第五章　大而快：层次化存储

5.2 cache 对于为处理器提供高性能存储器层次结构非常重要。下面是 64 位存储器地址访问顺序列表，以字地址的形式给出。

0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5, 0x2c, 0xba, 0xbd

5.2.1 [10]<5.3> 对于一个有 16 个 cache 块、每个块大小为 1 个字的 cache，标识这些引用的二进制字地址、标签和索引。此外，假设 cache 最初为空，列出每个地址的访问会命中还是失效。

缓存块有 16 个，也就是 2^4，所以索引的大小为 4 个位，给出的地址为两个十六进制位，即 8 位二进制，所以索引实际上就是低 4 位地址，标签实际上是就是高 4 位地址，是否命中只需要看在某一行之前是否出现过相同的标签和索引。

	Address	Tag	Index	Hit
0x03	0000 0011	0000	0011	×
0xb4	1011 0100	1011	0100	×
0x2b	0010 1011	0010	1011	×
0x02	0000 0010	0000	0010	×
0xbf	1011 1111	1011	1111	×
0x58	0101 1000	0101	1000	×
0xbe	1011 1110	1011	1110	×
0x0e	0000 1110	0000	1110	×
0xb5	1011 0101	1011	0101	×
0x2c	0010 1100	0010	1100	×
0xba	1011 1010	1011	1010	×
0xfd	1111 1101	1111	1101	×

5.2.2 [10]<5.3> 对于一个有 8 个 cache 块、每个块大小为 2 个字的 cache，标识这些引用的二进制字地址、标签、索引和偏移。此外，假设 cache 最初为空，列出每个地址的访问会命中还是失效。

每个缓存块的大小变大了，缓存块的数量变小了，所以偏移实际上就是把索引的最

低位移过来了。同样是否命中只需要看在某一行之前是否出现过相同的标签和索引，并且在这之后没有索引相同但标签不同的访问（不然就被置换了），不需要考虑偏移。这里的命中一列，不命中使用 × 表示，而命中使用 ○ 表示（用 ○ 比 √ 从视觉上更好区分）。

	Address	Tag	Index	Offset	Hit
0x03	0000 0011	0000	001	1	×
0xb4	1011 0100	1011	010	0	×
0x2b	0010 1011	0010	101	1	×
0x02	0000 0010	0000	001	0	○
0xbf	1011 1111	1011	111	1	×
0x58	0101 1000	0101	100	0	×
0xbe	1011 1110	1011	111	0	○
0x0e	0000 1110	0000	111	0	×
0xb5	1011 0101	1011	010	1	○
0x2c	0010 1100	0010	110	0	×
0xba	1011 1010	1011	101	0	×
0xfd	1111 1101	1111	110	1	×

5.2.3 [20]<5.3,5.4> 请针对给定的访问顺序优化 cache 设计。这里有三种直接映射 cache 的设计方案, 每种方案都可以容纳 8 个字的数据:

- C1 的块大小为 1 个字
- C2 的块大小为 2 个字
- C3 的块大小为 4 个字

	Tag	Index / Offset	Hit C1	Hit C2	Hit C3
0x03	00000	011			
0xb4	10110	100			
0x2b	00101	011			
0x02	00000	010			
0xbf	10111	111			
0x58	01011	000			
0xbe	10111	110		√	√
0x0e	00001	110			
0xb5	10110	101		√	
0x2c	00101	100			
0xba	10111	010			
0xfd	11111	101			

可见 C2 的缓存命中次数最多，所以这里应该选择 C2 方案。

5.3 按照惯例, cache 以它包含的数据量来进行命名（例如，4KiB cache 可以容纳 4KiB 的数据），但是 cache 还需要 SRAM 来存储元数据，如标签和有效位等。本题研究 cache 的配置如何影响实现它所需的 SRAM 总量以及 cache 的性能。对所有的部分，都假设 cache 是字节可寻址的，并且地址和字都是 64 位的。

