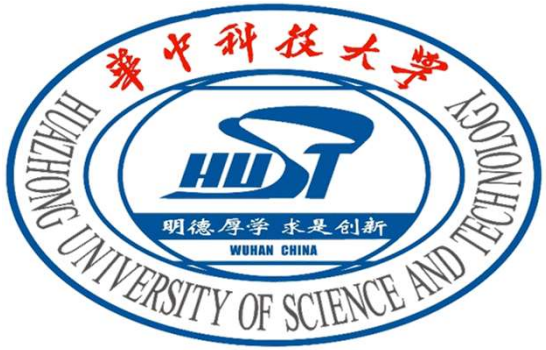
微机原理与接口技术

高速缓存管理策略

华中科技大学　左冬红



读策略

- 命中
  - 高速缓存提供数据给CPU
- 未命中-行填充
  - Cache控制器将内存中包含该地址的一个完整行拷贝到Cache中
  - 之后再由高速缓存提供数据给CPU

根据给定内存地址以及Cache结构，如何确定一个完整行的首、末地址？

行填充

内存物理地址

直接映射

内存行号：$\frac{\text{内存地址}}{\text{行大小}}$

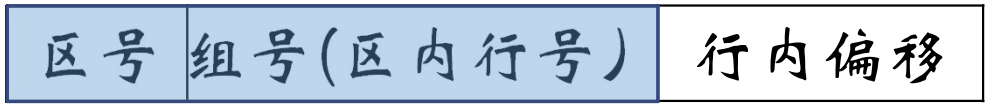
内存物理地址

全相联

行首地址：行内偏移全0

内存物理地址

组相联



行末地址：行内偏移全1

替换策略

当缓存内行已填充满，新的行需要进入时，将替换已存在的行

直接映射无选择

1.随机替换

随机选择一行替换，电路实现简单

路数越多，两种策略性能基本一致

2.最近最少使用（LRU）替换

替换较长时间内没有访问的行，电路实现复杂

缓存控制器需定时设置行访问标志，定时器自动清除访问标志为0，若命中则缓存控制器设置访问标志为1，替换标志为0的行

例题

已知某Cache结构为12行×8字节/行，CPU连续访问以下内存地址的数据时，试描述采用直接映射、全相联、3路组相联映射策略下，Cache最后的存储映像

21, 166, 201, 143, 61, 166, 62, 133, 111, 143, 144, 61

地址	21	166	201	143	61	166	62	133	111	143	144	61
内存行	2	20	25	17	7		7	16	13		18	

内存行号：$\frac{\text{内存地址}}{\text{行大小}}$

例题

Cache结构为12行×8字节/行

地址	21	166	201	143	61	166	62	133	111	143	144	61
内存行	2	20	25	17	7	20	7	16	13	17	18	7
缓存行	2	8	1	5	7	8	7	4	1	5	6	7

直接映射

缓存行号：　内存行号%缓存行数

Tag标志：　内存行号/缓存行数

	V	Tag	数据
0			
1	1	1	Mem[104..111]
2	1	0	Mem[16..23]
3			
4	1	1	Mem[128..135]
5	1	1	Mem[136..143]
6	1	1	Mem[144..151]
7	1	0	Mem[56..63]
8	1	1	Mem[160..167]
9			
10			
11			

例题

Cache结构为12行×8字节/行

地址	21	166	201	143	61	166	62	133	111	143	144	61
内存行	2	20	25	17	7	20	7	16	13	17	18	7

全相联

Tag标志：　内存行号

V	Tag	数据
1	2	Mem[16..23]
1	20	Mem[160..167]
1	25	Mem[200..207]
1	17	Mem[136..143]
1	7	Mem[56..63]
1	16	Mem[128..135]
1	13	Mem[104..111]
1	18	Mem[144..151]

例题

Cache结构为12行×8字节/行

地址	21	166	201	143	61	166	62	133	111	143	144	61
内存行	2	20	25	17	7	20	7	16	13	17	18	7
缓存行	2	0	1	1	3	0	3	0	1	1	2	3

3路组相联映射　每区行数=12/3=4行

缓存行号：

内存行号%各区缓存行数

Tag标志：

内存行号/各区缓存行数

	V	Tag	数据
0	1	5	Mem[160..167]
1	1	6	Mem[200..207]
2	1	0	Mem[16..23]
3	1	1	Mem[56..63]
0	1	4	Mem[128..135]
1	1	4	Mem[136..143]
2	1	4	Mem[144..151]
3			
0			
1	1	3	Mem[104..111]
2			
3			

