习题3.8

3.8 ［5］<3.2>假定带符号的 8 位十进制整数 185 和 122 以符号 – 数值形式存储。计算 185–122。是否上溢或下溢，或都没有？

习题3.8 - 讲解

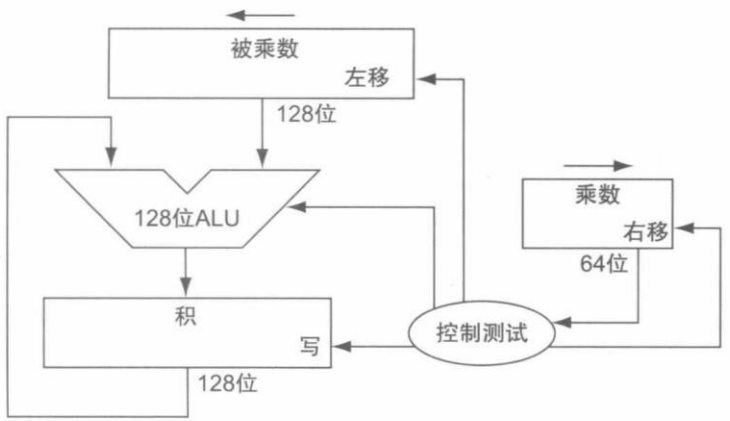
7个二进制位能表示的最大绝对值是2的7次方减1，即127。 此时，数值122在范围内（122小于等于127）。 但是，数值185超出了7位二进制能表示的最大范围127。因此，185无法用8位带符号的符号-数值形式正确存储，在输入阶段就已经发生了上溢。

习题3.12

3.12 ［20］<3.3>使用类似于图 3-6 所示的表格，使用图 3-3 中的硬件描述计算八进制无符号 6 位整数 62 和 12 的乘积。须在每步中写出各个寄存器的内容。

迭代次数	步骤	乘数	被乘数	积
0	初始值	001(1)	0000 0010	0000 0000
1	1a：1=> 积 = 积 + 被乘数	0011	0000 0010	0000 0010
	2 ：被乘数左移	0011	0000 0100	0000 0010
	3 ：乘数右移	0001	0000 0100	0000 0010
2	1a：1=> 积 = 积 + 被乘数	0001	0000 0100	0000 0110
	2 ：被乘数左移	0001	0000 1000	0000 0110
	3 ：乘数右移	0000	0000 1000	0000 0110
3	1 ：0=> 无操作	0000	0000 1000	0000 0110
	2 ：被乘数左移	0000	0001 0000	0000 0110
	3 ：乘数右移	0000	0001 0000	0000 0110
4	1 ：0=>无操作	0000	0001 0000	0000 0110
	2 ：被乘数左移	0000	0010 0000	0000 0110
	3 ：乘数右移	0000	0010 0000	0000 0110

图 3-6　采用图 3-4 算法的乘法举例。圆圈圈起来的是决定下一步执行的待检测位



第一版乘法器硬件。被乘数寄存器、ALU 和乘积寄存器都是 128 位长，只有乘数寄

助教的求解结果（供参考）

迭代次数	步骤	乘数 (6位)	被乘数 (12位)	积 (12位)
0	初始值	00101(0)	000000 110010	0000 0000 0000
1	1: 0 => 无操作	001010	000000 110010	0000 0000 0000
	2: 被乘数左移	001010	000001 100100	0000 0000 0000
	3: 乘数右移	00010(1)	000001 100100	0000 0000 0000
2	1a: 1 => 积 = 积 + 被乘数	000101	000001 100100	0000 0110 0100
	2: 被乘数左移	000101	000011 001000	0000 0110 0100
	3: 乘数右移	00001(0)	000011 001000	0000 0110 0100
3	1: 0 => 无操作	000010	000011 001000	0000 0110 0100
	2: 被乘数左移	000010	001110 010000	0000 0110 0100
	3: 乘数右移	00000(1)	001110 010000	0000 0110 0100
4	1a: 1 => 积 = 积 + 被乘数	000001	001110 010000	0001 1111 0100
	2: 被乘数左移	000001	001100 100000	0001 1111 0100
	3: 乘数右移	00000(0)	001100 100000	0001 1111 0100
5	1: 0 => 无操作	000000	001100 100000	0001 1111 0100
	2: 被乘数左移	000000	011001 000000	0001 1111 0100
	3: 乘数右移	00000(0)	011001 000000	0001 1111 0100
6	1: 0 => 无操作	000000	011001 000000	0001 1111 0100
	2: 被乘数左移	000000	110010 000000	0001 1111 0100
	3: 乘数右移	000000	110010 000000	0001 1111 0100

