

# 计算机系统基础第五次作业

## 201300035 方盛俊

《计算机系统基础》教材第2版第6章后习题中的第3、5、9、10、12、13、18、21、23题作业

### 3.

#### (1)

因为  $512\text{MB} / (64\text{M} \times 8\text{bit}) = 8$  片, 因此每个内存条需要 8 片 DRAM 芯片.

#### (2)

因为  $2\text{GB} / 512\text{MB} = 4$  个, 因此主存需要 4 个内存条.

#### (3)

对于按字节编址 4GB 寻址空间的主存, 我们可知主存地址有 32 位, 即  $A_{31}A_{30} \cdots A_1A_0$

因为一个内存条大小为 512MB, 因此需要 29 位地址, 即芯片内地址为  $A_{28}A_{27} \cdots A_1A_0$

并且  $A_{28}A_{27} \cdots A_{16}$  为行地址,  $A_{15}A_{14} \cdots A_3$  为列地址,  $A_2A_1A_0$  用于选择芯片.

### 5.

磁盘旋转一圈的时间:  $(60 \times 1000 \text{ ms}) / 7200 \text{ r/m} = 8.33 \text{ ms}$

平均旋转等待时间:  $8.33 \text{ ms} / 2 = 4.17 \text{ ms}$

数据块传输时间:  $4\text{KB} / 40 \text{ MB/s} = 4 \times 2^{10} \text{ B} / (40 \times 10^6) \text{ B/s} = 0.1024 \text{ ms}$

数据块平均读取写回时间:  $2 \text{ ms} + 10 \text{ ms} + 4.17 \text{ ms} + 0.1024 \text{ ms} = 16.27 \text{ ms}$

数据块处理时间:  $20000 / 500 \text{ MHz} = 0.04 \text{ ms}$

读出处理写回时间:  $2 \times 16.27 \text{ ms} + 0.04 \text{ ms} = 32.58 \text{ ms}$

可以完成次数:  $1000 \text{ ms} / 32.58 \text{ ms} = 30$  次

## 9.

### (1)

我们用 -1 代表 cache 有效位置为 0, 代表缺失.

列表格如下:

数据地址	2	3	11	16	21	13	64	48
cache 行号	2	3	11	0	5	13	0	0
cache 主存块	-1	-1	-1	-1	-1	-1	16	64
是否命中	miss	miss	miss	miss	miss	miss	miss	miss

数据地址	19	11	3	22	4	27	6	11
cache 行号	3	11	3	6	4	11	6	11
cache 主存块	3	11	19	-1	-1	11	22	27
是否命中	miss	hit	miss	miss	miss	miss	miss	miss

可以看出, 16 次查询仅有 1 此命中, 命中率为  $1 / 16 = 6.25\%$

### (2)

将主存块换为 4 个字后, 数据区容量不变, 则 cache 变为 4 行.

数据地址	2	3	11	16	21	13	64	48
数据主存块	0	0	2	4	5	3	16	12
cache 行号	0	0	2	0	1	3	0	0
cache 主存块	-1	0	-1	0	-1	-1	4	16
是否命中	miss	hit	miss	miss	miss	miss	miss	miss

数据地址	19	11	3	22	4	27	6	11
数据主存块	4	2	0	5	1	6	1	2
cache 行号	0	2	0	1	1	2	1	2

数据地址	19	11	3	22	4	27	6	11
cache 主存块	12	2	4	5	5	2	1	6
是否命中	miss	hit	miss	hit	miss	miss	hit	miss

可以看出, 16 次命中了 4 次, 命中率提高为  $4 / 16 = 25\%$

## 10.

例子如下:

```
int sum_array(int a[N][N][N]) {
    int i, j, k, sum = 0;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            for (k = 0; k < N; k++) {
                sum += a[i][j][k];
            }
        }
    }
    return sum;
}
```

如果数组元素的访问与排列顺序一致, 该数组连续的那一部分数据也容易被装载在同一个数据块中, 可以直接从 cache 中读取, 不用频繁地访问内存, 命中率更高, 执行时间也就更短.

## 12.

### (1)

数组 `x` 和 `y` 均为按顺序访问的, 因此空间局部性好; 但数组每一个元素仅被访问一次, 因此没有时间局部性.

因为没有给出 cache 相关的信息, 只能根据空间局部性判断, 在大部分情况下命中率是高的, 但是并不能给出具体情况.

### (2)

cache 有  $32\text{ B} / 16\text{ B} = 2$  行, 并且我们知道, 每 4 个 float 一共 16 B 的数据作为一个数据块, 则 `x[0] ~ x[3]`, `x[4] ~ x[7]`, `y[0] ~ y[3]`, `y[4] ~ y[7]` 则为 4 个不同的主存块.

