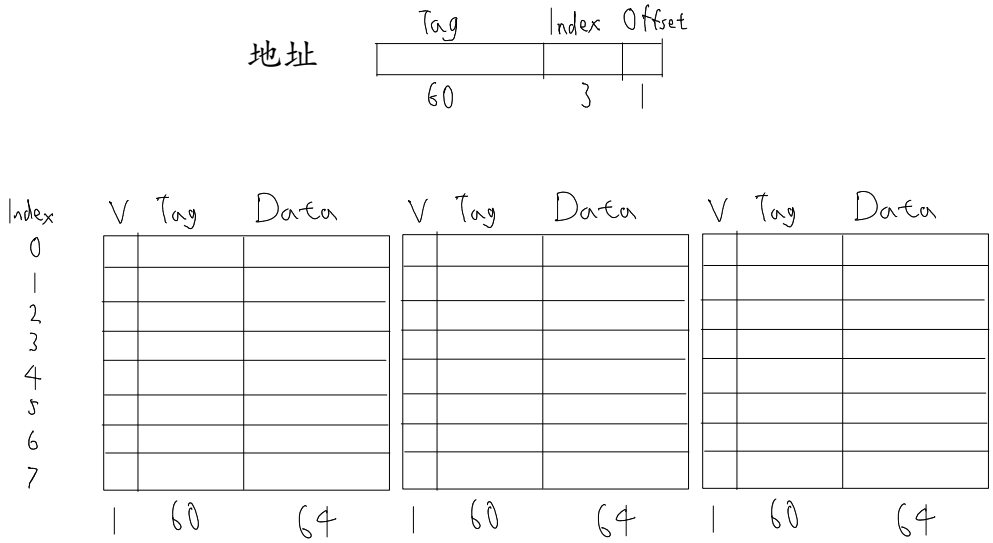
5.11 本题研究不同 cache 设计的效果，特别是将组相联 cache 与 5.4 节中的直接映射 cache 进行比较。有关这些练习，请参阅下面显示的字地址序列:

0x03, 0xb4, 0x2b, 0xbe, 0x58, 0xbf, 0x0e, 0x1f, 0xb5, 0xba, 0x2e, 0xce

5.11.1 [10]<5.4> 绘制块大小为 2 字、总容量为 48 字的三路组相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。



块大小为 $2 = 2^1$ 字，所以地址中的偏移量为 1 位。$48 \div 3 \div 2 = 8$，所以索引有 $8 = 2^3$ 行，所以地址中的索引有 3 位。所以标签为 $64 - 3 - 1 = \mathbf{60}$ 位。一个字应该是 32 位，并且块大小为 2 字，那么数据字段就是 $2 \times 32 = \mathbf{64}$ 位。

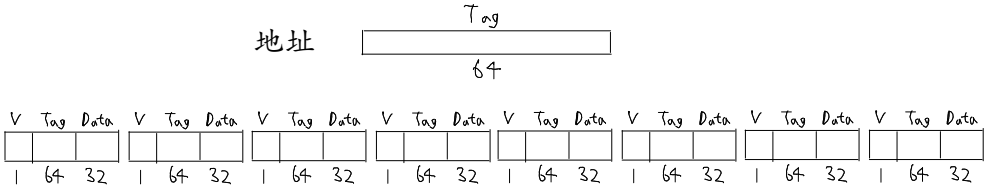
5.11.2 [10]<5.4> 从 5.11.1 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问, 确定:

