**第五章**

**5.2 cache为处理器提供了一个高性能的存储器层次结构，因此十分重要。下面是一个32位存储器地址引用的列表，给出的是字地址。**

**3，180，43, 2，191，88, 190, 14，181, 44, 186，253**

**5.2.1 [10]<5.3>已知一个直接映射的cache 有16个块，块大小为1个字。对于每次访问，请标识出二进制地址、标记以及索引。假设cache最开始为空，那么请列出每次访问是命中还是缺失。**

|  |  |
| --- | --- |
|  | **4** |

**5.2.2 [10]<5.3>已知一个直接映射的cache 有8个块，块大小为2个字。对于每次访问，请标识出二进制地址、标记以及索引。假设cache最开始为空，那么请列出每次访问是命中还是缺失。**

|  |  |  |
| --- | --- | --- |
|  | **3** | **1** |

**5.2.3 [20]<5.3, 5.4>对已知的访问来优化cache的设计。这里有三种直接映射的cache 设计方案，每个容量都为8个字:C1块大小为1个字，C2块大小为2个字，C3块大小为4个字。根据缺失率，哪种cache设计最好？如果缺失阻塞时间为25个周期，Cl的访问时间为2个周期，C2为3个周期，C3为5个周期，那么哪种cache设计最好?**

|  |  |
| --- | --- |
|  | **3** |

|  |  |  |
| --- | --- | --- |
|  | **2** | **1** |

|  |  |  |
| --- | --- | --- |
|  | **1** | **2** |

**这里有许多对cache整体性能很重要的不同的设计参数。下面列出了对于不同的直接映射cache设计的参数。**

**cache数据量:** **32KiB = 32KiB/4 字节 = 8KiW = 213W**

**cache块大小: 2个字**

**cache访问时间: 1个周期**

**5.2.4 [15]<5.3> 假定32位的地址，计算上面列出的cache所需的总位数。给定总的大小，找出最接近的直接映像cache的总的大小，该cache块的大小为16个字长或更大。请解释为什么第二种cache比第一种cache的访问速度更慢，尽管第二种cache的数据量更大。**

**5.2.5 [20]<5.3, 5.4>在一个2KiB的****两路组相联cache上产生一系列读请求时的缺失率要比在表中cache上执行读请求的缺失率低。请给出一个可能的解决方案，使得表中列出的cache的缺失率等于或者低于2KiB cache的缺失率。讨论这种解决方案的优点和缺点。**

**5.2.6 [15]<5.3>5.3 节的公式说明了用来索引直接映射cache的典型方法: (块地址) mod (cache中的块数)。假设地址为32位，cache中有1024个块，考虑一个不同的索引函数: (块地址[31:27]XOR块地址[26:22])可以使用这个公式来索引直接映射的cache吗?如果可以，请解释原因，并且讨论可能需要对cache做的一些改动。如果不可以，请解释原因。**

解：

1. 由于cache有16个块，24=16，所以二进制的字地址后四位为块号，前面其他位为标记位

主存地址格式：

|  |  |
| --- | --- |
|  | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字地址 | 二进制地址 | 标记 | 索引 | 命中/缺失 |
| 3 | 0000 0011 | 0 | 3 | 缺失 |
| 180 | 1011 0100 | 11 | 4 | 缺失 |
| 43 | 0010 1011 | 2 | 11 | 缺失 |
| 2 | 0000 0010 | 0 | 2 | 缺失 |
| 191 | 1011 1111 | 11 | 15 | 缺失 |
| 88 | 0101 1000 | 5 | 8 | 缺失 |
| 190 | 1011 1110 | 11 | 14 | 缺失 |
| 14 | 0000 1110 | 0 | 14 | 缺失 |
| 181 | 1011 0101 | 11 | 5 | 缺失 |
| 44 | 0010 1100 | 2 | 12 | 缺失 |
| 186 | 1011 1010 | 11 | 10 | 缺失 |
| 253 | 1111 1101 | 15 | 13 | 缺失 |

