值n'，相联度为$2^{n'-2}$
  - 举例：2路组相联Cache



实验原理

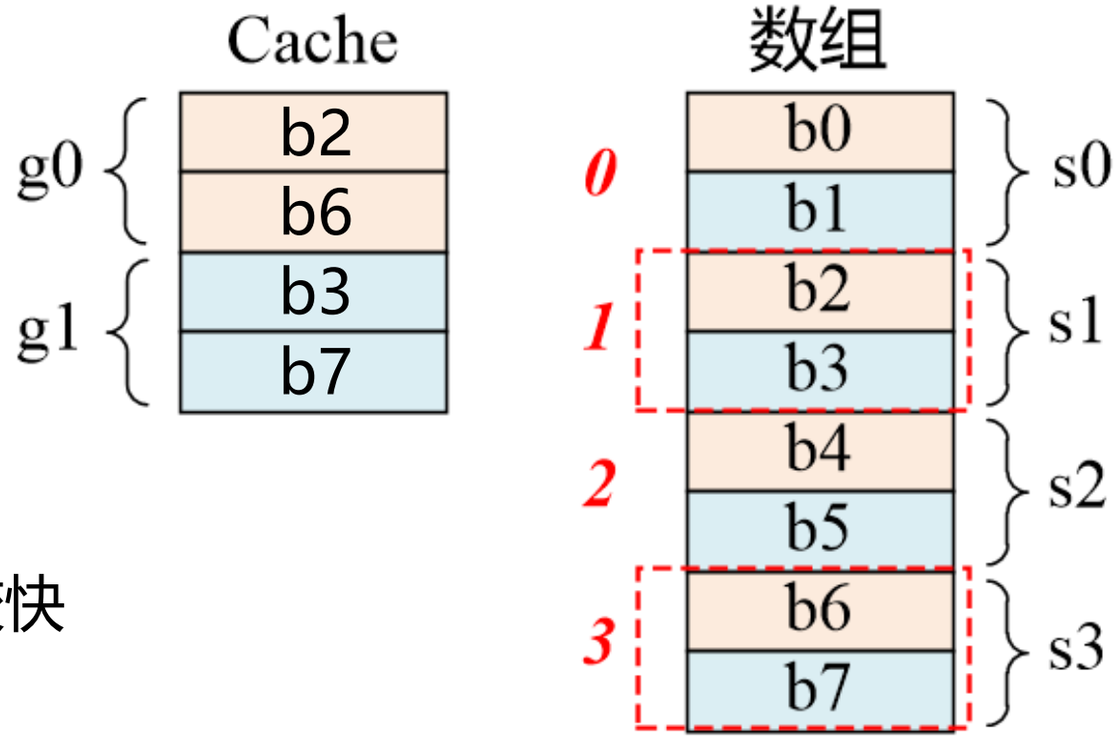
- **4、测量Cache相联度**
  - 举例：2路组相联Cache
    - 1）n取1，将数组分成2^1块，访问第1块