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 每一路中有哪些标签。

Hexadecimal Address	Binary Address			Hit	Way 1	Way 2	Way 3
	Tag	Index	Offset		Index - Tag	Index - Tag	Index - Tag
03	0000	001	1	×	1-0		
b4	1011	010	0	×	1-0, 2-b		
2b	0011	101	1	×	1-0, 2-b, 5-2		
02	0000	001	0	○	1-0, 2-b, 5-2		
be	1011	111	0	×	1-0, 2-b, 5-2, 7-b		
58	0101	100	0	×	1-0, 2-b, 4-5, 5-2, 7-b		
bf	1011	111	1	○	1-0, 2-b, 4-5, 5-2, 7-b		
0e	0000	111	0	×	1-0, 2-b, 4-5, 5-2, 7-b	7-0	
1f	0001	111	0	×	1-0, 2-b, 4-5, 5-2, 7-b	7-0	7-1
b5	1011	010	1	○	1-0, 2-b, 4-5, 5-2, 7-b	7-0	7-1
bf	1011	111	1	○	1-0, 2-b, 4-5, 5-2, 7-b	7-0	7-1
ba	1011	101	0	×	1-0, 2-b, 4-5, 5-2, 7-b	5-b, 7-0	7-1
2e	0010	111	0	×	1-0, 2-b, 4-5, 5-2, 7-b	5-b, 7-2	7-1
ce	1100	111	0	×	1-0, 2-b, 4-5, 5-2, 7-b	5-b, 7-2	7-c

图中的 Hit（命中）列用蓝色圈表示命中，命中时在上一行会有蓝色框表示命中了哪个缓存。红色下划线表示在这一时刻产生了缓存替换，标记了新的缓存放在哪个位置。

5.11.3 [5]<5.4> 绘制块大小为 1 字、总容量为 8 字的全相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。



地址是按字索引的，块大小是 1 字，所以没有偏移量。由于是全相联，所以没有索引位，所以标签为 64 位。总容量为 8 字，所以 8 个块排成一行。数据字段就是块大小即 1 字 = 32 位。

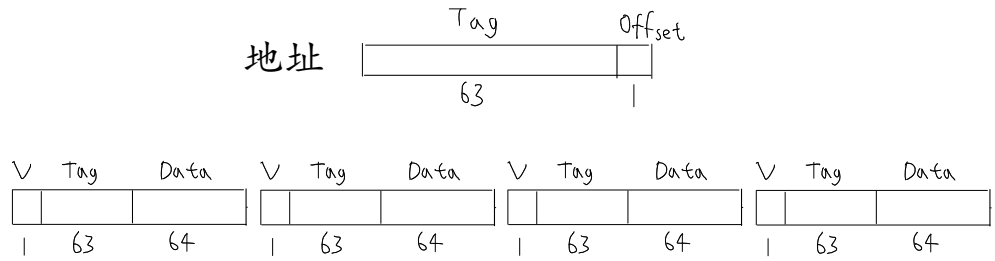
5.11.4 [10]<5.4> 从 5.11.3 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问，确定：

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 每一路中有哪些标签。

Hexadecimal Address	Binary Address		Way							
	Tag	Hit	0	1	2	3	4	5	6	7
03	00000011	×	03							
b4	10110100	×	03	b4						
2b	00101011	×	03	b4	2b					
02	00000010	×	03	b4	2b	02				
be	10111110	×	03	b4	2b	02	be			
58	01011000	×	03	b4	2b	02	be	58		
bf	10111111	×	03	b4	2b	02	be	58	bf	
0e	00001110	×	03	b4	2b	02	be	58	bf	0e
1f	00011111	×	1f	b4	2b	02	be	58	bf	0e
b5	10110101	×	1f	b5	2b	02	be	58	bf	0e
bf	10111111	○	1f	b5	2b	02	be	58	bf	0e
ba	10111010	×	1f	b5	ba	02	be	58	bf	0e
2e	00101110	×	1f	b5	ba	2e	be	58	bf	0e
ce	11001110	×	1f	b5	ba	2e	ce	58	bf	0e