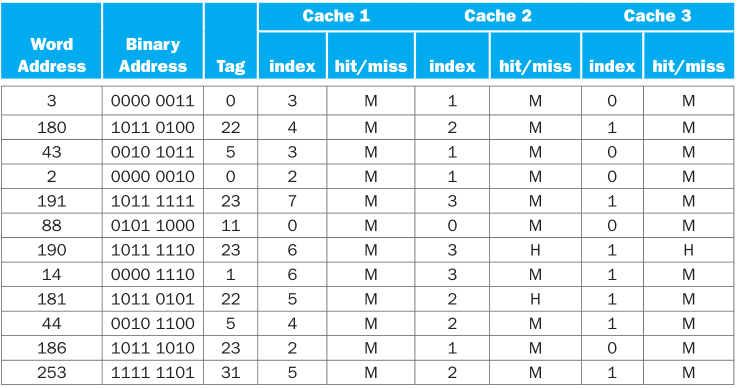
1. 由于cache 有8个块，23=8，且块的大小为2个字，所以二进制的字地址倒数中间3位为cache块号（索引），前四位为标记

主存地址格式：

|  |  |  |
| --- | --- | --- |
|  | 3 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字地址 | 二进制地址 | 标记 | 索引 | 命中/缺失 |
| 3 | 0000 0011 | 0 | 1 | 缺失 |
| 180 | 1011 0100 | 11 | 2 | 缺失 |
| 43 | 0010 1011 | 2 | 5 | 缺失 |
| 2 | 0000 0010 | 0 | 1 | 命中 |
| 191 | 1011 1111 | 11 | 7 | 缺失 |
| 88 | 0101 1000 | 5 | 4 | 缺失 |
| 190 | 1011 1110 | 11 | 7 | 命中 |
| 14 | 0000 1110 | 0 | 7 | 缺失 |
| 181 | 1011 0101 | 11 | 2 | 命中 |
| 44 | 0010 1100 | 2 | 6 | 缺失 |
| 186 | 1011 1010 | 11 | 5 | 缺失 |
| 253 | 1111 1101 | 15 | 4 | 缺失 |





Cache1 cache缺失率 100%

Cache1的总周期：12\*2+25\*12 = 324

Cache2 cache缺失率 10/12 = 83%

Cache2的总周期：12\*3+25\*10 = 286

Cache3 cache缺失率 11/12 = 92%

Cache3的总周期：12\*5+25\*11 = 335 cache2性能好

1. 32KiB = 213 个字 则我们有 213/2 = 212 =4096 个块 索引位为12位 字偏移量1位 字节偏移量2位 所以标记位有 32-12-1-2 = 17位

标记位17 + 1（有效位）= 18 位

Cache总大小为 4096\*（2\*32+18）= 335872 bits = 41984 bytes

从2字的块增加到16字的块，标记位会从17位变成14位

总大小 = 数据大小 + (有效位数 + 标记位数) \* 块数

数据大小 = 块数 \* 块大小 \* 字大小

我们设块数为x，则有41984 <= 16\*32\*x +（14+1）\* x

解得 x >= 41984/527 =79.7 > 64(26) 所以我们只能取27个块的cache

与原始cache相比，选择更大的块可能会有更高的缺失率。因为cache中块数变少，对于这些块将会有大量的竞争发生，造成一个块中的数据在多次访问前就被替换出cache。另外，块中各字之间的空间局部性也会降低。

1. 关联cache的设计目的是降低冲突缺失的发生率。

具有相同的12位索引字段，但不同的标记字段的读请求序列将产生许多缺失。对于上面描述的cache，序列0, 32768, 0, 32768, 0, 32768, 0, 32768，…，每一个访问都会缺失，而一个使用LRU策略的两路组相联cache，在前两次缺失后，之后的每一次访问都会命中。

1. 是的，可以使用这个函数索引cache。但是，有5位的信息会丢失，因为这些位是经过XOR 运算的，所以必须包含更多的标记位来标识cache中的地址。

**5.3 对于一个32位地址的直接映射的cache 设计，下面的地址位用来访问cache。**

**C:\Users\jingming\AppData\Local\Temp\1560515827(1).png**

**5.3.1 [5] <5.3> cache块大小是多少(单位为字)?**

**5.3.2 [5] <5.3 >cache有多少项?**

**5.3.3 [5] <5.3>这样的cache执行时所需的总位数与数据存储位数之间的比率是多少?**

**下表记录了从上电开始的cache访问的字节地址。**

**C:\Users\jingming\AppData\Local\Temp\1560515845(1).png**

**5.3.4 [10] <5.3>有多少块被替换**

**5.3.5 [10] <5.3>命中率是多少?**

**5.3.6 [20] <5.3 >列出cache的最终状态，每个有效项以记录的形式<索引，标记，数据>表示出来。**

解：

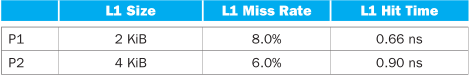
1. 偏移位5位 则块大小为25字节 = 23字
2. 25=32项
3. （1+22+23\*32）/ 23\*32 =1.09 ？？？
4. 四块被替换