习题3.14

3.14 [10] <3.3> 如果一个整数是 8 位宽且每步操作需要 4 个时间单位，使用图 3-3 和图 3-4 中给出的方法计算执行一次乘法所需的时间。假设在步骤 1a 中总是执行加法，无论是加上被乘数还是零。另外假设寄存器已经被初始化（你只需要计算执行乘法循环本身需要多长时间）。如果是在硬件中执行，则可以同时完成被乘数和乘数的移位。如果这是在软件中执行，则必须依次完成。给出每种情况的解答。

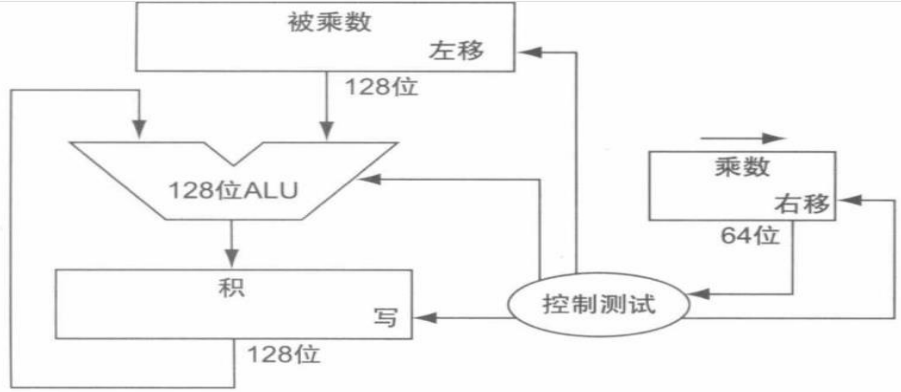


图 3-3　第一版乘法器硬件。被乘数寄存器、ALU 和乘积寄存器都是 128 位长，只有乘数寄存器是 64 位。（附录 A 描述了 ALU。）64 位被乘数最初存放在被乘数寄存器的右半部分，每一步左移 1 位。乘数在每步以相反方向移位。算法开始前，积初始化为 0。控制部件决定何时对被乘数寄存器和乘数寄存器进行移位，以及何时将新值写入积寄存器

图 3-4 显示了对于操作数的每一位都需要做的三个基本步骤。第一步中的乘数最低位（乘数第 0 位）决定了是否要把被乘数加到积寄存器当中。第二步中的左移起着将中间操作数左移的作用，就像手工纸笔做乘法一样。第三步中的右移给出了下次迭代要检测的乘数的下一位。这三个步骤重复 64 次就会得到最后的积。如果每个步骤花费一个时钟周期，那么该算法计算两个 64 位数相乘差不多要花费 200 个时钟周期。像乘法这样的算术运算的重要性随程序的不同而变化，但一般加法和减法出现的次数会是乘法的 5 到 100 倍。因此，在许多应用中，乘法花费若干时钟周期并不会显著影响性能。但是，Amdahl 定律（参见 1.10 节）提醒我们，一个慢速操作如果占据