图中的 Hit（命中）列用蓝色圈表示命中，命中时在上一行会有蓝色框表示命中了哪个缓存。红色下划线表示在这一时刻产生了缓存替换，标记了新的缓存放在哪个位置。

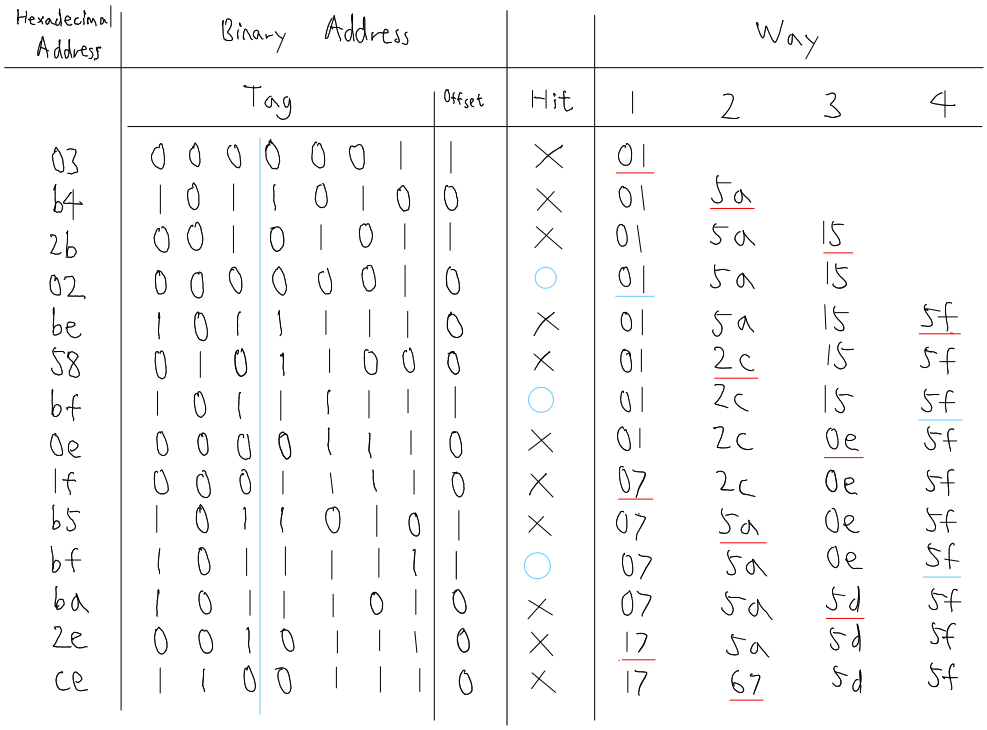
5.11.5 [5]<5.4> 绘制块大小为 2 字、总容量为 8 字的全相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。



块大小为 2 字，所以偏移量为 1 位，一个字是 32 位，所以数据大小为 64 位。总容量为 8 字，所以有 4 个块，全相联所以 4 个块排成一行，没有索引位，所以标签为 $64 - 1 = 63$ 位。

5.11.6 [10]<5.4> 从 5.11.5 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问，确定:

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 每一路中有哪些标签。



图中 Tag 列中的蓝色线表示 4 位分隔，便于二进制转十六进制。Hit（命中）列用蓝色圈表示命中，在命中时右侧的蓝色下划线表示命中了哪一路的标签，同时也代表产生了一次访问（相当于 LRU 把它放到链表最后）。在未命中时右侧的红色下划线表示替换了哪一路的标签，同时也代表产生了一次访问。

所以，对于每一行，填写方法是：先把 Tag 转成十六进制，再查看 Way（这里全相联就是四路组相联，Way 表示路）中是否有这个十六进制地址，如果有，表示命中，那就把上一行的 Way 复制下来，并在命中的地址上划一条蓝色下划线；如果没有，表示未命中，就从这行开始向上查看最近的哪条下划线距离最远（表示最久没访问），那么就替换这个地址，其他地址不变，替换后在这个地址上划一条红色下划线。

5.12 多级 cache 是一种重要的技术，可以在克服一级 cache 提供的有限空间的同时仍然保持速度。考虑具有以下参数的处理器:

无内存停顿的基本 CPI	处理器速度	主存访问时间	每条指令的 L1 cache 的失效率 *	L2 直接映射 cache 速度	L2 直接映射 cache 全局失效率	L2 八路组相联速度	L2 八路组相联 cache 全局失效率
1.5	2GHz	100ns	7%	12cycles	3.5%	28cycles	1.5%

*Ll cache 失效率是针对每条指令而言的。假设 Ll cache 的总失效数量（包括指令和数据）为总指令数的 7%。

这题的参考答案有误，答案当成局部失效率计算了。（全局失效率与局部失效率的定义在课本第 290 页）

5.12.1 [10]<5.4> 使用以下方法计算表中处理器的 CPI: 仅有 Ll cache; 使用 L2 直接映射 cache; 使用 L2 八路组相联 cache。如果主存访问时间加倍，这些数据会如何变化?(将每个更改作为绝对 CPI 和百分比更改。) 请注意 L2 cache 可以隐藏慢速内存影响的程度。