大部分命中，访问较快

实验原理

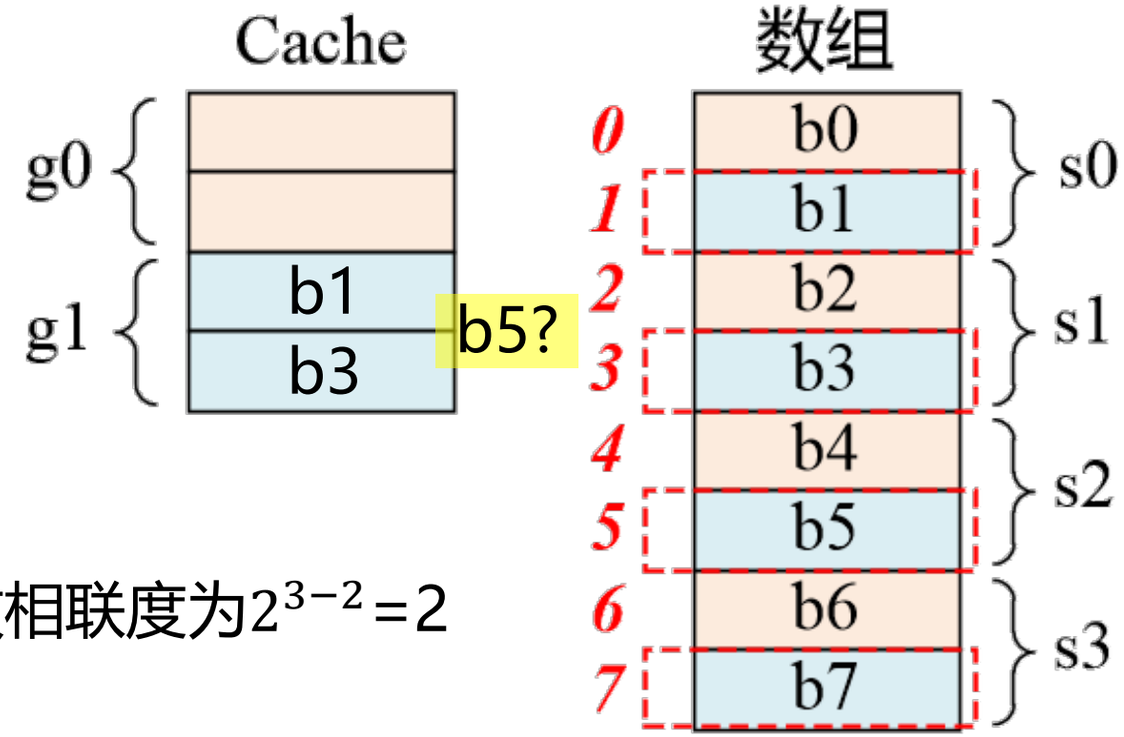
- **4、测量Cache相联度**
  - 举例：2路组相联Cache
    - 2）n取2，将数组分成2^2块，访问第1、3块



大部分命中，访问较快

实验原理

- **4、测量Cache相联度**
  - 举例：2路组相联Cache
    - 3）n取3，将数组分成2^3块，访问第1、3、5、7块



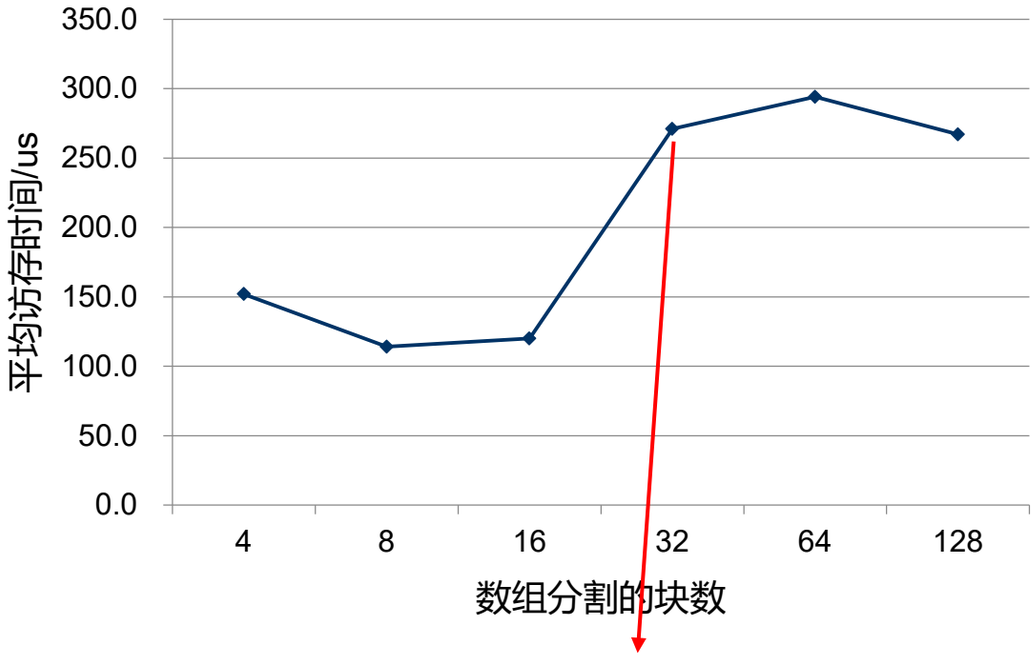
n=3时访问变慢，故相联度为$2^{3-2}=2$

实验原理

- **4、测量Cache相联度**
  - 某PC机的某次实验结果示例

    (L1 DCache为8路组相联)

```
 4]        Average access time = 152.0us
 8]        Average access time = 114.0us
16]        Average access time = 120.0us
32]        Average access time = 271.0us
64]        Average access time = 294.0us
128]       Average access time = 267.0us
```