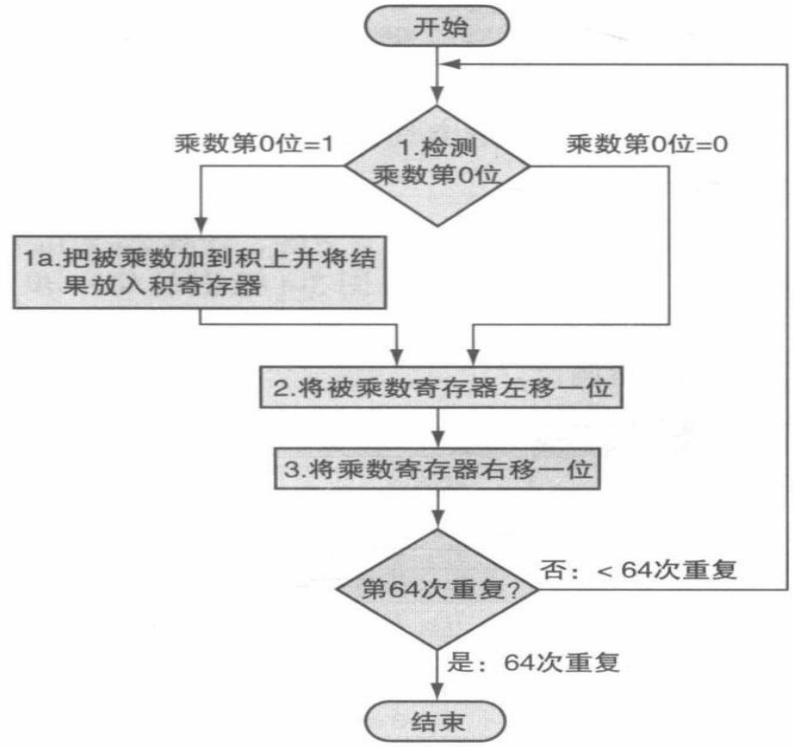


图 3-4　第一种乘法算法，采用了图 3-3 所示的硬件。

根据题意，8位整数乘法需要循环执行8次。由图3-4及文字说明可知，该算法每轮循环包含3个基本步骤：加法操作（步骤1和1a）、被乘数左移（步骤2）、乘数右移（步骤3）。每个步骤需要4个时间单位。