缓存数据存储率

缓存中可存储内存数据的容量与缓存总容量的比值

缓存每行结构都一样

每一行数据的存储率即为缓存数据存储率

例题

已知某32位计算机系统可访问的物理内存空间为4G，具有256KB高速缓存，缓存行大小为128B，试分析该高速缓存分别采用直接映射、全相联以及16路组相联映射策略下的行结构、总体结构以及数据存储率。

缓存行大小为128B，缓存总容量为256KB

缓存共有256KB/128B=2K行

内存物理地址32位

行内字节偏移的位数 $\log_2 128 = 7$

直接映射策略

- 行索引的位数 $\log_2 2K = 11$
- 标志的位数 32-7-11=14

例题

已知某32位计算机系统可访问的物理内存空间为4G，具有256KB高速缓存，缓存行大小为128B，试分析该高速缓存分别采用直接映射、全相联以及16路组相联映射策略下的行结构以及数据存储率。

缓存行大小为128B，缓存总容量为256KB

缓存共有256KB/128B=2K行

内存物理地址32位

行内字节偏移的位数为 $\log_2 128 = 7$

全相联映射策略
- 行索引的位数为0
- 标志的位数为32-7-0=25

例题

已知某32位计算机系统可访问的物理内存空间为4G，具有256KB高速缓存，缓存行大小为128B，试分析该高速缓存分别采用直接映射、全相联以及16路组相联映射策略下的行结构以及数据存储率。

缓存行大小为128B，缓存总容量为256KB

缓存共有256KB/128B=2K行

每路共有2K/16=128行

内存物理地址32位

行内字节偏移的位数为 $\log_2 128 = 7$

16路组映射策略

- 组索引的位数为 $\log_2 128 = 7$
- 标志的位数为32-7-7=18

例题

已知某32位计算机系统可访问的物理内存空间为4G，具有256KB高速缓存，缓存行大小为128B，试分析该高速缓存分别采用直接映射、全相联以及16路组相联映射策略下的行结构、总体结构以及数据存储率。

直接映射缓存行结构	V(1位)	标志(14位)	数据(128×8位)
全相联缓存行结构	V(1位)	标志(25位)	数据(128×8位)
16路组相联缓存行结构	V(1位)	标志(18位)	数据(128×8位)

直接映射策略数据存储率：$\frac{128\times8}{128\times8+1+14}=\frac{1024}{1039}$。

全相联映射策略数据存储率：$\frac{128\times8}{128\times8+1+25}=\frac{1024}{1050}$。

组相联映射策略数据存储率：$\frac{128\times8}{128\times8+1+18}=\frac{1024}{1043}$。

例题

已知一容量为32字节的缓存，分别采用以下两种结构：1）8行、每行4字节；2）4行、每行8字节。若程序段A、B，编译器采用列优先内存分配策略为数组a分配存储空间，试分析当M、N分别为16、2时，直接映射策略下两程序段在两种不同缓存的命中率。假设数据元素a[0][0]的地址为32的整数倍。

程序段A

```
sum-array-rows()
{
int i,j;
short a[M][N];
short sum;
for(i=0;i<M;i++)
        for(j=0;j<N;j++)
        sum=sum+a[i][j];
}
```

程序段B

```
sum-array-cols()
{
int i,j;
short a[M][N];
short sum;
for(j=0;j<N;j++)
        for(i=0;i<M;i++)
        sum=sum+a[i][j];
}
```

Cache与内存的行

Cache

8行、每行4字节

V	Tag	数据			

命中率 50%

程序段A

```
sum-array-rows()
{
int i,j;
short a[M][N];
short sum;
for(i=0;i<M;i++)
        for(j=0;j<N;j++)
        sum=sum+a[i][j];
}
```

a[0][0]	a[0][1]
a[1][0]	a[1][1]
a[2][0]	a[2][1]
a[3][0]	a[3][1]
a[4][0]	a[4][1]
a[5][0]	a[5][1]
a[6][0]	a[6][1]
a[7][0]	a[7][1]
a[8][0]	a[8][1]
a[9][0]	a[9][1]
a[10][0]	a[10][1]
a[11][0]	a[11][1]
a[12][0]	a[12][1]
a[13][0]	a[13][1]
a[14][0]	a[14][1]
a[15][0]	a[15][1]