分成32块(n=5)时，平均访存时间明显增加，
所以测得相联度为$2^{n-2} = 8$

实验步骤

- ## 1、查询本机器的Cache结构

运行cpu-z，查看本机的Cache结构

执行x86info -c查看TLB结构信息

```
debian@debian:~$ x86info -c
x86info v1.31pre  Dave Jones 2001-2011
Feedback to <davej@redhat.com>.

Found 6 identical CPUs
Extended Family: 0 Extended Model: 9 Family: 6 Model:
Type: 0 (Original OEM)
CPU Model (x86info's best guess): Unknown model.
Processor name string (BIOS programmed): Intel(R) Core

Cache info
TLB info
 Data TLB: 4KB pages, 4-way associative, 64 entries
 Data TLB: 4KB pages, 4-way associative, 64 entries
 64 byte prefetching.
Found unknown cache descriptors: 63 76 b5 c3 ff
Total processor threads: 6
This system has 1 six-core processor running at an est
```

实验步骤

- 2、编写程序
  - 将模板程序源文件cache_test.c拷贝到虚拟机的任意目录
  - 在TODO标记处补全代码，完成各测量函数的编写
  - 使用gettimeofday()、get_usec()计时

```
void Test_Cache_Size()
{
    printf("************************************************************\n");
    printf("L1 DCache and L2Cache Size Test\n");

    // TODO

    /**
     * struct timeval tp[2];
     * gettimeofday(&tp[0], NULL);
     * func();
     * gettimeofday(&tp[1], NULL);
     * time_used = getusec(tp[0], tp[1]);
     */
}

void L1_DCache_Block()
{
    printf("************************************************************\n");
    printf("L1 DCache Block Size Test\n");

    Clear_L1_Cache();                                   // Clear L1 Cache

    // TODO
}

void L2_Cache_Block()
{
    printf("************************************************************\n");
    printf("L2 Cache Block Size Test\n");

    Clear_L2_Cache();                                   // Clear L2 Cache

    // TODO
}

void L1_DCache_Way_Count()
{
    printf("************************************************************\n");
```

实验步骤

- **2、编写程序**
  - 访存次数太少，测量结果容易受各种因素影响而不稳定
  - CPU与L1 Cache之间存在优化访存的缓存，采用写操作访存受影响较小
  - 以一定的步长访存，或随机地址访存
  - 去除循环、运算、赋值等时间，统计净访存时间
  - 对于需要计时的代码块内部的中间变量，使用register约束以减少无关访存操作
  - 确保不同测试参数下的访存次数相同
  - 可以自行编写程序从测出的访存时间中解析出Cache参数（选）