情况一：在硬件中执行

被乘数和乘数的移位（步骤2和步骤3）可以同时完成，即合并为一个步骤的时间。

每轮循环时间 = (1个加法步骤 + 1个并行移位步骤) * 4 = 8 个时间单位

总时间 = 8次循环 * 8个时间单位 = 64 个时间单位

情况二：在软件中执行

所有步骤必须依次顺序完成。

每轮循环时间 = (1个加法步骤 + 1个被乘数移位步骤 + 1个乘数移位步骤) * 4 = 12 个时间单位

总时间 = 8次循环 * 12个时间单位 = 96 个时间单位

习题3.16

3.16 [20] <3.3> 如果一个整数是 8 位宽且一次加法需要 4 个时间单位，计算使用图 3-7 给出的方法执行乘法所需的时间。

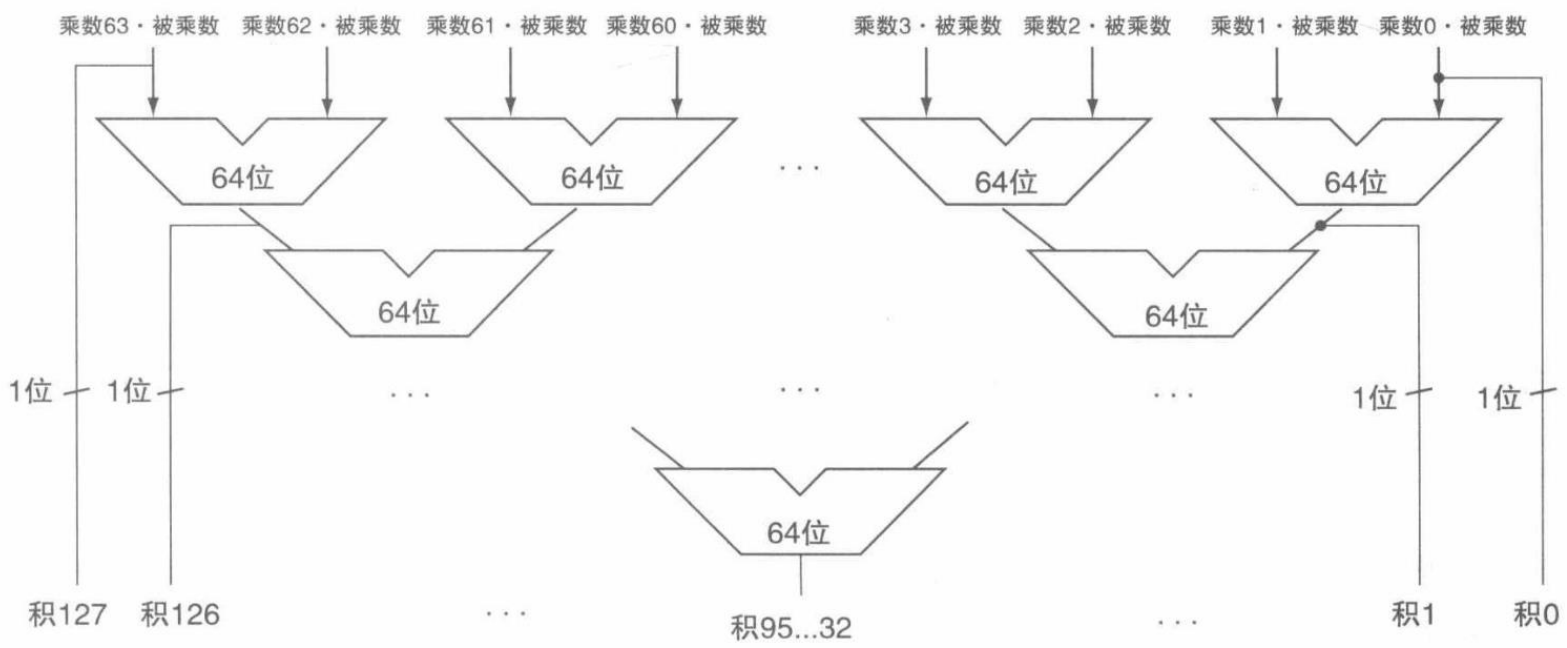


图 3-7 快速乘法器硬件。这个硬件“展开循环”来使用 63 个加法器，并协调它们实现最小化时延，而不是使用单个 64 位的加法器 63 次

对于 n 个部分积：
•需要 n−1 个加法器；
•关键路径长度为 $\log_2 n$ 层加法器；
•总时间 = （加法树高度）×（一次加法时间）。
本题中：
•乘数和被乘数均为 8 位；
•因此有 8 个部分积；
•加法树高度为
•$\log_2(8) = 3$
•一次加法需要 4 个时间单位。
所以乘法所需时间为
$3 \times 4 = 12$
个时间单位。
另外，8 个部分积对应的加法器总数为
$8 - 1 = 7$
个，但这些加法器分布在 3 层中并行工作，因此时间不是 7×4，而是由树的高度决定。
答案：12 个时间单位。

习题5.4

5.4 [15] <5.3> 在 5.3 节中显示了索引直接映射 cache 的典型方法：（块地址）mod（cache 中的块数）。假设 cache 地址为 64 位，有 1024 个块，考虑一个不同的索引函数：（块地址 [63:54] XOR 块地址 [53:44]）。是否可以使用这个索引函数来索引直接映射 cache？如果可以，请解释原因并讨论可能需要对 cache 进行的任何更改。如果不可以，请解释原因。