5.3.1 [10]<5.3> 计算实现每个块大小为 2 个字的 32KiB cache 所需的总位数。

32KiB 即 $2^5 \times 2^{10} = 2^{15}$B，每个块大小为 2 个字即 $2 \times 64/8 = 2^4$B，由于字节寻址所以偏移为 4 位，共有 $2^{15-4} = 2^{11}$ 个块，因此索引为 11 位，那么地址为 64 位，所以标签就是 $64 - 4 - 11 = 49$ 位，有效位为 1 位。每个块都有一个标签和一个有效位，所以 SRAM 存储的元数据需要 $2^{11} \times (49 + 1) = 102400$ 位。再加上存储数据的 2^{15+3} 位，即$\mathbf{2^{11} \times (49 + 1) + 2^{15+3} = 364544}$ **位**。

5.3.2 [10]<5.3> 计算实现每个块大小为 16 个字的 64KiB cache 所需的总位数。这个 cache 比 5.3.1 中描述的 32KiB cache 大多少?(请注意，通过更改块大小，我们将数据量增加了一倍，但并不是将 cache 的总大小加倍。)

64KiB 即 $2^6 \times 2^{10} = 2^{16}$B，每个块的大小为 16 个字即 $16 \times 64/8 = 2^7$B，所以偏移为 7 位。

共有 $2^{16-7} = 2^9$ 个块，索引为 9 位。 标签为 $64 - 7 - 9 = 48$ 位，有效位为 1 位。

所以需要的总位数为 $2^9 \times (48+1) + 2^{16+3} = \mathbf{549376}$ 位，比 5.3.1 大了 $\frac{549376-364544}{364544} = 0.507022471910112$，即约大了 $\mathbf{50.70}\%$。

5.3.3 [5]<5.3> 解释为什么 5.3.2 中的 64KiB cache 尽管数据量比较大，但是可能会提供比 5.3.1 中的 cache 更慢的性能。

5.3.2 中的 64KiB cache 每个块的大小比 5.3.1 大，所以每次缓存失效时都需要把更多的数据读入缓存或写回存储，所以失效代价会更大。而且总块数更少，所以命中率会更低。

5.3.4 [10]<5.3,5.4> 生成一系列读请求，这些请求需要在 32KiB 的两路组相联 cache 上的失效率低于在 5.3.1 中描述的 cache 的失效率。

在 5.3.1 中标签 49 位，索引 11 位，偏移 4 位。若改成两路组相联，那么每个块就只有 1 个字了，且索引为 10 位，标签为 50 位，每个块都各自有一个标签和一个有效位。所以只需要构造索引相同的地址，但是标签不同，即可让 5.3.1 的直接映射失效，而两路组相联不失效。所以构造的读请求地址如下：**0x00000, 0x10000, 0x00000, 0x10000, 0x00000, 0x10000, ……**，前两次请求都未命中缓存，但从第三次请求开始，直接映射会一直失效，而两路组相联会一直命中，所以符合要求。

5.5 对一个 64 位地址的直接映射 cache 的设计，地址的以下位用于访问 cache。

标签	索引	偏移
63 ~ 10	9 ~ 5	4 ~ 0

5.5.1 [5]<5.3>cache 块大小为多少（以字为单位）？

偏移是 0 到 4，也就是 5 位，所以 cache 块的大小是 $2^5 = 32$B，一个字是 4B，所以 cache 块的大小是 **8** 个字。

5.5.2 [5]<5.3 >cache 块有多少个?

索引从 5 到 9，也就是 5 位，所以 cache 块有 $2^5 = \mathbf{32}$ 个。

5.5.3 [5]<5.3> 这种 cache 实现所需的总位数与数据存储位之间的比率是多少?