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 标记 | 索引（十进制） | 偏移 | 命中/缺失 |
| 0 | …0000 | 00000（0） | 00000 | 缺失 |
| 4 | …0000 | 00000（0） | 00100 | 命中 |
| 16 | …0000 | 00000（0） | 10000 | 命中 |
| 132 | …0000 | 00100（4） | 00100 | 缺失 |
| 232 | …0000 | 00111（7） | 01000 | 缺失 |
| 160 | …0000 | 00101（5） | 00000 | 缺失 |
| 1024 | …0001 | 00000（0） | 00000 | 缺失（替换） |
| 30 | …0000 | 00000（0） | 11110 | 缺失（替换） |
| 140 | …0000 | 00100（4） | 01100 | 命中 |
| 3100 | …0011 | 00000（0） | 11100 | 缺失（替换） |
| 180 | …0000 | 00101（5） | 10100 | 命中 |
| 2180 | …0010 | 00100（4） | 00100 | 缺失（替换） |

1. 4次命中，命中率4/12 = 1/3
2. 最终状态

|  |  |  |  |
| --- | --- | --- | --- |
| 索引 | 有效位 | 标记 | 数据 |
| 00000 | 1 | …0011 | 3100 |
| 00001 | 0 |  |  |
| …… | 0 |  |  |
| 00100 | 1 | …0010 | 2180 |
| …… | 0 |  |  |
| 00101 | 1 | …0000 | 180? |
| …… | 0 |  |  |
| 00111 | 1 | …0000 | 232 |
| …… | 0 |  |  |