设块地址为 A，共 64 位，cache 有 1024 = 2^10 个块，因此索引需要 10 位。
题目给出的索引函数：
I = A[63:54] ⊕ A[53:44]
其中 A[63:54]、A[53:44] 都是 10 位，所以 I 也是 10 位，能够唯一选出 1024 个 cache 行中的一行。
因此从定义上讲，仍然是
A → I
的确定映射，每个主存块只对应一个 cache 行，所以可以用于直接映射 cache。
关键在于 Tag。
传统直接映射中：
块地址 = Tag || Index
访问时根据 Index 找到一行，再比较 Tag。
而这里
I = A[63:54] ⊕ A[53:44]
索引值只保留了 10 位信息，而参与计算的地址位共有 20 位，因此由 I 无法恢复这 20 位：
(A[63:54], A[53:44]) → I
是多对一映射。
因此这 20 位不能像传统方案那样部分作为 Index 丢弃，而必须保存在 Tag 中用于比较。
故需要的修改：
1.在索引路径增加 10 位 XOR 电路，计算 I。
2.扩大 Tag，至少要包含参与异或的地址位中未被索引保留的信息。
结论：
可以实现直接映射 cache，因为索引函数仍能唯一确定 cache 行；但需要增加 XOR 索引逻辑，并增大 Tag 存储与比较位数。

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）？

5.5.2 [5] <5.3>cache 块有多少个？

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？

下表记录了从上电开始 cache 访问的字节地址。

	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

5.5.4 [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）。

5.5.5 [5] <5.3> 命中率是多少？

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]>
```

已知地址划分：
•Tag：63~10（54位）
•Index：9~5（5位）
•Offset：4~0（5位）

5.5.1 cache块大小

偏移字段5位：
块大小 = 2^5 = 32B
按字（4B）计：
32/4 = 8字
答案：8字

5.5.2 cache块数

索引字段5位：
块数 = 2^5 = 32
答案：32块

5.5.3 总位数/数据位

每块数据：
32B = 256bit
每块额外需要：
Tag = 54bit
Valid = 1bit
共：
256 + 54 + 1 = 311bit
32块总容量：
数据位 = 32×256 = 8192bit
总位数 = 32×311 = 9952bit
比率：
9952/8192 = 1.2148
≈1.215
答案：9952:8192≈1.215:1

5.5.4 逐次访问

由于

Offset = 地址 mod 32

Index = (地址/32) mod 32

Tag = 地址/1024

地址	Tag	Index	Offset	结果
0x00	0	0	0	Miss
0x04	0	0	4	Hit
0x10	0	0	16	Hit
0x84	0	4	4	Miss
0xE8	0	7	8	Miss
0xA0	0	5	0	Miss
0x400	1	0	0	Miss，替换块0~31
0x1E	0	0	30	Miss，替换块1024~1055
0x8C	0	4	12	Hit
0xC1C	3	0	28	Miss，替换块0~31
0xB4	0	5	20	Hit
0x884	2	4	4	Miss，替换块128~159

5.5.5 命中率

命中：
•0x04
•0x10
•0x8C
•0xB4
共4次
总访问：
12次
命中率：
4/12 = 1/3 = 33.3%
答案：33.3%

5.5.6 最终cache状态

只列有效项：
<0,3, Mem[0xC00]-Mem[0xC1F]>
<4,2, Mem[0x880]-Mem[0x89F]>
<5,0, Mem[0xA0]-Mem[0xBF]>
<7,0, Mem[0xE0]-Mem[0xFF]>
其余28项无效。

5.8 播放音频或视频文件的媒体应用程序是称为“流”工作负载的一类工作负载的一部分（也就是说，它们带来大量数据但是不重用大部分数据）。考虑使用以下字地址流顺序访问 512KiB 工作集的视频流工作负载：

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ...
```

5.8.1 [10] <5.4, 5.8> 假设有一个块大小为 32 字节的 64KiB 直接映射 cache，那么上述地址流的失效率是多少？ cache 或工作集的大小变化时，cache 失效率如何变化？基于 3C 模型，如何对这个工作负载所经历的这些失效进行分类？