内存

Cache与内存的行

Cache

8行、每行4字节

V	Tag	数据			

命中率 0%

程序段B

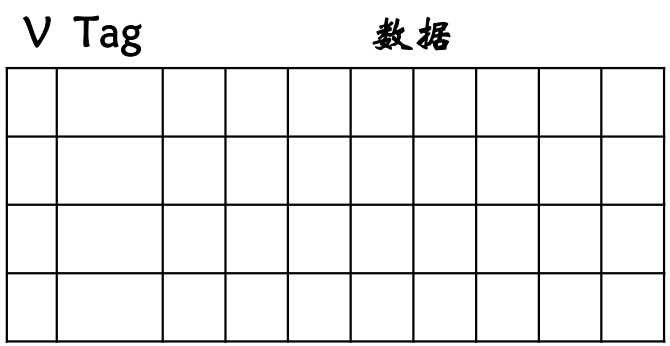
```
sum-array-cols()
{
int i,j;
short a[M][N];
short sum;
for(j=0;j<N;j++)
        for(i=0;i<M;i++)
        sum=sum+a[i][j];
}
```

内存	
a[0][0]	a[0][1]
a[1][0]	a[1][1]
a[2][0]	a[2][1]
a[3][0]	a[3][1]
a[4][0]	a[4][1]
a[5][0]	a[5][1]
a[6][0]	a[6][1]
a[7][0]	a[7][1]
a[8][0]	a[8][1]
a[9][0]	a[9][1]
a[10][0]	a[10][1]
a[11][0]	a[11][1]
a[12][0]	a[12][1]
a[13][0]	a[13][1]
a[14][0]	a[14][1]
a[15][0]	a[15][1]

Cache与内存的行

Cache

4行、每行8字节



程序段A

```
sum-array-rows()
{
int i,j;
short a[M][N];
short sum;
for(i=0;i<M;i++)
        for(j=0;j<N;j++)
        sum=sum+a[i][j];
}
```

a[0][0]	a[0][1]	a[1][0]	a[1][1]
a[2][0]	a[2][1]	a[3][0]	a[3][1]
a[4][0]	a[4][1]	a[5][0]	a[5][1]
a[6][0]	a[6][1]	a[7][0]	a[7][1]
a[8][0]	a[8][1]	a[9][0]	a[9][1]
a[10][0]	a[10][1]	a[11][0]	a[11][1]
a[12][0]	a[12][1]	a[13][0]	a[13][1]
a[14][0]	a[14][1]	a[15][0]	a[15][1]

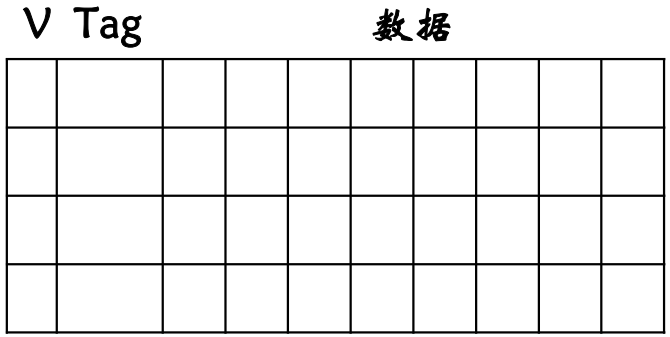
内存

命中率 75%

Cache与内存的行

Cache

4行、每行8字节



a[0][0]	a[0][1]	a[1][0]	a[1][1]
a[2][0]	a[2][1]	a[3][0]	a[3][1]
a[4][0]	a[4][1]	a[5][0]	a[5][1]
a[6][0]	a[6][1]	a[7][0]	a[7][1]
a[8][0]	a[8][1]	a[9][0]	a[9][1]
a[10][0]	a[10][1]	a[11][0]	a[11][1]
a[12][0]	a[12][1]	a[13][0]	a[13][1]
a[14][0]	a[14][1]	a[15][0]	a[15][1]

程序段B

```
sum-array-cols()
{
int i,j;
short a[M][N];
short sum;
for(j=0;j<N;j++)
        for(i=0;i<M;i++)
        sum=sum+a[i][j];
}
```

命中率 50%

内存

缓存行大小变大，可以一定程度上提高命中率

缓存行大小变大到一定程度后，不再提高命中率，反而增大惩罚时间

缓存写策略

- 命中
  - 透写（Write through）
    - 既写缓存，也写内存
  - 回写（Write back）
    - 仅写缓存，当需替换时再写入内存
- 未命中
  - 配写（Write Allocate）
    - 写内存后，再拷贝到缓存
  - 不配写（Write No Allocate）
    - 仅写内存

小结

- 缓存管理策略
  - 映射策略
  - 读策略
  - 替换策略
  - 写策略