**5.6 在这个练习中，我们将研究不同容量对整体性能的影响。通常来说，cache访问时间与cache容量成正比。假设访问主存需要70ns,并且在所有指令中，有36%的指令需要访存。下表是P1和P2两个处理器各自的一级cache的数据。**

****

**5.6.1 [5]<5.4> 假定-级cache命中时间决定了PI和P2的周期时间，它们各自的时钟频率是多少?**

**5.6.2 [5]<5.4> PI和P2各自的AMAT (平均存储器访问时间)分别是多少?**

**5.6.3 [5]<5.4> 假定在没有任何存储器阻塞时基本的CPI为1.0, P1和P2各自的总CPI分别是多少?哪个处理器更快?**

**对下面的三个问题，我们考虑在P1中增加二级cache,以弥补一级cache容量的限制。在解决这些问题时，依然使用上表中一级cache的容量和命中时间。二级cache缺失率是它的局部缺失率。**

**C:\Users\jingming\AppData\Local\Temp\1560601801(1).png**

**5.6.4 [10] <5.4>增加二级cache后，PI的AMAT是多少?有了二级cache, AMAT是更好还是更差了?**

**5.6.5 [5]<5.4>假定在没有任何存储器阻塞时基本的CPI为1.0,增加二级cache后，P1的总的CPI是多少?**

**5.6.6 [10]<5.4>P1有了二级 cache后，哪个处理器更快?如果P1更快，那么P2中一级cache的缺失率需要为多少才能匹配P1的性能?如果P2更快，那么P1中一级cache的缺失率需要为多少才能匹配P2的性能?**

解：

1. P1=1/0.66\*=1.52GHz P2=GHz
2. P1=0.66+8%\*70=6.26ns P2=0.9+0.06\*70=5.1ns
3. P1缺失代价 = 70/0.66 = 106个时钟周期

P1缺失时钟周期 = 0.36\*0.08\*106 = 3.05

P1总CPI =1+3.05 = 4.05

P1性能= 1/（I\*1.52\*4.05）=0.162/I

P2性能= 1/（I\*1.11\*2.68）=0.336/I

所以P2快

P2缺失代价 = 70/0.9 = 77.8个时钟周期

P2缺失时钟周期 = 0.36\*0.06\*77.8 = 1.68

P2总CPI =1+1.68 = 2.68

1. P1 AMAT = 0.66+0.08\*(0.95\*70+5.62) = 6.34 > 6.26 更差了
2. 二级cache缺失代价 = 70/5.62 = 12.5周期

缺失时钟周期 = 0.36\*0.08\*0.95\*12.5 = 0.34

总CPI = 1+0.34 =1.34

1. P1（加了2级cache）性能 = 1/（I\*1.52\*1.34）= 0.49/I

P2（未加2级cache）性能 = 0.336/I

所以P1更快 设P2中一级cache的缺失率为x，则有

0.36\*x\*77.8+1 = 1.34 => x = 1.2%

（注：若理解为 P2加了2级cache，则

P2缺失的时钟周期= 0.36\*0.06\*0.95\*12.5=0.2565

P2总cpi = 1+0.2565 = 1.2565

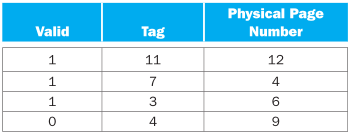
P2性能 = 1、（I\*1.11\*1.2565）=0.717/I

那就是P2快了….）

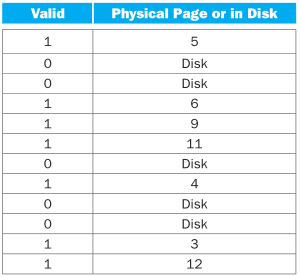
**5.11如5.7节所述，虚拟存储器使用一个页表来追踪虚拟地址到物理地址之间的映射。这个练习说明了当地址被访问时，页表如何更新。下表是在一个系统上所看见的虚拟地址流。假设4KiB页，一个4项的全相联TLB,使用严格的LRU替换算法。如果必须从磁盘中取回页，那么增加下一次能取的最大页数:**

**4669, 2227, 13916, 34587, 48870, 12608, 49225**

**TLB**

****

**页表**

****

**5.11.1 [10] <5.7>根据给出的地址流，以及TLB、页表的初始状态，请给出系统的最终状态。对于每次访问，请列出是否在TLB中命中，是否在页表中命中或是发生缺页。**

**5.11.2 [15] <5.7>重复练习题5.11.1,但是这次使用16KiB的页来代替4KiB的页。使用更大的页的好处有哪些?缺点又有哪些?**

**5.11.3 [15] <5.4,5.7>如果使用两路组相联的TLB,请指出TLB中最终的内容。如果TLB是直接映射的，同样指出TLB中最终的内容。讨论使用TLB来获得高性能的重要性。如果没有TLB,那么如何处理虚拟存储器访问?**

**有一些参数对页表整个大小会有影响。下面就列出一些关键的页表参数。**

**C:\Users\jingming\AppData\Local\Temp\1560516479(1).png**

**5.11.4[5]<5.7>已知上表中的参数，一个系统用了一半的内存来运行5个应用程序，计算该系统使用的页表总大小。**

**5.11.5[10]<5.7>已知上表中的参数，一个系统用了一半的内存来运行5个应用程序，给定个两级的有256项的页表，计算该系统使用的页表总大小。假设主页表中每项是6字节，计算所需的最小和最大的内存容量。**

**5.11.6[10]<5.7>一名cache设计者要将一个4KiB的虚拟索引、物理标记的cache的容量增大，已知页的大小在上表中列出，那么能否构建一个16KiB的直接映射cache,假设块大小为2个字?设计者如何增加cache的数据大小?**