5.8.2 [5] <5.1, 5.8> 当 cache 块大小为 16 字节、64 字节和 128 字节时，重新计算失效率。这个工作负载利用了什么类型的局部性？

5.8.3 [10] <5.13> “预取”是一种利用可预测的地址模式在访问特定 cache 块时推测性地取回额外 cache 块的技术。预取的一个示例是流缓冲区，当取回特定的 cache 块时，它将相邻的 cache 块顺序预取到单独的缓冲区中。如果在预取缓冲区中找到数据，则将其视为命中，移入 cache 中，同时预取下一个 cache 块。假设流缓冲区有 2 个表项，并且假设 cache 延迟满足可以在完成对先前 cache 块的计算之前加载 cache 块，那么上述地址流的失效率是多少？

5.8.1
块大小32B，**工作集**512KiB。
工作集块数：
512KiB / 32B = 16384块
顺序访问时，每个块只会被访问一次。进入一个新块时第**1个字**节失效，后面**31个字**节命中。
失效数 = 16384
总访问数 = 512×1024 = 524288
失效率：
16384 / 524288 = 1/32 = 3.125%
若改变Cache**容量或工作集容量，只要仍是一次**顺序扫描且数据不重用，则每块仍只有**1次**强制失效，因**此失效率仍**为1/32，**与**Cache**容量基本无关**。
3C**分**类：全部属于强制失效，**没有** Capacity Miss，**也没有** Conflict Miss。
5.8.2
块大小为B时，每块产生**1次失效**，**因此**：
失效率 = 1/B
16B块：1/16 = 6.25%
64B块：1/64 = 1.5625%
128B块：1/128 = 0.78125%
块越大，失效率越低，因为一次失效带入更多后续将被访问的数据。
该工作负载利用的是空间局部性，几乎不利用时间局部性。
5.8.3
采用2项流缓冲区。
访问第**1个**块B_0时发生失效，同时预取B_1；访问B_1时在流缓冲区命中，并继续预取B_2；**之后同理，所有后**续块都可从流缓冲区获得。
因此16384**个**块中只有第**1个**块失效。
失效数 = 1
总访问数 = 524288
失效率：
1 / 524288 ≈ 1.91×10^{-6} ≈ 0.0001907%
即约为0.00019%。**从**块的角度看，16384**个**块仅第1块失效，其余16383块均由预取命中。

习题5.11

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, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f,
0xb5, 0xbf, 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。

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。

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。

5.11.1
块大小=2字，总容量=48字，三路组相联。
块数=48/2=24
组数=24/3=8=2^3
偏移位=$\log_2 2$=1
索引位=$\log_2 8$=3
地址最大为0xCE(<256)，故字地址为8位。
Tag位数=8−3−1=4
地址格式：
Tag(4) | Index(3) | Offset(1)
Cache结构：8组×3路，每块2字。图略…

5.11.2
计算公式：
Tag=A>>4
Index=(A>>1) mod 8
Offset=A mod 2

5.11.3
偏移位=0索引位=0
Tag位数=8
地址格式：
Tag(8)
Cache结构：
8路全相联，每行：
Valid | Tag(8) | Data(1字)
访问时8个Tag并行比较，采用LRU替换策略。图略。

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 中的内容。

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。

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 中的内容。

5.11.7 [10] <5.4> 将替换策略改为 MRU（最多最常使用）策略，再次完成 5.11.6。

5.11.8 [15] <5.4> 将替换策略改为最优替换策略（造成最低失效率的替换策略），再次完成 5.11.6。

5.11.4
按照LRU策略，逐步进行即可。
5.11.5
还是画图题，参考题目中指代的图即可。
5.11.6
这题和前面的雷同，只是把**cache**结构微调了，都是一样的做法。
5.11.7和5.11.8
两道题都是改策略后，重做**5.11.6**，**所以都几乎一**样。