先计算主存访问的时钟周期，$2 \times 10^9\text{Hz} \times 100 \times 10^{-9}\text{s} = 200\text{cycles}$。

全局失效率是指访问 L2 并且 L2 失效的指令数量与全部指令数量的比值；局部失效率是指访问 L2 并且 L2 失效的指令数量与访问 L2 的指令的数量的比值。

- 仅有 L1 cache 时，CPI 为 $1.5 + 7\% \times 200 = \mathbf{15.5}$ 周期；
- 使用 L2 直接映射 cache 时，CPI 为 $1.5 + 7\% \times 12 + 3.5\% \times 200 = \mathbf{9.34}$ 周期；
- 使用 L2 八路组相联 cache 时，CPI 为 $1.5 + 7\% \times 28 + 1.5\% \times 200 = \mathbf{6.46}$ 周期。

如果主存访问时间加倍，

- 仅有 L1 cache 时，CPI 为 $1.5 + 7\% \times 400 = \mathbf{29.5}$ 周期，增加了 $29.5 - 15.5 = \mathbf{14}$ 个周期，增加了 $14/15.5 = \mathbf{90.3225806451613}\%$；
- 使用 L2 直接映射 cache 时，CPI 为 $1.5 + 7\% \times 12 + 3.5\% \times 400 = \mathbf{16.34}$ 周期，增加了 $16.34 - 9.34 = \mathbf{7}$ 个周期，增加了 $7/15.5 = \mathbf{45.1612903225806}\%$；
- 使用 L2 八路组相联 cache 时，CPI 为 $1.5 + 7\% \times 28 + 1.5\% \times 400 = \mathbf{9.46}$ 周期，增加了 $9.46 - 6.46 = \mathbf{3}$ 个周期，增加了 $3/6.46 = \mathbf{46.4396284829721}\%$。

5.12.2 [10]<5.4> 可能有比两级更多的 cache 层次结构吗? 已知上述处理器具有 L2 直接映射 cache, 设计人员希望添加一个 L3 cache，访问时间为 50 个时钟周期，并且该 cache 将具有 13% 的失效率。这会提供更好的性能吗? 一般来说，添加 L3 cache 有哪些优缺点?

可能有比两级更多的 cache 层次结构。这里的 13% 失效率没有说局部还全局。那么，如果它是全局失效率，那么肯定要比 L2 的全局失效率低，但是这里它比 L2 的全局失效率高，所以只能是局部失效率。

那么加入 L3 cache 后的 CPI 为：$1.5 + 7\% \times 12 + 3.5\% \times (50 + 13\% \times 200) = 5$ 周期 < 9.34 周期，所以**会**提供更好的性能。

添加 L3 cache 的优点是能用更小的全局失效率兜底，隐藏慢速内存影响的程度，减小总体的 CPI；缺点是一旦全部缓存都失效，必须访问主存时，会产生很大的延迟。

5.12.3 [20]<5.4> 在较老的处理器中，例如 Intel Pentium 或 Alpha 21264，L2 cache 在主处理器和 Ll cache 的外部 (位于不同芯片上)。虽然这种做法使得大型 L2 cache 成为可能，但是访问 cache 的延迟也变得很高，并且因为 L2 cache 以较低的频率运行，所以带宽通常也很低。假设 512KiB 的片外 L2 cache 的失效率为 4%, · 如果每增加一个额外的 512KiB cache 能够降低 0.7% 的失效率，并且 cache 的总访问时间为 50 个时钟周期，那么 cache 容量必须多大才能与上面列出的 L2 直接映射 cache 的性能相匹配?