实验步骤

- 3、运行程序并查看结果
  - 运行命令编译程序
    - gcc -Og cache_test.c -o cache_test
  - 运行命令执行程序
    - ./cache_test

```
debian@debian:~$ gcc -Og cache_test.c -o cache_test && ./cache_test
************************************************************
L1 DCache and L2Cache Size Test
[Test_Array_Size =   4KB]       Average access time: 59.8us
[Test_Array_Size =   8KB]       Average access time: 37.2us
[Test_Array_Size =  16KB]       Average access time: 45.4us
[Test_Array_Size =  32KB]       Average access time: 56.4us
[Test_Array_Size =  64KB]       Average access time: 112.2us
[Test_Array_Size = 128KB]       Average access time: 111.5us
[Test_Array_Size = 256KB]       Average access time: 128.8us
[Test_Array_Size = 512KB]       Average access time: 189.9us
[Test_Array_Size = 1024KB]      Average access time: 184.7us
[Test_Array_Size = 2048KB]      Average access time: 184.4us
L1_DCache_Size is 32KB
L2_Cache_Size is 256KB
************************************************************
L1 DCache Block Size Test
[Test_Array_Jump =  16B]        Average access time = 23.0us
[Test_Array_Jump =  32B]        Average access time = 23.0us
[Test_Array_Jump =  64B]        Average access time = 62.0us
[Test_Array_Jump = 128B]        Average access time = 97.0us
[Test_Array_Jump = 256B]        Average access time = 131.0us
[Test_Array_Jump = 512B]        Average access time = 258.0us
L1_DCache_Block is 64B
************************************************************
L2 Cache Block Size Test
[Test_Array_Jump =  16B]        Average access time = 24.0us
[Test_Array_Jump =  32B]        Average access time = 25.0us
[Test_Array_Jump =  64B]        Average access time = 68.0us
[Test_Array_Jump = 128B]        Average access time = 103.0us
[Test_Array_Jump = 256B]        Average access time = 137.0us
[Test_Array_Jump = 512B]        Average access time = 250.0us
L2_Cache_Block is 64B
************************************************************
L1 DCache Way Count Test
[Test_Split_Groups =  4]        Average access time = 152.0us
[Test_Split_Groups =  8]        Average access time = 114.0us
[Test_Split_Groups = 16]        Average access time = 120.0us
[Test_Split_Groups = 32]        Average access time = 271.0us
[Test_Split_Groups = 64]        Average access time = 294.0us
[Test_Split_Groups = 128]       Average access time = 267.0us
L1_DCache_Way_Count is 8 ways
************************************************************
L2 Cache Way Count Test
[Test_Split_Groups =  2]        Average access time = 224.0us
[Test_Split_Groups =  4]        Average access time = 222.0us
[Test_Split_Groups =  8]        Average access time = 225.0us
[Test_Split_Groups = 16]        Average access time = 244.0us
[Test_Split_Groups = 32]        Average access time = 252.0us
[Test_Split_Groups = 64]        Average access time = 251.0us
L2_Cache_Way_Count is 4 ways
************************************************************
TLB Size Test
[Test_TLB_entries =  16]        average access time: 878.2 us
[Test_TLB_entries =  32]        average access time: 880.0 us
[Test_TLB_entries =  64]        average access time: 962.9 us
[Test_TLB_entries = 128]        average access time: 1496.7 us
[Test_TLB_entries = 256]        average access time: 1541.2 us
[Test_TLB_entries = 512]        average access time: 1536.1 us
The number of TLB entries is 64
************************************************************
```

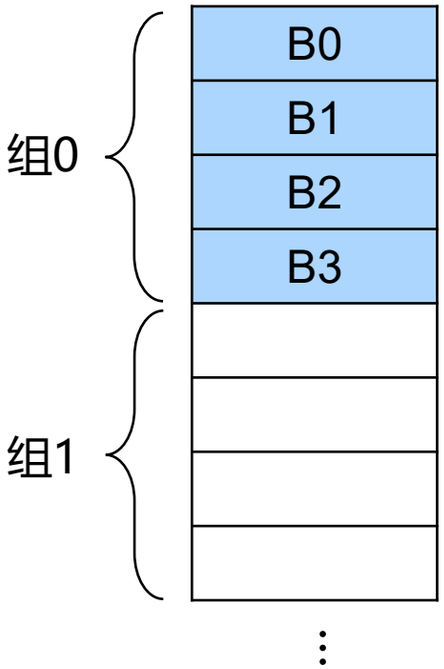
附加题

- **1、测量L2 Cache容量、块大小、相联度（+0.5分）**
  - 从原理、实现2个方面，在报告中描述测量L1 Dcache、L2 Cache时的区别

- **2、测量Cache写回策略（+1分）**
  - 写直达：数据在写到Cache块的同时，还要写入存储器
    - 写命中与写缺失的延迟相差不大——总是要写内存
  - 写回法：数据仅写入Cache；仅当Cache的数据块被替换时，才写回到主存中
    - 写命中与写缺失的延迟相差较大——只有写缺失需要写内存

附加题

- **3、测量Cache替换策略（+1分）**
  - 根据测得的相联度，设计不同的访问序列，产生不同次数的缺失，以验证Cache所使用的替换策略（如LRU、FIFO、随机等）



- 先“填满”某个Cache组，再设计一定的Cache块访问顺序
- 替换策略不同，缺失次数不同

实验报告要求

- 算法原理理解及设计思路
  - 若采用其他测量方法，介绍原理并通过举例/证明来分析其正确性
- 关键实现
  - 可借助图表描述
- 测试结果及分析
  - 需绘制测量测参数时，Cache的访问时间折线图
- 回答实验指导书中的思考题

实验检查与提交

- 课堂检查：所测量的参数是否准确
- 将**源码**(.c)、**实验报告**打包提交
  - 命名规则：**学号_姓名_ARCH实验4.zip**
  - 提交方法：https://hitsz-cslab.gitee.io/arch/ojguide
  - Deadline：下周同一上课时间前
- 附加题：将设计思路、关键代码等写入报告，与源码一起打包提交（**+0.5~1**分）

开始实验