标签是 10 到 63，也就是 54 位，还需要一位有效位，所以实现所需的总位数是 $2^5 \times (54 + 1) + 2^5 \times 2^5 \times 8 = 9952$，数据存储位是 $2^5 \times 2^5 \times 8 = 8192$。所以这种 cache 实现所需的总位数与数据存储位之间的比率是 $\frac{9952}{8192} \times 100\% = \mathbf{121.484375}\%$。

5.5.4 下表记录了从上电开始 cache 访问的字节地址。

十六进制	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
十进制	0	4	16	132	232	160	1024	30	140	3100	180	2180

[20]<5.3> 对每一次访问，列出：它的标签、索引和偏移；指出命中还是失效；替换了哪个字节 (如果有的话)。

Address	Tag	Index	Offset	Hit	Sub
00		0 0 0	0 0 0 0 0	×	
04		0 0 0	0 0 1 0 0	√	
10		0 0 0	1 0 0 0 0	√	
84		1 0 0	0 0 1 0 0	×	
E8		1 1 1	0 1 0 0 0	×	
A0		1 0 1	0 0 0 0 0	×	
400	0 1	0 0	0 0 0	0 0 0 0 0	×	00~1F
1E			0 0 0	1 1 1 1 0	×	400~41F
8C			1 0 0	0 1 1 0 0	√	
C1C	1 1	0 0	0 0 0	1 1 1 0 0	×	00~1F
B4			1 0 1	1 0 1 0 0	√	
884	1 0	0 0	1 0 0	0 0 1 0 0	×	80~9F

图中的 Address 表示十六进制的地址，Tag 为标签,，Index 为索引，Offset 为偏移，Hit 列中勾表示命中，叉表示失效；Sub 表示替换的字节地址范围。

5.5.5 [5]<5.3> 命中率是多少?

12 次访问中有 4 次命中，所以命中率为 $\frac{1}{3} \approx \mathbf{33.33}\%$。

5.5.6 [5]<5.3> 列出 cache 的最终状态，每个有效表项表示为 < 索引，标签，数据 > 的记录。例如:

```
<0, 3, Mem[0xC00]-Mem[0xC1F]
```

对于某个索引，从后往前看 Index，首次找到这个索引的地方所在的标签即为最终状态的标签，之后把偏移全填为 0，再把地址转成十六进制（已按 4 位一组用蓝色线分隔），即为起始地址；把偏移量全填为 1，再把地址转成十六机制，即为结束地址。

例如，对于 0 号索引，最后一次出现是在地址 C1C 的地方，它的标签为 11，也就是十六进制的 3。之后看这一行的地址为 1100 0001 1100，将偏移量全填成 0，也就是 1100 0000 0000，这就是 0xC00，将偏移量全填成 1，也就是 1100 0001 1111，这就是 0xC1F，所以结果为 `<0, 3, Mem[0xC00]-Mem[0xC1F]` 。

所以 cache 的最终状态如下:

```
<0, 3, Mem[0xC00]-Mem[0xC1F]>
<4, 2, Mem[0x880]-Mem[0x89F]>
<5, 0, Mem[0x0A0]-Mem[0x0BF]>
<7, 0, Mem[0x0E0]-Mem[0x0FF]>
```

5.7 考虑以下的程序和 cache 行为:

每 1000 条指令的数据读次数	每 1000 条指令的数据写次数	指令 cache 失效率	数据 cache 失效率	块大小（字节）
250	100	0.30%	2%	64

5.7.1 [10]<5.3,5.8> 假设一个带有写直达、写分配 cache 的 CPU 实现了 2 的 CPI，那么 RAM 和 cache 之间的读写带宽（用每个周期的字节数进行测量）是多少?(假设每个失效都会生成一个块的请求。)

CPI 为 2，每条指令 2 个周期，那么每个周期就是 0.5 条指令，那么指令的读带宽就是 $0.5 \times 0.30\% \times 64 = 0.096$ 字节/周期，而数据的读带宽就是 $0.5 \times \frac{250}{1000} \times 2\% \times 64 = 0.16$ 字节/周期。

由于写直达，所以不管是否失效，都需要将某个寄存器的数据写入到 RAM 中，RISC-V 中的寄存器为 64 位，也就是 8 个字节，那么写入 RAM 的带宽为 $0.5 \times \frac{100}{1000} \times 8 = 0.4$ 字节/周期。

由于写分配，所以当失效时，需要先（后 *）将 RAM 中的数据放回到寄存器中，这时会产生读带宽 $0.5 \times \frac{100}{1000} \times 2\% \times 64 = 0.064$ 字节/周期。

(* 如果先取回数据，那么需要同时写入 cache 和 RAM；如果后取回数据，那么就只写入 RAM，取回时就已经是最新的数据了。)

所以总的读带宽为 0.096+0.16+0.064 = **0.32**字节/周期，总的写带宽为 **0.4**字节/周期。

5.7.2 [10]<5.3,5.8> 对于一个写回、写分配 cache 来说，假设替换出的数据 cache 块中有 30% 是脏块，那么为了实现 CPI 为 2，读写带宽需要达到多少?