解：

替换掉一个老页（其实这三个换哪一个都行）

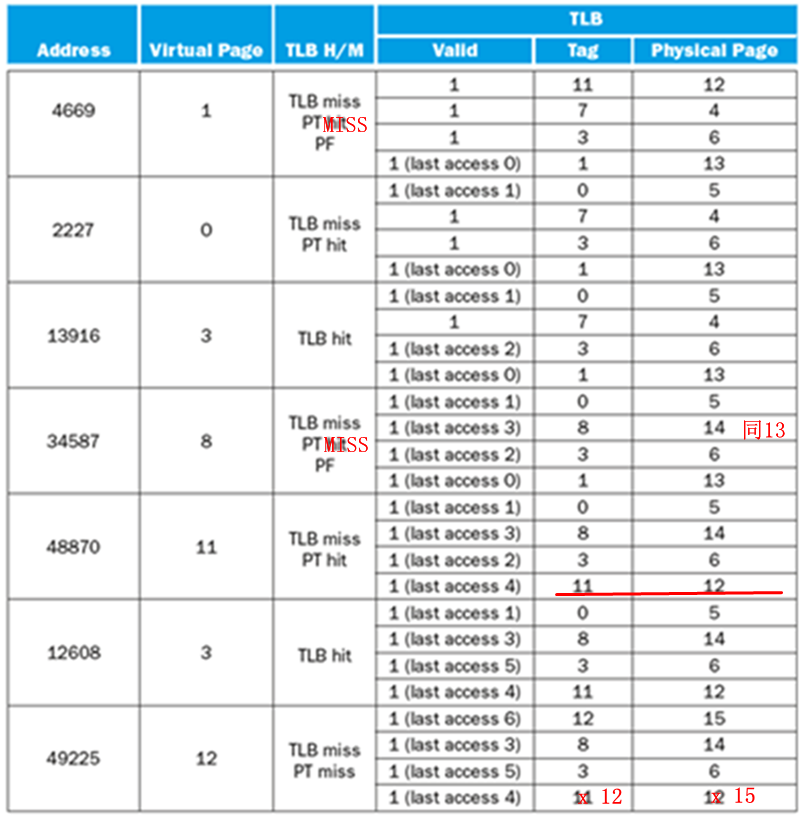
1. 页的大小4KiB = 4096B

Address整除4096得到虚页号 先查TLB再查PT

TLB快表无虚页1，所以miss

PT（页表中）虚页1在磁盘，所以miss，所以PF（Page Fault）缺页。同时TLB调入在最后的空位上。PT更新第二行（0，磁盘）-〉（0，13）

为何是13？因为原PT中最大虚页号是12，需一新页为13



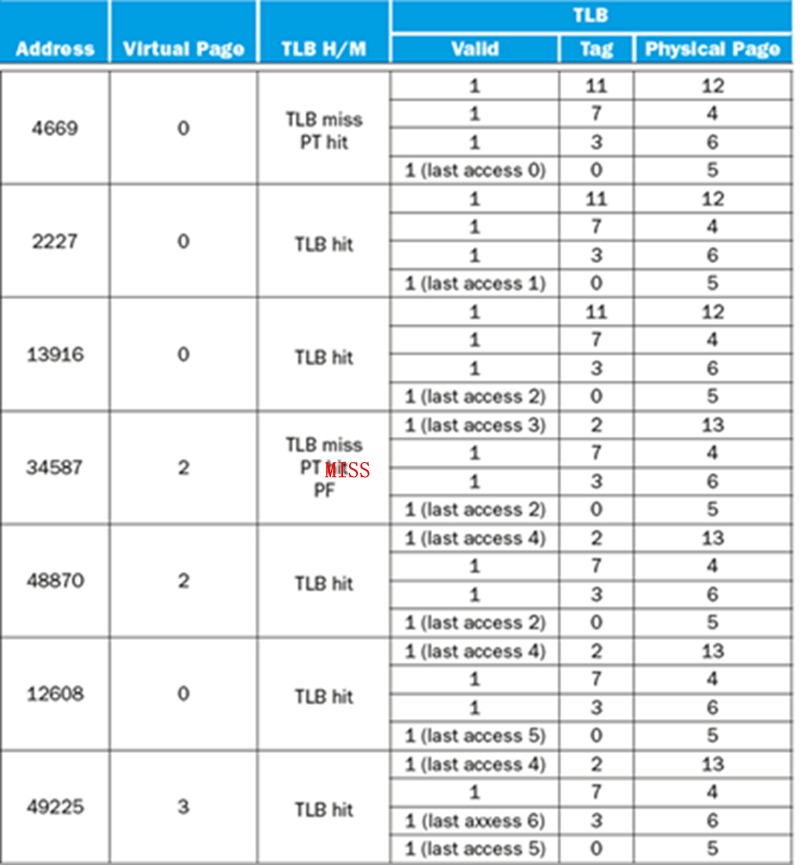
在页表中查得

从页表PT中查行，0虚页对应5实页（last access次序）记录谁老

页表最终状态

|  |  |
| --- | --- |
| 有效位 | 物理页/在磁盘上 |
| 1 | 5 |
| 1 | 13 |
| 0 | Disk |
| 1 | 6 |
| 1 | 9 |
| 1 | 11 |
| 0 | Disk |
| 1 | 4 |
| 1 | 14 |
| 0 | Disk |
| 1 | 3 |
| 1 | 12 |
| 1 | 15 |

（2）页的大小16KiB = 24\*210 B = 214 = 16384



第二页在PT中可见在磁盘中，所以要配一新页13。同时更新PT并用last access记录访问新老

第0页在PT中命中，对应5实页，写入TLB，以便下次TLB命中不必访存查PT

使用更大的页减少TLB缺失率，但是会导致物理存储器更高的碎片化和更低的利用率。

页表最终状态

|  |  |
| --- | --- |
| 有效位 | 物理页/在磁盘上 |
| 1 | 5 |
| 0 | Disk |
| 1 | 3 |
| 1 | 6 |
| 1 | 9 |
| 1 | 11 |
| 0 | Disk |
| 1 | 4 |
| 0 | Disk |
| 0 | Disk |
| 1 | 3 |
| 1 | 12 |