可以看出,  $x[0] \sim x[3]$ ,  $y[0] \sim y[3]$  会被加载到 cache 的同一行,  $x[4] \sim x[7]$ ,  $y[4] \sim y[7]$  会被加载到 cache 的另一行. 所以, 在每一次执行  $sum += x[i] * y[i]$  的时候,  $x[i]$  和  $y[i]$  会互相冲刷, 导致每一次都不命中, 即命中率为 0 %

### (3)

如果改为 2 路组相联映射, 主存块改为 8 B, 则 cache 变为 2 组, 每组 2 行. 因此  $x[0] \sim x[1]$ ,  $x[2] \sim x[3]$  等等依次分为 4 个主存块, 同理  $y$  也依次分为了 4 个主存块.

因为每组有两行, 因此  $x[i] \sim x[i+1]$  和  $y[i] \sim y[i+1]$  可以放入同一个组内的不同两行, 所以每两个元素, 就能有一个元素命中, 最后的命中率为 50%

### (4)

这样就会变为  $x[0] \sim x[3]$ ,  $x[4] \sim x[7]$ ,  $x[8] \sim x[11]$ ,  $y[0] \sim y[3]$ ,  $y[4] \sim y[7]$ ,  $y[8] \sim y[11]$  六个主存块, 此时就不会发生  $x[i]$  和  $y[i]$  互相冲刷的情况.

在类似  $x[0]$  访问会 miss 后,  $x[1] \sim x[3]$  均能 hit, 即命中率为 75%

## 13.

由块大小为 16 B 知每一个内存块包含 4 个 int 数组元素.

数据区容量为 32 B 时, cache 只有 2 行:

	src 数组				dst 数组			
	col = 0	col = 1	col = 2	col = 3	col = 0	col = 1	col = 2	col = 3
row = 0	0 / miss	0 / miss	0 / hit	0 / miss	0 / miss	0 / miss	0 / miss	0 / miss
row = 1	1 / miss	1 / hit	1 / miss	1 / hit	1 / miss	1 / miss	1 / miss	1 / miss
row = 2	0 / miss	0 / miss	0 / hit	0 / miss	0 / miss	0 / miss	0 / miss	0 / miss
row = 3	1 / miss	1 / hit	1 / miss	1 / hit	1 / miss	1 / miss	1 / miss	1 / miss

数据区容量为 128 B 时, cache 有 8 行:

	src 数组				dst 数组			
	col = 0	col = 1	col = 2	col = 3	col = 0	col = 1	col = 2	col = 3
row = 0	4 / miss	4 / hit	4 / hit	4 / hit	0 / miss	0 / hit	0 / hit	0 / hit
row = 1	5 / miss	5 / hit	5 / hit	5 / hit	1 / miss	1 / hit	1 / hit	1 / hit
row = 2	6 / miss	6 / hit	6 / hit	6 / hit	2 / miss	2 / hit	2 / hit	2 / hit
row = 3	7 / miss	7 / hit	7 / hit	7 / hit	3 / miss	3 / hit	3 / hit	3 / hit

## 18.

设运行  $N$  条指令.

对于处理器 1, 额外开销为  $(4\% + 0.5 \times 6\%) \times (1 + 6) \times N = 0.49 N$  个时钟周期, 执行时间为  $(2.0 N + 0.49 N) \times 420 \text{ ps} = 1045.8 N \text{ ps}$

对于处理器 2, 额外开销为  $(2\% + 0.5 \times 4\%) \times (4 + 6) \times N = 0.40 N$  个时钟周期, 执行时间为  $(2.0 N + 0.40 N) \times 420 \text{ ps} = 1008 N \text{ ps}$

对于处理器 3, 额外开销为  $(2\% + 0.5 \times 3\%) \times (4 + 6) \times N = 0.35 N$  个时钟周期, 执行时间为  $(2.0 N + 0.35 N) \times 450 \text{ ps} = 1057.5 N \text{ ps}$

因此处理器 1 的额外开销最大, 处理器 2 的执行速度最快.

## 21.

(1)

因为页大小为 128 B, 所以页内偏移量长度为  $\log 128 = 7$ , 即低 7 位为页内偏移量; 高 9 位为虚拟页号; 因为 TLB 一共有 4 组, 所以虚拟页号低 2 位为 TLB 组索引, 高 7 位为 TLB 标记.

(2)

物理地址高 5 位为物理页号, 低 7 位为页内偏移量.

### (3)

因为 cache 块大小为 4 B, 共 16 行, 所以低 2 位为块内地址, 中间 4 位为 cache 行索引, 高 6 位为标记.