指令的读带宽仍为 0.096 字节/周期，数据的读带宽仍为 0.16 字节/周期。但是这里的“写分配”似乎没用？写的时候如果不失效，那 cache 中就是最新的；如果失效，那先替换旧的块，之后再写入 cache，也不会出现分配的情况。

对于写回策略，当写不失效时，不需要在 cache 与 RAM 中传输数据。当写失效时，如果被替换的 cache 块是“脏”块，也就是被修改过，那么需要将这个块写入到 RAM 中。当读失效时，仍然会产生 cache 块的替换，所以如果是“脏”块也需要写入 RAM。并且这里没有提及缓冲的事，所以认为没有写缓冲，那么写带宽为 $0.5\times\left(\frac{250}{1000}+\frac{100}{1000}\right)\times 2\%\times 30\%\times 64 = 0.0672$。

所以总的读带宽为 **0.096** 字节/周期，总的写带宽为 **0.0672** 字节/周期。

5.9 cache 块大小 (B) 可以影响失效率和失效延迟。假设一台机器的基本 CPI 为 1，每条指令的平均访问次数（包括指令和数据）为 1.35，给定以下各种不同 cache 块大小的失效率，找到能够最小化总失效延迟的 cache 块大小。

8:4%	16:3%	32:2%	64:1.5%	128:1%

5.9.1 [10]<5.3> 失效延迟为 20×B 周期时，最优块大小是多少?

总失效延迟为 $1.35\times$ 失效率 $\times 20\times B$ 周期，所以所求问题为

$$\arg\min_i \quad 1.35\times 失效率_i \times 20 \times B_i$$

$$\{(B_i, 失效率_i)\} = \{(8, 0.04), (16, 0.03), (32, 0.02), (64, 0.015), (128, 0.01)\}$$

枚举可得

B_i	失效率$_i$	总失效延迟
8	0.04	8.64
16	0.03	12.96
32	0.02	17.28
64	0.015	25.92
128	0.01	34.56

所以最优块大小是 8。

5.9.2 [10]<5.3> 失效延迟为 24+B 周期时，最优块大小是多少?

总失效延迟为 $1.35 \times$ 失效率 $\times (24 + B)$ 周期，所以所求问题为

$$\arg\min_i \quad 1.35 \times \text{失效率} \times (24 + B)$$

$$\{(B_i, \text{失效率}_i)\} = \{(8, 0.04), (16, 0.03), (32, 0.02), (64, 0.015), (128, 0.01)\}$$

枚举可得

B_i	失效率$_i$	总失效延迟
8	0.04	1.728
16	0.03	1.62
32	0.02	1.512
64	0.015	1.782
128	0.01	2.052

所以最优块大小是 32。

5.9.3 5.9.3[10]<5.3> 失效延迟为定值时，最优块大小是多少?

总失效延迟为 $1.35 \times$ 失效率 $\times$ 定值，所以失效率越小，总失效延迟就越小，**所以最优块大小是 128。**

5.10 本题研究不同 cache 容量对整体性能的影响。通常, cache 访问时间与 cache 容量成正比。假设主存访问需要 70ns，并且在所有指令中有 36% 的指令访问数据内存。下表显示了两个处理器 P1 和 P2 中每个处理器各自的 L1 cache 的数据。

	L1 大小	L1 失效率	L1 命中时间
P1	2 KiB	8.0%	0.66 ns
P2	4 KiB	6.0%	0.90 ns

5.10.1 [5]<5.4> 假设 L1 命中时间决定 Pl 和 P2 的时钟周期时间，它们各自的时钟频率是多少?