这里 4% 的失效率应该为局部失效率。设有 $x + 1$ 个 512 KiB 的片外 L2 cache。那么

$$1.5 + 7\% \times 12 + 3.5\% \times 200 = 1.5 + 7\% \times (50 + (4\% - 0.7\%x) \times 200)$$

解得 $x = -\frac{270}{7} = -38.5714285714286$，但 x 应 $\geqslant 0$，所以**不存在合适的 cache 容量与上面列出的 L2 直接映射 cache 的性能相匹配。**

5.16 如 5.7 节所述，虚拟内存使用页表来跟踪虚拟地址到物理地址的映射。本题显示了在访问地址时必须如何更新页表。以下数据构成了在系统上看到的虚拟字节地址流。假设有 4KiB 页，一个 4 表项全相联的 TLB，使用严格的 LRU 替换策略。如果必须从磁盘中取回页，请增加下一次能取的最大页码:

十进制	4669	2227	13916	34587	48870	12608	49225
十六进制	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

有效位	标签	物理页号	上次访问时间间隔
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

页表

索引	有效位	物理页号/在磁盘中
0	1	5
1	0	在磁盘中
2	0	在磁盘中
3	1	6
4	1	9
5	1	11
6	0	在磁盘中
7	1	4
8	0	在磁盘中
9	0	在磁盘中
a	1	3
b	1	12

5.16.1 [10]<5.7> 对于上述每一次访问，列出:

- 本次访问在 TLB 会命中还是失效。
- 本次访问在页表中会命中还是失效。
- 本次访问是否会造成缺页错误。
- TLB 的更新状态。

4KiB 即 2^{12}Bytes，所以对于一个十六进制地址，右侧三位十六进制表示页内偏移，左侧一位表示标签。

Address		TLB	PT	Page				
Tag	Offset	Hit	Hit	Fault	V	Tag	Physical Page Number	Last Access Interval
1	2 3 d	×	√	√	1	b	12	5
					1	7	4	2
					1	3	6	4
					1	1	13	0
0	8 b 3	×	√	×	1	0	5	0
					1	7	4	3
					1	3	6	5
					1	1	13	1
3	6 5 c	√	√	×	1	0	5	1
					1	7	4	4
					1	3	6	0
					1	1	13	2
8	7 1 b	×	√	√	1	0	5	2
					1	8	14	0
					1	3	6	1
					1	1	13	3
b	e e 6	×	√	×	1	0	5	3
					1	8	14	1
					1	3	6	2
					1	b	12	0
3	1 4 0	√	√	×	1	0	5	4
					1	8	14	2
					1	3	6	0
					1	b	12	1
c	0 4 9	×	×	√	1	c	15	0
					1	8	14	3
					1	3	6	1
					1	b	12	2

- TLB 是否失效，只需要查看是否在上一行的 Tag 中出现过；
- 页表是否失效，只需要看标签是否在页表中出现；
- 如果某个标签在磁盘中，或者不在页表中，都会造成缺页错误；
- 每次访问后，TLB 中的上次访问时间间隔都需要加一（用蓝色表示），如果 TLB 未命中，则需要替换上次访问时间间隔最大的那一行（用红色表示），不管是否命中都需要把当前访问到的 Tag 所在的那行的上次访问时间间隔改为 0（用红色表示）；
- 题目中的“如果必须从磁盘中取回页，请增加下一次能取的最大页码”的意思是发生缺页错误时，分配的物理页号是当前最大的物理页号加一。
- 答案中的 last access 是相对顺序，每次替换序号最小的一行，而这里是指访问时间间隔，所以每次替换最大的一行。

5.16.2 [15]<5.7> 重复 5.16.1，但这次使用 16KiB 页而不是 4KiB 页。拥有更大页大小的优势是什么？有什么缺点？

16KiB = 2^{14}Bytes，所以右侧 14 位二进制位表示页内偏移，左侧 2 位二进制位表示页号。

Tag	Offset binary	Offset hexadecimal	TLB Hit	PT Hit	Page Fault	V	Tag	Physical Page Number	Last Access Interval
0 0	0 1	23d	×	√	×	1	b	12	
						1	7	4	
						1	3	6	
						1	0	5	
0 0	0 0	8b3	√	√	×	1	b	12	
						1	7	4	
						1	3	6	
						1	0	5	
0 0	1 1	65c	√	√	×	1	b	12	
						1	7	4	
						1	3	6	
						1	0	5	
1 0	0 0	71b	×	√	√	1	2	13	
						1	7	4	
						1	3	6	
						1	0	5	
1 0	1 1	ee6	√	√	×	1	2	13	
						1	7	4	
						1	3	6	
						1	0	5	
0 0	1 1	140	√	√	×	1	2	13	
						1	7	4	
						1	3	6	
						1	0	5	
1 1	0 0	049	√	√	×	1	2	13	
						1	7	4	
						1	3	6	
						1	0	5	