### (4)

因为 067AH = 0000 0110 0111 1010B, 所以虚拟页号为 000001100B, TLB 组索引为 00B, 标记为 0000011B, 即 03H, 可惜在 TLB 中对应有有效位为 0, 那么 TLB 缺失.

接着查询页表, 查找虚拟页号 1100B = 00CH 处的页表项, 发现页框号为 19H = 11001B, 并且有效位为 1, 页表命中.

那么物理地址为 110011 1110 10B, 接着查询 cache 行索引 1110B = EH 处的内容, 发现有效位为 1, 且标记为 33H = 110011B, 因此 cache 命中.

最后我们根据低 2 位的块内地址 10B, 可以从 cache 中读出这个 short 变量值为 2D4AH.

## 23.

### (1)

指令序列为:

```
movl    $0, %ecx
.LOOP
cmpl    %ebx, %ecx
jge     .EXIT
addl    (%edx, %ecx, 4), %eax
incl    %ecx
jmp     .LOOP
.EXIT
```

### (2)

因为执行程序 P 时是保护模式和分页模式, 所以 PE = 1 (保护模式), PG = 1 (启用分页).

### (3)

汇编形式为 `addl (%edx, %ecx, 4), %eax`, 寻址方式为 "基址 + 比例变址 + 偏移量", 偏移量为 0, 比例因子为 4.

### (4)

因为是 IA-32 / Linux, 各个段的线性地址都是从基址 0 开始, 并且 DPL 为 3, 不会发生保护错误.

那么指令 I 的线性地址为  $0x0 + 0x8049c08 = 0x8049c08$ .

线性地址高 20 位 0000 1000 0000 0100 1001B 是虚页号.

虚页号的高 10 位 0000 1000 00B 为页目录索引, 低 10 位 00 0100 1001B 为页表索引, 线性地址低 12 位 1100 0000 1000B 为页内偏移量.

指令 I 页目录项的在内存中的地址为  $0x3d000 + 0x20 \times 4 = 0x3d080$ , 页表项的在内存中的地址为  $0x5c8000 + 0x49 \times 4 = 0x5c8124$ .

指令 I 在主存的  $0x1020\ 000 + 0xc08 = 0x1020c08$  地址处.

字段  $P = 1$  (所在页已经装入主存),  $R/W = 0$  (只能读, 不能写),  $U/S = 1$  (允许用户进程访问),  $A = 1$  (所在页已被访问过),  $D = 0$  (所在页为代码页, 不会被改变).

## (5)

获取指令 I 不会发生缺页异常, 因为在执行前面代码的过程中会将指令 I 调用内存; 但是读取操作数 `a[0]` 可能会发生缺页异常, 因为 `a[0]` 的地址  $0x8049300$  可能还没被加载到内存里.

若发生缺页异常, 页故障线性地址为  $0x8049000$ , 保存在 CR2 寄存器中.

## (6)

指令 I 在第一次执行过程中, 也不会发生 TLB 缺失, 因为指令 I 并不在页首. 但是取操作数 `a[0]` 可能会发生 TLB 缺失.

对于 16 个表项, 采用 4 路组相联, 说明有 4 个组, 则虚页号低 2 位为 TLB 组索引, 高 18 位为 TLB 标记.

因为指令 I 的虚页号为 00 0010 0000 0001 0010 01B, 所以是第 1 组中的 02012H 对应的页表项, 页框号为 028B0H, 最后主存地址为  $0x28b0c08$ .

## (7)

因为 cache 数据区容量为 8KB, 内存大小块为 32B, 所以 cache 共有  $8KB / 32B = 256$  行. 因为采用 2 路组相联, 所以 cache 共有 128 组, 主存地址被划分为: 高 20 位的标记, 中间 7 位的组索引, 还有低 5 位的块内地址.

指令 I 的线性地址为  $0x8049c08$ , 其中低 12 位 1100 0000 1000B 为页内偏移量, 因此组索引为 1100000, 块内地址为 01000, 并不在一个主存块的起始位置, 所以在执行它之前的指

令, 就会将其加载进了 cache 中, 不会发生 cache 缺失, 同时会被映射到  $1100000B = 60H$  组中.

**(8)**

当  $N = 2000$  时, 占用空间大小为  $4 \times 2000 = 8000$  字节,  $a$  的首地址为  $0x8049300$ , 不在页首, 占用页面为  $8000B / 4KB = 1.953125$ , 最后大概占用了三个页面.

每个页的虚页号分别是  $0000\ 1000\ 0000\ 0100\ 1001B$ ,  $0000\ 1000\ 0000\ 0100\ 1010B$  和  $0000\ 1000\ 0000\ 0100\ 1010B$ .

数组元素  $a[1200]$  对应的  $4 \times 1200 = 4800 > 4096$ , 因此位于第二个页面中.