P1：$\frac{1}{0.66\times10^{-9}} \approx 1.515 \times 10^9\text{Hz} = 1.515\text{GHz}$。

P2：$\frac{1}{0.90\times10^{-9}} \approx 1.111 \times 10^9\text{Hz} = 1.111\text{GHz}$。

5.10.2 [10]<5.4>P1 和 P2 各自的 AMAT (平均内存访问时间) 是多少 (以周期为单位)?

这里的平均内存访问时间应该是每条指令的，而且是已知产生内存访问的情况下计算的，所以为

$$\text{P1:} \quad 1 + 8\% \times \left\lceil \frac{70}{0.66} \right\rceil = 9.56\text{周期}$$

$$\text{P2:}\quad 1+6\%\times\left\lceil\frac{70}{0.90}\right\rceil=5.68\text{周期}$$

5.10.3 [5]<5.4> 假设基本 CPI 为 1.0 而且没有任何内存停顿，那么 P1 和 P2 的总 CPI 是多少? 哪个处理器更快?(当我们说 “基本 CPI 为 1.0” 时，意思是指令在一个周期内完成，除非指令访问或者数据访问导致 cache 失效。)

这里计算的总 CPI 仍然是每条指令的周期数，但是并没有已知产生内存访问，所以

$$\text{P1:}\quad 1+8\%\times\left\lceil\frac{70}{0.66}\right\rceil+36\%\times8\%\times\left\lceil\frac{70}{0.66}\right\rceil=12.6416\text{周期/指令}$$

$$\text{P2:}\quad 1+6\%\times\left\lceil\frac{70}{0.90}\right\rceil+36\%\times6\%\times\left\lceil\frac{70}{0.90}\right\rceil=7.3648\text{周期/指令}$$

所以 P2 更快。

对于接下来的三个问题，我们将考虑向 P1 添加 L2 cache(可能弥补其有限的 Ll cache 容量)。解决这些问题时，请使用上一个表中的 Ll cache 容量和命中时间。L2 失效率表示的是其局部失效率。

L2 大小	L2 失效率	L2 命中时间
1 MiB	95%	5.62 ns

5.10.4 [10]<5.4> 添加 L2 cache 的 P1 的 AMAT 是多少? 在使用 L2 cache 后,AMAT 变得更好还是更差?

$$1+8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times95\%\times\left\lceil\frac{70}{0.66}\right\rceil=9.852\text{周期}$$

显然使用 L2 cache 后，AMAT 变得更差。

5.10.5 [5]<5.4> 假设基本 CPI 为 1.0 而且没有任何内存停顿，那么添加 L2 cache 的 P1 的总 CPI 是多少?

$$1+8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times95\%\times\left\lceil\frac{70}{0.66}\right\rceil+36\%\times\left(8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times95\%\times\left\lceil\frac{70}{0.66}\right\rceil\right)=13.03872$$

5.10.6 [10]<5.4> 为了使具有 L2 cache 的 P1 比没有 L2 cache 的 P1 更快，需要 L2 的失效率为多少?

设 L2 的失效率最大为 x，则

$$1+8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times x\times\left\lceil\frac{70}{0.66}\right\rceil=9.56$$

解得 $x=\frac{98}{107}\approx 0.9159=91.59\%$，所以 L2 的失效率需要小于 91.59%。

5.10.7 [15]<5.4> 为了使具有 L2 cache 的 P1 比没有 L2 cache 的 P2 更快，需要 L2 的失效率为多少?

设 L2 的失效率最大为 x，由于涉及到 P1 和 P2，所以这里需要比较的是实际的执行时间，即 CPI × 每条指令执行的时间，由于 5.10.1 的假设，所以每条指令执行的时间就是 L1 命中时间。

$$\left[1+8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times x\times\left\lceil\frac{70}{0.66}\right\rceil+36\%\times\left(8\%\times\left\lceil\frac{5.62}{0.66}\right\rceil+8\%\times x\times\left\lceil\frac{70}{0.66}\right\rceil\right)\right]\times 0.66$$
$$=7.3648\times 0.90$$

解得 $x=\frac{27719}{40018}\approx 0.6927=69.27\%$，所以 L2 的失效率需要小于 69.27%。