由于标签只有 2 位，所以一共 4 行的 TLB 不会出现很多次置换，这里的上次访问间隔就省略了（直接按照原始的 TLB，先置换第 4 行，再置换第 1 行，再置换第 3 行，再置换第 2 行），每次置换的行仍然用红色表示。

拥有更大页大小的优势是有更高的快表命中率，缺点是会降低内存使用率（产生了更多内零头）。

5.16.3 [15]<5.7> 重复 5.16.1，但这次使用 4KiB 页和一个两路组相联 TLB。

一个页是 4KiB，两个页组成一行。右侧 12 位表示页内偏移，中间 1 位表示索引，左侧 3 位表示标签。页号仍然是左侧 4 位。

Page Number	Address			Physical Page Number	TLB Hit	PT Hit	Page Fault	Way 1				Way 2			
	Tag	Index	Offset					V	Tag	Physical Page Number	Last Access Interval	V	Tag	Physical Page Number	Last Access Interval
1	000	1	23d	13	×	√	√	1	b	12	4	1	3	6	3
								1	7	4	1	1	0	13	0
0	000	0	8b3	5	×	√	×	1	0	5	0	1	3	6	4
								1	7	4	2	1	0	13	1
3	001	1	65c	6	×	√	×	1	0	5	1	1	3	6	5
								1	1	6	0	1	0	13	2
8	100	0	71b	14	×	√	√	1	0	5	2	1	4	14	0
								1	1	6	1	1	0	13	3
b	101	1	ee6	12	×	√	×	1	0	5	3	1	4	14	1
								1	1	6	2	1	5	12	0
3	001	1	140	6	√	√	×	1	0	5	4	1	4	14	2
								1	1	6	0	1	5	12	1
c	110	0	049	15	×	×	√	1	6	15	0	1	4	14	3
								1	1	6	1	1	5	12	2

5.16.4 [15]<5.7> 重复 5.16.1，但这次使用 4KiB 页和一个直接映射 TLB。

一个页是 4KiB，右侧 12 位表示页内偏移，中间 2 位表示索引，左侧 2 位表示标签。页号仍然是左侧 4 位。

Page Number	Physical Page Number	Tag	Index	Offset	TLB Hit	PT Hit	Page Fault	V	Tag	Physical Page Number
1	13	00	01	23d	×	√	√	1	b	12
								1	0	13
								1	3	6
								0	4	9
0	5	00	00	8b3	×	√	×	1	0	5
								1	0	13
								1	3	6
								0	4	9
3	6	00	11	65c	×	√	×	1	0	5
								1	0	13
								1	3	6
								1	0	6
8	14	10	00	71b	×	√	√	1	2	14
								1	0	13
								1	3	6
								1	0	6
b	12	10	11	ee6	×	√	×	1	2	14
								1	0	13
								1	3	6
								1	2	12
3	6	00	11	140	×	√	×	1	2	14
								1	0	13
								1	3	6
								1	0	6
c	15	11	00	049	×	×	√	1	3	15
								1	0	13
								1	3	6
								1	0	6

5.16.5 [10]<5.4,5.7> 讨论为什么 CPU 必须使用 TLB 才能实现高性能。如果没有 TLB，如何处理虚拟内存访问?

为了便于编写程序，写代码时不需要指定某段数据放在哪个物理地址中，出现了虚拟地址，为了将虚拟地址存放到实际的物理地址中，需要有个表存放这个映射关系，这就是页表，而页表存放在内存中，访问比较慢。但是每次访问虚拟地址时，都需要访问一次页表再访问实际的数据，也就是访问两次内存，所以就出现了 TLB（快表）作为页表的缓存，在 TLB 命中时只需要访问一次内存，从而提升性能。

如果没有 TLB，每次虚拟内存访问就需要访问两次内存，第一次访问页表，第二次再访问实际的数据。